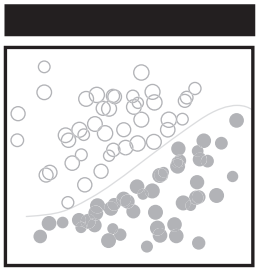
DEAN ABBOTT

APPLIED PREDICTIVE ANALYTICS

PRINCIPLES AND TECHNIQUES FOR THE PROFESSIONAL DATA ANALYST

WILEY



Applied Predictive Analytics

Principles and Techniques for the Professional Data Analyst

Dean Abbott

WILEY

Applied Predictive Analytics: Principles and Techniques for the Professional Data Analyst

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

ISBN: 978-1-118-72796-6
ISBN: 978-1-118-72793-5 (ebk)
ISBN: 978-1-118-72769-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2013958302

To Barbara

About the Author

Dean Abbott is President of Abbott Analytics, Inc. in San Diego, California. Dean is an internationally recognized data-mining and predictive analytics expert with over two decades of experience applying advanced modeling and data preparation techniques to a wide variety of real-world problems. He is also Chief Data Scientist of SmarterRemarketer, a startup company focusing on data-driven behavior segmentation and attribution.

Dean has been recognized as a top-ten data scientist and one of the top ten most influential people in data analytics. His blog has been recognized as one of the top-ten predictive analytics blogs to follow.

He is a regular speaker at Predictive Analytics World and other analytics conferences. He is on the advisory board for the University of California Irvine Certificate Program for predictive analytics and the University of California San Diego Certificate Program for data mining, and is a regular instructor for courses on predictive modeling algorithms, model deployment, and text mining. He has also served several times on the program committee for the KDD Conference Industrial Track.

About the Technical Editor

William J. Komp has a Ph.D. from the University of Wisconsin–Milwaukee, with a specialization in the fields of general relativity and cosmology. He has been a professor of physics at the University of Louisville and Western Kentucky University. Currently, he is a research scientist at Humana, Inc., working in the areas of predictive analytics and data mining.

Credits

Executive Editor
Robert Elliott

Project Editor
Adaobi Obi Tulton

Technical Editor
William J. Komp

Senior Production Editor
Kathleen Wisor

Copy Editor
Nancy Rapoport

Manager of Content Development and Assembly
Mary Beth Wakefield

Director of Community Marketing
David Mayhew

Marketing Manager
Ashley Zurcher

Business Manager
Amy Knies

Vice President and Executive Group Publisher
Richard Swadley

Associate Publisher
Jim Minatel

Project Coordinator, Cover
Todd Klemme

Proofreader
Nancy Carrasco

Indexer
Johnna VanHoose Dinse

Cover Designer
Ryan Sneed

Acknowledgments

The idea for this book began with a phone call from editor Bob Elliott, who presented the idea of writing a different kind of predictive analytics book geared toward business professionals. My passion for more than a decade has been to teach principles of data mining and predictive analytics to business professionals, translating the lingo of mathematics and statistics into a language the practitioner can understand. The questions of hundreds of course and workshop attendees forced me to think about why we do what we do in predictive analytics. I also thank Bob for not only persuading me that I could write the book while continuing to be a consultant, but also for his advice on the scope and depth of topics to cover.

I thank my father for encouraging me in analytics. I remember him teaching me how to compute batting average and earned run average when I was eight years old so I could compute my Little League statistics. He brought home reams of accounting pads, which I used for playing thousands of Strat-O-Matic baseball games, just so that I could compute everyone's batting average and earned run average, and see if there were significant differences between what I observed and what the players' actual statistics were. My parents put up with a lot of paper strewn across the floor for many years.

I would never have been in this field were it not for Roger Barron, my first boss at Barron Associates, Inc., a pioneer in statistical learning methods and a man who taught me to be curious, thorough, and persistent about data analysis. His ability to envision solutions without knowing exactly how they would be solved is something I reflect on often.

I've learned much over the past 27 years from John Elder, a friend and colleague, since our time at Barron Associates, Inc. and continuing to this day. I am very grateful to Eric Siegel for inviting me to speak regularly at Predictive

Analytics World in sessions and workshops, and for his advice and encouragement in the writing of this book.

A very special thanks goes to editors Adaobi Obi Tulton and Nancy Rapoport for making sense of what of I was trying to communicate and making this book more concise and clearer than I was able to do alone. Obviously, I was a mathematics major and not an English major. I am especially grateful for technical editor William Komp, whose insightful comments throughout the book helped me to sharpen points I was making.

Several software packages were used in the analyses contained in the book, including KNIME, IBM SPSS Modeler, JMP, Statistica, Predixion, and Orange. I thank all of these vendors for creating software that is easy to use. I also want to thank all of the other vendors I've worked with over the past two decades, who have supplied software for me to use in teaching and research.

On a personal note, this book project could not have taken place without the support of my wife, Barbara, who encouraged me throughout the project, even as it wore on and seemed to never end. She put up with me as I spent countless nights and Saturdays writing, and cheered with each chapter that was submitted. My two children still at home were very patient with a father who wasn't as available as usual for nearly a year.

As a Christian, my worldview is shaped by God and how He reveals Himself in the Bible, through his Son, and by his Holy Spirit. When I look back at the circumstances and people He put into my life, I only marvel at His hand of providence, and am thankful that I've been able to enjoy this field for my entire career.

Contents

Introduction

The methods behind predictive analytics have a rich history, combining disciplines of mathematics, statistics, social science, and computer science. The label "predictive analytics" is relatively new, however, coming to the forefront only in the past decade. It stands on the shoulders of other analytics-centric fields such as data mining, machine learning, statistics, and pattern recognition.

This book describes the predictive modeling process from the perspective of a practitioner rather than a theoretician. Predictive analytics is both science and art. The science is relatively easy to describe but to do the subject justice requires considerable knowledge of mathematics. I don't believe a good practitioner needs to understand the mathematics of the algorithms to be able to apply them successfully, any more than a graphic designer needs to understand the mathematics of image correction algorithms to apply sharpening filters well. However, the better practitioners understand the effects of changing modeling parameters, the implications of the assumptions of algorithms in predictions, and the limitations of the algorithms, the more successful they will be, especially in the most challenging modeling projects.

Science is covered in this book, but not in the same depth you will find in academic treatments of the subject. Even though you won't find sections describing how to decompose matrices while building linear regression models, this book does not treat algorithms as black boxes.

The book describes what I would be telling you if you were looking over my shoulder while I was solving a problem, especially when surprises occur in the data. How can algorithms fool us into thinking their performance is excellent when they are actually brittle? Why did I bin a variable rather than just transform it numerically? Why did I use logistic regression instead of a neural network, or *vice versa*? Why did I build a linear regression model to predict a

binary outcome? These are the kinds of questions, the art of predictive modeling, that this book addresses directly and indirectly.

I don't claim that the approaches described in this book represent the *only* way to solve problems. There are many contingencies you may encounter where the approaches described in this book are not appropriate. I can hear the comments already from other experienced statisticians or data miners asking, "but what about . . . ?" or "have you tried doing this . . . ?" I've found over the years that there are often many ways to solve problems successfully, and the approach you take depends on many factors, including your personal style in solving problems and the desires of the client for how to solve the problems. However, even more often than this, I've found that the biggest gains in successful modeling come more from understanding data than from applying a more sophisticated algorithm.

How This Book Is Organized

The book is organized around the Cross-Industry Standard Process Model for Data Mining (CRISP-DM). While the name uses the term "data mining," the steps are the same in predictive analytics or any project that includes the building of predictive or statistical models.

This isn't the only framework you can use, but it a well-documented framework that has been around for more than 15 years. CRISP-DM should not be used as a recipe with checkboxes, but rather as a guide during your predictive modeling projects.

The six phases or steps in CRISP-DM:

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Deployment

After an introductory chapter, Business Understanding, Data Understanding, and Data Preparation receive a chapter each in Chapters 2, 3, and 4. The Data Understanding and Data Preparation chapters represent more than one-quarter of the pages of technical content of the book, with Data Preparation the largest single chapter. This is appropriate because preparing data for predictive modeling is nearly always considered the most time-consuming stage in a predictive modeling project.

Chapter 5 describes association rules, usually considered a modeling method, but one that differs enough from other descriptive and predictive modeling techniques that it warrants a separate chapter.

The modeling methods—descriptive, predictive, and model ensembles—are described in Chapters 6, 8, and 10. Sandwiched in between these are the modeling evaluation or assessment methods for descriptive and predictive modeling, in Chapters 7 and 9, respectively. This sequence is intended to connect more directly the modeling methods with how they are evaluated.

Chapter 11 takes a side-turn into text mining, a method of analysis of unstructured data that is becoming more mainstream in predictive modeling. Software packages are increasingly including text mining algorithms built into the software or as an add-on package. Chapter 12 describes the sixth phase of predictive modeling: Model Deployment.

Finally, Chapter 13 contains two case studies: one based on work I did as a contractor to Seer Analytics for the YMCA, and the second for a Fortune 500 company. The YMCA case study is actually two case studies built into a single narrative because we took vastly different approaches to solve the problem. The case studies are written from the perspective of an analyst as he considers different ways to solve the problem, including solutions that were good ideas but didn't work well.

Throughout the book, I visit many of the problems in predictive modeling that I have encountered in projects over the past 27 years, but the list certainly is not exhaustive. Even with the inherent limitations of book format, the thought process and principles are more important than developing a complete list of possible approaches to solving problems.

Who Should Read This Book

This book is intended to be read by anyone currently in the field of predictive analytics or its related fields, including data mining, statistics, machine learning, data science, and business analytics.

For those who are just starting in the field, the book will provide a description of the core principles every modeler needs to know.

The book will also help those who wish to enter these fields but aren't yet there. It does not require a mathematics background beyond pre-calculus to understand predictive analytics methods, though knowledge of calculus and linear algebra will certainly help. The same applies to statistics. One can understand the material in this book without a background in statistics; I, for one, have never taken a course in statistics. However, understanding statistics certainly provides additional intuition and insight into the approaches described in this book.

Tools You Will Need

No predictive modeling software is needed to understand the concepts in the book, and no examples are shown that require a particular predictive analytics software package. This was deliberate because the principles and techniques described in this book are general, and apply to any software package. When applicable, I describe which principles are usually available in software and which are rarely available in software.

What's on the Website

All data sets used in examples throughout the textbook are included on the Wiley website at `www.wiley.com/go/appliedpredictiveanalytics` and in the Resources section of `www.abbottanalytics.com`. These data sets include:

- KDD Cup 1998 data, simplified from the full version
- Nasadata data set
- Iris data set

In addition, baseline scripts, workflows, streams, or other documents specific to predictive analytics software will be included as they become available. These will provide baseline processing steps to replicate analyses from the book chapters.

Summary

My hope is that this book will encourage those who want to be more effective predictive modelers to continue to work at their craft. I've worked side-by-side with dozens of predictive modelers over the years.

I always love watching how effective predictive modelers go about solving problems, perhaps in ways I had never considered before. For those with more experience, my hope is that this book will describe data preparation, modeling, evaluation, and deployment in a way that even experienced modelers haven't thought of before.

CHAPTER

1

Overview of Predictive Analytics

A small direct response company had developed dozens of programs in cooperation with major brands to sell books and DVDs. These affinity programs were very successful, but required considerable up-front work to develop the creative content and determine which customers, already engaged with the brand, were worth the significant marketing spend to purchase the books or DVDs on subscription. Typically, they first developed test mailings on a moderately sized sample to determine if the expected response rates were high enough to justify a larger program.

One analyst with the company identified a way to help the company become more profitable. What if one could identify the key characteristics of those who responded to the test mailing? Furthermore, what if one could generate a score for these customers and determine what minimum score would result in a high enough response rate to make the campaign profitable? The analyst discovered predictive analytics techniques that could be used for both purposes, finding key customer characteristics and using those characteristics to generate a score that could be used to determine which customers to mail.

Two decades before, the owner of a small company in Virginia had a compelling idea: Improve the accuracy and flexibility of guided munitions using optimal control. The owner and president, Roger Barron, began the process of deriving the complex mathematics behind optimal control using a technique known as variational calculus and hired a graduate student to assist him in the task. Programmers then implemented the mathematics in computer code so

they could simulate thousands of scenarios. For each trajectory, the variational calculus minimized the miss distance while maximizing speed at impact as well as the angle of impact.

The variational calculus algorithm succeeded in identifying the optimal sequence of commands: how much the fins (control surfaces) needed to change the path of the munition to follow the optimal path to the target. The concept worked in simulation in the thousands of optimal trajectories that were run. Moreover, the mathematics worked on several munitions, one of which was the MK82 glide bomb, fitted (in simulation) with an inertial guidance unit to control the fins: an early smart-bomb.

There was a problem, however. The variational calculus was so computationally complex that the small computers on-board could not solve the problem in real time. But what if one could *estimate* the optimal guidance commands at any time during the flight from observable characteristics of the flight? After all, the guidance unit can compute where the bomb is in space, how fast it is going, and the distance of the target that was programmed into the unit when it was launched. If the estimates of the optimum guidance commands were close enough to the actual optimal path, it would be *near optimal* and still succeed. Predictive models were built to do exactly this. The system was called *Optimal Path-to-Go* guidance.

These two programs designed by two different companies seemingly could not be more different. One program knows characteristics of people, such as demographics and their level of engagement with a brand, and tries to predict a human decision. The second program knows locations of a bomb in space and tries to predict the best physical action for it to hit a target.

But they share something in common: They both need to estimate values that are unknown but tremendously useful. For the affinity programs, the models estimate whether or not an individual will respond to a campaign, and for the guidance program, the models estimate the best guidance command. In this sense, these two programs are very similar because they both involve predicting a value or values that are known historically, but are unknown at the time a decision is needed. Not only are these programs related in this sense, but they are far from unique; there are countless decisions businesses and government agencies make every day that can be improved by using historic data as an aid to making decisions or even to automate the decisions themselves.

This book describes the back-story behind how analysts build the predictive models like the ones described in these two programs. There is science behind much of what predictive modelers do, yet there is also plenty of *art*, where no theory can inform us as to the best action, but experience provides principles by which tradeoffs can be made as solutions are found. Without the art, the science would only be able to solve a small subset of problems we face. Without

the science, we would be like a plane without a rudder or a kite without a tail, moving at a rapid pace without any control, unable to achieve our objectives.

What Is Analytics?

Analytics is the process of using computational methods to discover and report influential patterns in data. The goal of analytics is to gain insight and often to affect decisions. Data is necessarily a measure of historic information so, by definition, analytics examines historic data. The term itself rose to prominence in 2005, in large part due to the introduction of Google Analytics. Nevertheless, the ideas behind analytics are not new at all but have been represented by different terms throughout the decades, including *cybernetics*, *data analysis*, *neural networks*, *pattern recognition*, *statistics*, *knowledge discovery*, data *mining*, and now even *data science*.

The rise of analytics in recent years is pragmatic: As organizations collect more data and begin to summarize it, there is a natural progression toward using the data to improve estimates, forecasts, decisions, and ultimately, efficiency.

What Is Predictive Analytics?

Predictive analytics is the process of discovering interesting and meaningful patterns in data. It draws from several related disciplines, some of which have been used to discover patterns in data for more than 100 years, including pattern recognition, statistics, machine learning, artificial intelligence, and data mining. What differentiates predictive analytics from other types of analytics?

First, predictive analytics is data-driven, meaning that algorithms derive key characteristic of the models from the data itself rather than from assumptions made by the analyst. Put another way, data-driven algorithms *induce* models from the data. The induction process can include identification of variables to be included in the model, parameters that define the model, weights or coefficients in the model, or model complexity.

Second, predictive analytics algorithms automate the process of finding the patterns from the data. Powerful induction algorithms not only discover coefficients or weights for the models, but also the very form of the models. Decision trees algorithms, for example, learn which of the candidate inputs best predict a target variable in addition to identifying which values of the variables to use in building predictions. Other algorithms can be modified to perform searches, using exhaustive or greedy searches to find the best set of inputs and model parameters. If the variable helps reduce model error, the variable is included

in the model. Otherwise, if the variable does not help to reduce model error, it is eliminated.

Another automation task available in many software packages and algorithms automates the process of transforming input variables so that they can be used effectively in the predictive models. For example, if there are a hundred variables that are candidate inputs to models that can be or should be transformed to remove skew, you can do this with some predictive analytics software in a single step rather than programming all one hundred transformations one at a time.

Predictive analytics doesn't do anything that any analyst couldn't accomplish with pencil and paper or a spreadsheet if given enough time; the algorithms, while powerful, have no common sense. Consider a supervised learning data set with 50 inputs and a single binary target variable with values 0 and 1. One way to try to identify which of the inputs is most related to the target variable is to plot each variable, one at a time, in a histogram. The target variable can be superimposed on the histogram, as shown in Figure 1-1. With 50 inputs, you need to look at 50 histograms. This is not uncommon for predictive modelers to do.

If the patterns require examining two variables at a time, you can do so with a scatter plot. For 50 variables, there are 1,225 possible scatter plots to examine. A dedicated predictive modeler might actually do this, although it will take some time. However, if the patterns require that you examine three variables simultaneously, you would need to examine 19,600 3D scatter plots in order to examine all the possible three-way combinations. Even the most dedicated modelers will be hard-pressed to spend the time needed to examine so many plots.

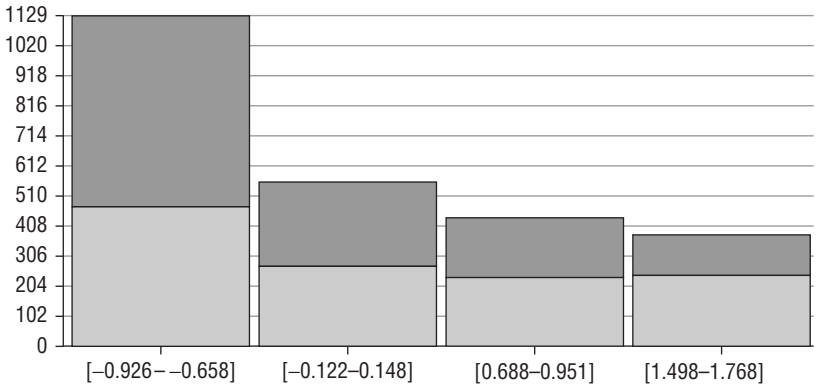


Figure 1-1: Histogram

You need algorithms to sift through all of the potential combinations of inputs in the data—the patterns—and identify which ones are the most interesting. The analyst can then focus on these patterns, undoubtedly a much smaller number of inputs to examine. Of the 19,600 three-way combinations of inputs, it may

be that a predictive model identifies six of the variables as the most significant contributors to accurate models. In addition, of these six variables, the top three are particularly good predictors and much better than any two variables by themselves. Now you have a manageable subset of plots to consider: 63 instead of nearly 20,000. This is one of the most powerful aspects of predictive analytics: identifying which inputs are the most important contributors to patterns in the data.

Supervised vs. Unsupervised Learning

Algorithms for predictive modeling are often divided into two groups: supervised learning methods and unsupervised learning methods. In supervised learning models, the *supervisor* is the target variable, a column in the data representing values to predict from other columns in the data. The target variable is chosen to represent the answer to a question the organization would like to answer or a value unknown at the time the model is used that would help in decisions. Sometimes supervised learning is also called *predictive modeling*. The primary predictive modeling algorithms are *classification* for categorical target variables or *regression* for continuous target variables.

Examples of target variables include whether a customer purchased a product, the amount of a purchase, if a transaction was fraudulent, if a customer stated they enjoyed a movie, how many days will transpire before the next gift a donor will make, if a loan defaulted, and if a product failed. Records without a value for the target variable cannot be used in building predictive models.

Unsupervised learning, sometimes called *descriptive modeling*, has no target variable. The inputs are analyzed and grouped or clustered based on the proximity of input values to one another. Each group or cluster is given a label to indicate which group a record belongs to. In some applications, such as in customer analytics, unsupervised learning is just called segmentation because of the function of the models (segmenting customers into groups).

The key to supervised learning is that the inputs to the model are known but there are circumstances where the target variable is unobserved or unknown. The most common reason for this is a target variable that is an event, decision, or other behavior that takes place at a time future to the observed inputs to the model. Response models, cross-sell, and up-sell models work this way: Given what is known now about a customer, can you predict if they will purchase a particular product in the future?

Some definitions of *predictive analytics* emphasize the function of algorithms as forecasting or predicting *future* events or behavior. While this is often the case, it certainly isn't always the case. The target variable could represent an unobserved variable like a missing value. If a taxpayer didn't file a return in a prior year, predictive models can predict that missing value from other examples of tax returns where the values are known.

Parametric vs. Non-Parametric Models

Algorithms for predictive analytics include both parametric and non-parametric algorithms. Parametric algorithms (or models) assume known distributions in the data. Many parametric algorithms and statistical tests, although not all, assume normal distributions and find linear relationships in the data. Machine learning algorithms typically do not assume distributions and therefore are considered non-parametric or distribution-free models.

The advantage of parametric models is that if the distributions are known, extensive properties of the data are also known and therefore algorithms can be proven to have very specific properties related to errors, convergence, and certainty of learned coefficients. Because of the assumptions, however, the analyst often spends considerable time transforming the data so that these advantages can be realized.

Non-parametric models are far more flexible because they do not have underlying assumptions about the distribution of the data, saving the analyst considerable time in preparing data. However, far less is known about the data *a priori*, and therefore non-parametric algorithms are typically iterative, without any guarantee that the best or optimal solution has been found.

Business Intelligence

Business intelligence is a vast field of study that is the subject of entire books; this treatment is brief and intended to summarize the primary characteristics of business intelligence as they relate to predictive analytics. The output of many business intelligence analyses are reports or dashboards that summarize interesting characteristics of the data, often described as Key Performance Indicators (KPIs). The KPI reports are user-driven, determined by an analyst or decision-maker to represent a key descriptor to be used by the business. These reports can contain simple summaries or very complex, multidimensional measures. Interestingly, KPI is almost never used to describe measures of interest in predictive analytics software and conferences.

Typical business intelligence output is a report to be used by analysts and decision-makers. The following are typical questions that might be answered by business intelligence for fraud detection and customer analytics:

Fraud Detection

- How many cases were investigated last month?
- What was the success rate in collecting debts?
- How much revenue was recovered through collections?
- What was the ROI for the various collection avenues: letters, calls, agents?

- What was the close rate of cases in the past month? Past quarter? Past year?
- For debts that were closed out, how many days did it take on average to close out debts?
- For debts that were closed out, how many contacts with the debtor did it take to close out debt?

Customer Analytics

- What were the e-mail open, click-through, and response rates?
- Which regions/states/ZIPs had the highest response rates?
- Which products had the highest/lowest click-through rates?
- How many repeat purchasers were there last month?
- How many new subscriptions to the loyalty program were there?
- What is the average spend of those who belong to the loyalty program? Those who aren't a part of the loyalty program? Is this a significant difference?
- How many visits to the store/website did a person have?

These questions describe characteristics of the unit of analysis: a customer, a transaction, a product, a day, or even a ZIP code. Descriptions of the unit of analysis are contained in the columns of the data: the attributes. For fraud detection, the unit of analysis is sometimes a debt to be collected, or more generally a case. For customer analytics, the unit of analysis is frequently a customer but could be a visit (a single customer could visit many times and therefore will appear in the data many times).

Note that often these questions compare directly one attribute of interest with an outcome of interest. These questions were developed by a domain expert (whether an analyst, program manager, or other subject matter expert) as a way to describe interesting relationships in the data relevant to the company. In other words, these measures are user-driven.

Are these KPIs and reports actionable decisions in and of themselves? The answer is no, although they can be with small modifications. In the form of the report, you know what happened and can even identify why it happened in some cases. It isn't a great leap, however, to take reports and turn them into predictions. For example, a report that summarizes the response rates for each ZIP code can then use ZIP as a predictor of response rate.

If you consider the reports related to a target variable such as response rate, the equivalent machine learning approach is building a *decision stump*, a single condition rule that predicts the outcome. But this is a very simple way of approaching prediction.

Predictive Analytics vs. Business Intelligence

What if you reconstruct the two lists of questions in a different way, one that is focused more directly on decisions? From a predictive analytics perspective, you may find these questions are the ones asked.

Fraud Detection

- What is the likelihood that the transaction is fraudulent?
- What is the likelihood the invoice is fraudulent or warrants further investigation?
- Which characteristics of the transaction are most related to or most predictive of fraud (single characteristics and interactions)?
- What is the expected amount of fraud?
- What is the likelihood that a tax return is non-compliant?
- Which line items on a tax return contribute the most to the fraud score?
- Historically, which demographic and historic purchase patterns were most related to fraud?

Customer Analytics for Predictive Analytics

- What is the likelihood an e-mail will be opened?
- What is the likelihood a customer will click-through a link in an e-mail?
- Which product is a customer most likely to purchase if given the choice?
- How many e-mails should the customer receive to maximize the likelihood of a purchase?
- What is the best product to up-sell to the customer after they purchase a product?
- What is the visit volume expected on the website next week?
- What is the likelihood a product will sell out if it is put on sale?
- What is the estimated customer lifetime value (CLV) of each customer?

Notice the differences in the kinds of questions predictive analytics asks compared to business intelligence. The word "likelihood" appears often, meaning we are computing a probability that the pattern exists for a unit of analysis. In customer analytics, this could mean computing a probability that a customer is likely to purchase a product.

Implicit in the wording is that the measures require an examination of the groups of records comprising the unit of analysis. If the likelihood an individual customer will purchase a product is one percent, this means that for every 100 customers with the same pattern of measured attributes for this customer,

one customer purchased the product in the historic data used to compute the likelihood. The comparable measure in the business intelligence lists would be described as a *rate* or a *percentage*; what is the response rate of customers with a particular purchase pattern.

The difference between the business intelligence and predictive analytics measures is that the business intelligence variables identified in the questions were, as already described, user driven. In the predictive analytics approach, the predictive modeling algorithms considered many patterns, sometimes all possible patterns, and determined which ones were most predictive of the measure of interest (likelihood). The discovery of the patterns is data driven.

This is also why many of the questions begin with the word "which." Asking *which* line items on a tax return are most related to noncompliance requires comparisons of the line items as they relate to noncompliance.

Do Predictive Models Just State the Obvious?

Often when presenting models to decision-makers, modelers may hear a familiar refrain: "I didn't need a model to tell me that!" But predictive models do more than just identify attributes that are related to a target variable. They identify the *best way* to predict the target. Of all the possible alternatives, all of the attributes that could predict the target and all of the interactions between the attributes, which combinations do the best job? The decision-maker may have been able to guess (hypothesize) that length or residence is a good attribute to predict a responder to a Medicare customer acquisition campaign, but that same person may not have known that the number of contacts is even more predictive, especially when the prospect has been mailed two to six times. Predictive models identify not only which variables are predictive, but how well they predict the target. Moreover, they also reveal which combinations are not just predictive of the target, but *how well* the combinations predict the target and how much better they predict than individual attributes do on their own.

Similarities between Business Intelligence and Predictive Analytics

Often, descriptions of the differences between business intelligence and predictive analytics stress that business intelligence is retrospective analysis, looking back into the past, whereas predictive analytics or prospective analysis predict future behavior. The "predicting the future" label is applied often to predictive analytics in general and the very questions described already imply this is the case. Questions such as "What is the likelihood a customer will purchase . . ." are forecasting future behavior.

Figure 1-2 shows a timeline relating data used to build predictive models or business intelligence reports. The vertical line in the middle is the time the

model is being built (today). The data used to build the models is always to the left: historic data. When predictive models are built to predict a "future" event, the data selected to build the predictive models is rolled back to a time prior to the date the future event is known.

For example, if you are building models to predict whether a customer will respond to an e-mail campaign, you begin with the date the campaign cured (when all the responses have come in) to identify everyone who responded. This is the date for the label "target variable computed based on this date" in the figure. The attributes used as inputs must be known prior to the date of the mailing itself, so these values are collected to the left of the target variable collection date. In other words, the data is set up with all the modeling data in the past, but the target variable is still future to the date the attributes are collected in the timeline of the data used for modeling.

However, it's important to be clear that both business intelligence and predictive analytics analyses are built from the same data, and the data is historic in both cases. The assumption is that future behavior to the right of the vertical line in Figure 1-2 will be consistent with past behavior. If a predictive model identifies patterns in the past that predicted (in the past) that a customer would purchase a product, you assume this relationship will continue to be present in the future.

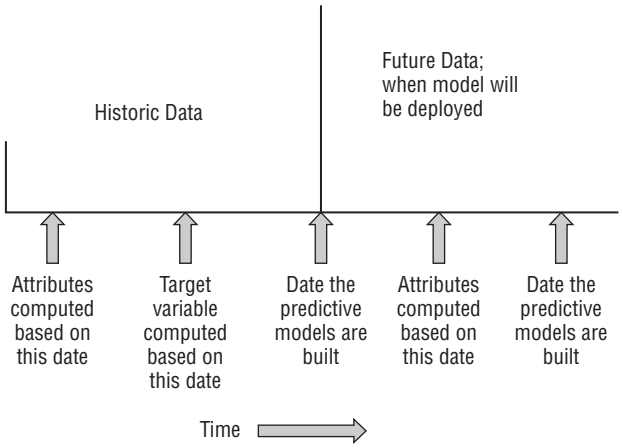


Figure 1-2: Timeline for building predictive models

Predictive Analytics vs. Statistics

Predictive analytics and statistics have considerable overlap, with some statisticians arguing that predictive analytics is, at its core, an extension of statistics. Predictive modelers, for their part, often use algorithms and tests common in statistics as a part of their regular suite of techniques, sometimes without

applying the diagnostics most statisticians would apply to ensure the models are built properly.

Since predictive analytics draws heavily from statistics, the field has taken to heart the amusing quote from statistician and creator of the bootstrap, Brad Efron: "Those who ignore Statistics are condemned to reinvent it." Nevertheless, there are significant differences between typical approaches of the two fields. Table 1-1 provides a short list of items that differ between the fields. Statistics is driven by theory in a way that predictive analytics is not, where many algorithms are drawn from other fields such as machine learning and artificial intelligence that have no provable optimum solution.

But perhaps the most fundamental difference between the fields is summarized in the last row of the table: For statistics, the model is king, whereas for predictive analytics, data is king.

Table 1-1: Statistics vs. Predictive Analytics

STATISTICS	PREDICTIVE ANALYTICS
Models based on theory: There is an optimum.	Models often based on non-parametric algorithms; no guaranteed optimum
Models typically linear.	Models typically nonlinear
Data typically smaller; algorithms often geared toward accuracy with small data	Scales to big data; algorithms not as efficient or stable for small data
The model is king.	Data is king.

Statistics and Analytics

In spite of the similarities between statistics and analytics, there is a difference in mindset that results in differences in how analyses are conducted. Statistics is often used to perform confirmatory analysis where a hypothesis about a relationship between inputs and an output is made, and the purpose of the analysis is to confirm or deny the relationship and quantify the degree of that confirmation or denial. Many analyses are highly structured, such as determining if a drug is effective in reducing the incidence of a particular disease.

Controls are essential to ensure that bias is not introduced into the model, thus misleading the analyst's interpretation of the model. Coefficients of models are critically important in understanding what the data are saying, and therefore great care is taken to transform the model inputs and outputs so they comply with assumptions of the modeling algorithms. If the study is predicting the effect of caloric intake, smoking, age, height, amount of exercise, and metabolism on an individual's weight, and one is to trust the relative contribution of each factor on an individual's weight, it is important to remove any bias due to the data itself so that the conclusions reflect the intent of the model. Bias in

the data could result in misleading the analyst that the inputs to the model have more or less influence that they actually have, simply because of numeric problems in the data.

Residuals are also carefully examined to identify departure from a Normal distribution, although the requirement of normality lessens as the size of the data increases. If residuals are not random with constant variance, the statistician will modify the inputs and outputs until these problems are corrected.

Predictive Analytics and Statistics Contrasted

Predictive modelers, on the other hand, often show little concern for final parameters in the models except in very general terms. The key is often the predictive accuracy of the model and therefore the ability of the model to make and influence decisions. In contrast to the structured problem being solved through confirmatory analysis using statistics, predictive analytics often attempts to solve less structured business problems using data that was not even collected for the purpose of building models; it just happened to be around. Controls are often not in place in the data and therefore causality, very difficult to uncover even in structured problems, becomes exceedingly difficult to identify. Consider, for example, how you would go about identifying which marketing campaign to apply to a current customer for a digital retailer. This customer could receive content from any one of ten programs the e-mail marketing group has identified. The modeling data includes customers, their demographics, their prior behavior on the website and with e-mail they had received in the past, and their reaction to sample content from one of the ten programs. The reaction could be that they ignored the e-mail, opened it, clicked through the link, and ultimately purchased the product promoted in the e-mail. Predictive models can certainly be built to identify the best program of the ten to put into the e-mail based on a customer's behavior and demographics.

However, this is far from a controlled study. While this program is going on, each customer continues to interact with the website, seeing other promotions. The customer may have seen other display ads or conducted Google searches further influencing his or her behavior. The purpose of this kind of model cannot be to uncover fully why the customer behaves in a particular way because there are far too many unobserved, confounding influences. But that doesn't mean the model isn't useful.

Predictive modelers frequently approach problems in this more unstructured, even casual manner. The data, in whatever form it is found, drives the models. This isn't a problem as long as the data continues to be collected in a manner consistent with the data as it was used in the models; consistency in the data will increase the likelihood that there will be consistency in the model's predictions, and therefore how well the model affects decisions.

Predictive Analytics vs. Data Mining

Predictive analytics has much in common with its immediate predecessor, data mining; the algorithms and approaches are generally the same. Data mining has a history of applications in a wide variety of fields, including finance, engineering, manufacturing, biotechnology, customer relationship management, and marketing. I have treated the two fields as generally synonymous since "predictive analytics" became a popular term.

This general overlap between the two fields is further emphasized by how software vendors brand their products, using both data mining and predictive analytics (some emphasizing one term more than the other).

On the other hand, data mining has been caught up in the specter of privacy concerns, spam, malware, and unscrupulous marketers. In the early 2000s, congressional legislation was introduced several times to curtail specifically any data mining programs in the Department of Defense (DoD). Complaints were even waged against the use of data mining by the NSA, including a letter sent by Senator Russ Feingold to the National Security Agency (NSA) Director in 2006:

> *One element of the NSA's domestic spying program that has gotten too little attention is the government's reportedly widespread use of data mining technology to analyze the communications of ordinary Americans. Today I am calling on the Director of National Intelligence, the Defense Secretary and the Director of the NSA to explain whether and how the government is using data mining technology, and what authority it claims for doing so.*

In an interesting *déjà vu*, in 2013, information about NSA programs that sift through phone records was leaked to the media. As in 2006, concerns about privacy were again raised, but this time the mathematics behind the program, while typically described as data mining in the past, was now often described as predictive analytics.

Graduate programs in analytics often use both data mining and predictive analytics in their descriptions, even if they brand themselves with one or the other.

Who Uses Predictive Analytics?

In the 1990s and early 2000s, the use of advanced analytics, referred to as data mining or computational statistics, was relegated to only the most forward-looking companies with deep pockets. Many organizations were still struggling with collecting data, let alone trying to make sense of it through more advanced techniques.

Today, the use of analytics has moved from a niche group in large organizations to being an instrumental component of most mid- to large-sized organizations.

The analytics often begins with business intelligence and moves into predictive analytics as the data matures and the pressure to produce greater benefit from the data increases. Even small organizations, for-profit and non-profit, benefit from predictive analytics now, often using open source software to drive decisions on a small scale.

Challenges in Using Predictive Analytics

Predictive analytics can generate significant improvements in efficiency, decision-making, and return on investment. But predictive analytics isn't always successful and, in all likelihood, the majority of predictive analytics models are never used operationally.

Some of the most common reasons predictive models don't succeed can be grouped into four categories: obstacles in management, obstacles with data, obstacles with modeling, and obstacles in deployment.

Obstacles in Management

To be useful, predictive models have to be deployed. Often, deployment in of itself requires a significant shift in resources for an organization and therefore the project often needs support from management to make the transition from research and development to operational solution. If program management is not a champion of the predictive modeling project and the resulting models, perfectly good models will go unused due to lack of resources and lack of political will to obtain those resources.

For example, suppose an organization is building a fraud detection model to identify transactions that appear to be suspicious and are in need of further investigation. Furthermore, suppose the organization can identify 1,000 transactions per month that should receive further scrutiny from investigators. Processes have to be put into place to distribute the cases to the investigators, and the fraud detection model has to be sufficiently trusted by the investigators for them to follow through and investigate the cases. If management is not fully supportive of the predictive models, these cases may be delivered but end up dead on arrival.

Obstacles with Data

Predictive models require data in the form of a single table or flat file containing rows and columns: two-dimensional data. If the data is stored in transactional databases, keys need to be identified to join the data from the data sources to form the single view or table. Projects can fail before they even begin if the keys don't exist in the tables needed to build the data.

Even if the data can be joined into a single table, if the primary inputs or outputs are not populated sufficiently or consistently, the data is meaningless. For example, consider a customer acquisition model. Predictive models need examples of customers who were contacted and did *not* respond as well as those who were contacted and *did* respond. If active customers are stored in one table and marketing contacts (leads) in a separate table, several problems can thwart modeling efforts. First, unless customer tables include the campaign they were acquired from, it may be impossible to reconstruct the list of leads in a campaign along with the label that the lead responded or didn't respond to the contact.

Second, if customer data, including demographics (age, income, ZIP), is overwritten to keep it up-to-date, and the demographics at the time they were acquired is not retained, a table containing leads as they appeared at the time of the marketing campaign can never be reconstructed. As a simple example, suppose phone numbers are only obtained after the lead converts and becomes a customer. A great predictor of a lead becoming a customer would then be whether the lead has a phone number; this is leakage of future data unknown at the time of the marketing campaign into the modeling data.

Obstacles with Modeling

Perhaps the biggest obstacle to building predictive models from the analyst's perspective is *overfitting*, meaning that the model is too complex, essentially memorizing the training data. The effect of overfitting is twofold: The model performs poorly on new data and the interpretation of the model is unreliable. If care isn't taken in the experimental design of the predictive models, the extent of model overfit isn't known until the model has already been deployed and begins to fail.

A second obstacle with building predictive models occurs when zealous analysts become too ambitious in the kind of model that can be built with the available data and in the timeframe allotted. If they try to "hit a home run" and can't complete the model in the timeframe, no model will be deployed at all. Often a better strategy is to build simpler models first to ensure a model of some value will be ready for deployment. Models can be augmented and improved later if time allows.

For example, consider a customer retention model for a company with an online presence. A zealous modeler may be able to identify thousands of candidate inputs to the retention model, and in an effort to build the best possible model, may be slowed by the sheer combinatorics involved with data preparation and variable selection prior to and during modeling.

However, from the analyst's experience, he or she may be able to identify 100 variables that have been good predictors historically. While the analyst suspects that a better model could be built with more candidate inputs, the first model can be built from the 100 variables in a much shorter timeframe.

Obstacles in Deployment

Predictive modeling projects can fail because of obstacles in the deployment stage of modeling. The models themselves are typically not very complicated computationally, requiring only dozens, hundreds, thousands, or tens of thousands of multiplies and adds, easily handled by today's servers.

At the most fundamental level, however, the models have to be able to be interrogated by the operational system and to issue predictions consistent with that system. In transactional systems, this typically means the model has to be encoded in a programming language that can be called by the system, such as SQL, C++, Java, or another high-level language. If the model cannot be translated or is translated incorrectly, the model is useless operationally.

Sometimes the obstacle is getting the data into the format needed for deployment. If the modeling data required joining several tables to form the single modeling table, deployment must replicate the same joining steps to build the data the models need for scoring. In some transactional systems with disparate data forming the modeling table, complex joins may not be possible in the timeline needed. For example, consider a model that recommends content to be displayed on a web page. If that model needs data from the historic patterns of browsing behavior for a visitor and the page needs to be rendered in less than one second, all of the data pulls and transformations must meet this timeline.

What Educational Background Is Needed to Become a Predictive Modeler?

Conventional wisdom says that predictive modelers need to have an academic background in statistics, mathematics, computer science, or engineering. A degree in one of these fields is best, but without a degree, at a minimum, one should at least have taken statistics or mathematics courses. Historically, one could not get a degree in predictive analytics, data mining, or machine learning.

This has changed, however, and dozens of universities now offer master's degrees in predictive analytics. Additionally, there are many variants of analytics degrees, including master's degrees in data mining, marketing analytics, business analytics, or machine learning. Some programs even include a practicum so that students can learn to apply textbook science to real-world problems.

One reason the real-world experience is so critical for predictive modeling is that the science has tremendous limitations. Most real-world problems have data problems never encountered in the textbooks. The ways in which data can go wrong are seemingly endless; building the same customer acquisition models even within the same domain requires different approaches to data preparation, missing value imputation, feature creation, and even modeling methods.

However, the *principles* of how one can solve data problems are not endless; the experience of building models for several years will prepare modelers to at least be able to identify when potential problems may arise.

Surveys of top-notch predictive modelers reveal a mixed story, however. While many have a science, statistics, or mathematics background, many do not. Many have backgrounds in social science or humanities. How can this be?

Consider a retail example. The retailer Target was building predictive models to identify likely purchase behavior and to incentivize future behavior with relevant offers. Andrew Pole**,** a Senior Manager of Media and Database Marketing described how the company went about building systems of predictive models at the Predictive Analytics World Conference in 2010. Pole described the importance of a combination of domain knowledge, knowledge of predictive modeling, and most of all, a forensic mindset in successful modeling of what he calls a "guest portrait."

They developed a model to predict if a female customer was pregnant. They noticed patterns of purchase behavior, what he called "nesting" behavior. For example, women were purchasing cribs on average 90 days before the due date. Pole also observed that some products were purchased at regular intervals prior to a woman's due date. The company also observed that if they were able to acquire these women as purchasers of other products during the time before the birth of their baby, Target was able to increase significantly the customer value; these women would continue to purchase from Target after the baby was born based on their purchase behavior before.

The key descriptive terms are "*observed*" and "*noticed*." This means the models were not built as black boxes. The analysts asked, "does this make sense?" and leveraged insights gained from the patterns found in the data to produce better predictive models. It undoubtedly was iterative; as they "noticed" patterns, they were prompted to consider other patterns they had not explicitly considered before (and maybe had not even occurred to them before). This forensic mindset of analysts, noticing interesting patterns and making connections between those patterns and how the models could be used, is critical to successful modeling. It is rare that predictive models can be fully defined before a project and anticipate all of the most important patterns the model will find. So we shouldn't be surprised that we *will* be surprised, or put another way, we should *expect* to be surprised.

This kind of mindset is not learned in a university program; it is part of the personality of the individual. Good predictive modelers need to have a forensic mindset and intellectual curiosity, whether or not they understand the mathematics enough to derive the equations for linear regression.

CHAPTER

2

Setting Up the Problem

The most important part of any predictive modeling project is the very beginning when the predictive modeling project is defined. Setting up a predictive modeling project is a very difficult task because the skills needed to do it well are very broad, requiring knowledge of the business domain, databases, or data infrastructure, and predictive modeling algorithms and techniques. Very few individuals have all of these skill sets, and therefore setting up a predictive modeling project is inevitably a team effort.

This chapter describes principles to use in setting up a predictive modeling project. The role practitioners play in this stage is critical because missteps in defining the unit of analysis, target variables, and metrics to select models can render modeling projects ineffective.

Predictive Analytics Processing Steps: CRISP-DM

The Cross-Industry Standard Process Model for Data Mining (CRISP-DM) describes the data-mining process in six steps. It has been cited as the most-often used process model since its inception in the 1990s. The most frequently cited alternative to CRISP-DM is an organization's or practitioner's own process model, although upon more careful examination, these are also essentially the same as CRISP-DM.

One advantage of using CRISP-DM is that it describes the most commonly applied steps in the process and is documented in an 80-page PDF file. The CRISP-DM name itself calls out data mining as the technology, but the same process model applies to predictive analytics and other related analytics approaches, including business analytics, statistics, and text mining.

The CRISP-DM audience includes both managers and practitioners. For program managers, CRISP-DM describes the steps in the modeling process from a program perspective, revealing the steps analysts will be accomplishing as they build predictive models. Each of the steps can then have its own cost estimates and can be tracked by the manager to ensure the project deliverables and timetables are met. The last step in many of the sub-tasks in CRISP-DM is a report describing what decisions were made and why. In fact, the CRISP-DM document identifies 28 potential deliverables for a project. This certainly is music to the program manager's ears!

For practitioners, the step-by-step process provides structure for analysis and not only reminds the analyst of the steps that need to be accomplished, but also the need for documentation and reporting throughout the process, which is particularly valuable for new modelers. Even for experienced practitioners, CRISP-DM describes the steps succinctly and logically. Many practitioners are hesitant to describe the modeling process in linear, step-by-step terms because projects almost never proceed as planned due to problems with data and modeling; surprises occur in nearly every project. However, a good baseline is still valuable, especially as practitioners describe to managers what they are doing and why they are doing it; CRISP-DM provides the justification for the steps that need to be completed in the process of building models.

The six steps in the CRISP-DM process are shown in Figure 2-1: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. These steps, and the sequence they appear in the figure, represent the most common sequence in a project. These are described briefly in Table 2-1.

Table 2-1: CRISM-DM Sequence

STAGE	DESCRIPTION
Business Understanding	Define the project.
Data Understanding	Examine the data; identify problems in the data.
Data Preparation	Fix problems in the data; create derived variables.
Modeling	Build predictive or descriptive models.
Evaluation	Assess models; report on the expected effects of models.
Deployment	Plan for use of models.

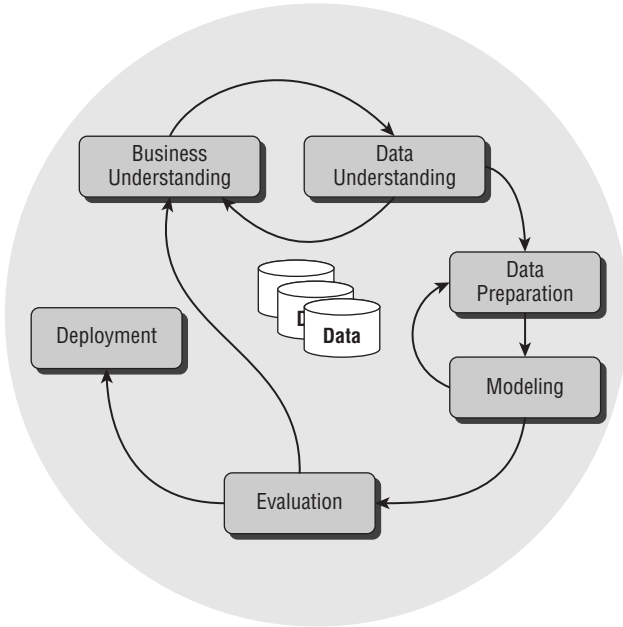


Figure 2-1: The CRISP-DM process model

Note the feedback loops in the figure. These indicate the most common ways the typical process is modified based on findings during the project. For example, if business objectives have been defined during Business Understanding, and then data is examined during Data Understanding, you may find that there is insufficient data quantity or data quality to build predictive models. In this case, Business Objectives must be re-defined with the available data in mind before proceeding to Data Preparation and Modeling. Or consider a model that has been built but has poor accuracy. Revisiting data preparation to create new derived variables is a common step to improve the models.

Business Understanding

Every predictive modeling project needs objectives. Domain experts who understand decisions, alarms, estimates, or reports that provide value to an organization must define these objectives. Analysts themselves sometimes have this expertise, although most often, managers and directors have a far better perspective on how models affect the organization. Without domain expertise, the definitions of what models should be built and how they should be assessed can lead to failed projects that don't address the key business concerns.

The Three-Legged Stool

One way to understand the collaborations that lead to predictive modeling success is to think of a three-legged stool. Each leg is critical to the stool remaining stable and fulfilling its intended purpose. In predictive modeling, the three legs of the stool are (1) domain experts, (2) data or database experts, and (3) predictive modeling experts. Domain experts are needed to frame a problem properly in a way that will provide value to the organization. Data or database experts are needed to identify what data is available for predictive modeling and how that data can be accessed and normalized. Predictive modelers are needed to build the models that achieve the business objectives.

Consider what happens if one or more of these three legs are missing. If the problem is not defined properly and only modelers and the database administrator are defining the problems, excellent models may be built with fantastic accuracy only to go unused because the model doesn't address an actual need of the organization. Or in a more subtle way, perhaps the model predicts the right kind of decision, but the models are assessed in a way that doesn't address very well what matters most to the business; the wrong model is selected because the wrong metric for describing good models is used.

If the database expert is not involved, data problems may ensue. First, there may not be enough understanding of the layout of tables in the database to be able to access all of the fields necessary for predictive modeling. Second, there may be insufficient understanding of fields and what information they represent even if the names of the fields seem intuitive, or worse still, if the names are cryptic and no data dictionary is available. Third, insufficient permissions may preclude pulling data into the predictive modeling environment. Fourth, database resources may not support the kinds of joins the analyst may believe he or she needs to build the modeling data. And fifth, model deployment options envisioned by the predictive modeling team may not be supported by the organization.

If the predictive modelers are not available during the business understanding stage of CRISP-DM, obstacles outlined in this chapter may result. First, a lack of understanding by program managers of what the predictive models can do, driven by hype around predictive modeling, can lead the manager to specify models that are impossible to actually build. Second, defining target variables for predictive modeling may not be undertaken at all or, if done, may be specified poorly, thwarting predictive modeling efforts. Third, without predictive modelers defining the layout of data needed for building predictive models, a

modeling table to be used by the modeler may not be defined at all or may lack key fields needed for the models.

Business Objectives

Assuming all three types of individuals that make up the three-legged stool of predictive modeling are present during the Business Understand stage of CRISP-DM, tradeoffs and compromises are not unusual during the hours or even days of meetings that these individuals and groups participate in so that solid business and predictive modeling objectives are defined.

Six key issues that should be resolved during the Business Understanding stage include definitions of the following:

- Core business objectives to be addressed by the predictive models
- How the business objectives can be quantified
- What data is available to quantify the business objectives
- What modeling methods can be invoked to describe or predict the business objectives
- How the goodness of model fit of the business objectives are quantified so that the model scores make business sense
- How the predictive models can be deployed operationally

Frequently, the compromises reached during discussions are the result of the imperfect environment that is typical in most organizations. For example, data that you would want to use in the predictive models may not be available in a timely manner or at all. Target variables that address the business objectives more directly may not exist or be able to be quantified. Computing resources may not exist to build predictive models in the way the analysts would prefer. Or there may not be available staff to apply to the project in the timeframe needed. And these are just a few possible issues that may be uncovered. Project managers need to be realistic about which business objectives can be achieved in the timeframe and within the budget available.

Predictive modeling covers a wide range of business objectives. Even the term "business objectives" is restrictive as modeling can be done for more than just what you normally associate with a commercial enterprise. Following is a short list of predictive modeling projects. I personally have either built models for, or advised a customer on, building models for each of these projects.

PROJECT		
Customer acquisition/ Response/Lead generation	Credit card application fraud	Medical image anomaly detection
Cross-sell/Up-sell	Loan application fraud	Radar signal, vehicle/aircraft identification
Customer next product to purchase	Invoice fraud	Radar, friend-or-foe differentiation
Customer likelihood to purchase in *N* days	Insurance claim fraud	Sonar signal object identification (long and short range)
Website—next site to interact with	Insurance application fraud	Optimum guidance commands for smart bombs or tank shells
Market-basket analysis	Medical billing fraud	Likelihood for flight to be on time
Customer value/Customer profitability	Payment fraud	Insurance risk of catastrophic claim
Customer segmentation	Warranty fraud	Weed tolerance to pesticides
Customer engagement with brand	Tax collection likelihood to pay	Mean time to failure/ Likelihood to fail
Customer attrition/Retention	Non-filer predicted tax liability	Likelihood of hardware failure due to complexity
Customer days to next purchase	Patient likelihood to re-admit	Fault detection/Fault explanation
Customer satisfaction	Patient likelihood to comply with medication protocols	Part needed for repair
Customer sentiment/ Recommend to a friend	Cancer detection	Intrusion detection/ Likelihood of an intrusion event
Best marketing creative	Gene expression/ Identification	New hire likelihood to succeed/advance
Credit card transaction fraud	Predicted toxicity (LD50 or LC50) of substance	New hire most desirable characteristics

While many models are built to predict the behavior of people or things, not all are. Some models are built expressly for the purpose of understanding the behavior of people, things, or processes better. For example, predicting "weed tolerance to pesticides" was built to test the hypothesis that the weeds were becoming intolerant to a specific pesticide. The model identified the primary

contributors in predicting success or failure in killing the weeds; this in and of itself was insightful. While the likelihood of a customer purchasing a product within seven days is interesting on its own, understanding *why* the customer is likely to purchase can provide even more value as the business decides how best to contact the individuals. Or if a Customer Retention model is built with high accuracy, those customers that match the profile for retention but attrite nevertheless become a good segment for call-center follow-up.

Defining Data for Predictive Modeling

Data for predictive modeling must be two-dimensional, comprised of rows and columns. Each row represents what can be called a *unit of analysis*. For customer analytics, this is typically a customer. For fraud detection, this may be a transaction. For call center analytics, this may refer to an individual call. For survey analysis, this may be a single survey. The unit of analysis is problem-specific and therefore is defined as part of the Business Understanding stage of predictive modeling.

If data for modeling is loaded from files, the actual form of the data is largely irrelevant because most software packages support data in a variety of formats:

- Delimited flat files, usually delimited with commas (`.csv` files), tabs, or some other custom character to indicate where field values begin and end
- Fixed-width flat files with a fixed number of characters per field. No delimiters are needed in this format but the exact format for each field must be known before loading the data.
- Other customized flat files
- Binary files, including formats specific to software packages such as SPSS files (`.sav`), SAS files (`.sas7bdat`), and Matlab files (`.mat`)

Most software packages also provide connectivity to databases through native or ODBC drivers so that tables and views can be accessed directly from the software. Some software allows for the writing of simple or even complex queries to access the data from within the software itself, which is very convenient for several reasons:

- Data does not have to be saved to disk and loaded into the predictive modeling software, a slow process for large data sets.
- Data can be maintained in the database without having to provide version control for the flat files.
- Analysts have greater control and flexibility over the data they pull from the database or data mart.

However, you must also be careful that the tables and views being accessed remain the same throughout the modeling project and aren't changing without any warning. When data changes without the knowledge of the analyst, models can also change inexplicably.

Defining the Columns as Measures

Columns in the data are often called *attributes*, *descriptors*, *variables*, *fields*, *features*, or just columns. The book will use these labels interchangeably. Variables in the data are measures that relate to or describe the record. For customer analytics, one attribute may be the customer ID, a second the customer's age, a third the customer's street address, and so forth. The number of attributes in the data is limited only by what is measured for the particular unit of analysis, which attributes are considered to be useful, and how many attributes can be handled by the database or predictive modeling software.

Columns in the data are measures of that unit of analysis, and for predictive modeling algorithms, the number of columns and the order of the columns must be identical from record to record. Another way to describe this kind of data is that the data is rectangular. Moreover, the meaning of the columns must be consistent. If you are building models based on customer behavior, you are faced with an immediate dilemma: How can you handle customers who have visited different numbers of times and maintain the rectangular shape of the data?

Consider Table 2-2 with two customers of a hotel chain, one of whom has visited three times and the other only once. In the table layout, the column labeled "Date of Visit 1" is the date of the first visit the customer made to the hotel property. Customer 100001 has visited only once and therefore has no values for visit 2 and visit 3. These can be labeled as "NULL" or just left blank. The fact that they are not defined, however, can cause problems for some modeling algorithms, and therefore you often will not represent multiple visits as separate columns. This issue is addressed in Chapter 4.

Table 2-2: Simple Rectangular Layout of Data

CUSTOMER ID	DATE OF VISIT 1	DATE OF VISIT 2	DATE OF VISIT 3
100001	5/2/12	NULL	NULL
100002	6/9/12	9/29/12	10/13/12

There is a second potential problem with this layout of the data, however. The "Date of Visit 1" is the first visit. What if the pattern of behavior related to the models is better represented by how the customer behaved most recently? For customer 100001, the most recent visit is contained in the column "Date of

Visit 1," whereas for customer 100002, the most recent visit is in the column "Date of Visit 3." Predictive modeling algorithms consider each column as a separate measure, and therefore, if there is a strong pattern related to the most recent visit, the pattern is broken in this representation of the data. Alternatively, you could represent the same data as shown in Table 2-3.

Table 2-3: Alternative Rectangular Layout of Data

CUSTOMER ID	DATE OF VISIT 1	DATE OF VISIT 2	DATE OF VISIT 3
100001	5/2/12	NULL	NULL
100002	10/13/12	9/29/12	6/9/12

In this data, Visit 1 is no longer the first visit but is the most recent visit, Visit 2 is two visits ago, Visit 3 is three visits ago, and so on. The representation of the data you choose is dependent on which representation is expected to provide the most predictive set of inputs to models.

A third option for this customer data is to remove the temporal data completely and represent the visits in a consistent set of attributes that summarizes the visits. Table 2-4 shows one such representation: The same two customers are described by their most recent visit, the first visit, and the number of visits.

Table 2-4: Summarized Representation of Visits

CUSTOMER ID	DATE OF FIRST VISIT	DATE OF MOST RECENT VISIT	NUMBER OF VISITS
100001	5/2/12	5/2/12	1
100002	6/9/12	10/13/12	3

Ultimately, the representation problems described in Tables 2-3, 2-4, and 2-5 occur because this data is inherently three dimensional, not two. There is a temporal dimension that has to be represented in the row-column format, usually by summarizing the temporal dimension into *features* of the data.

Defining the Unit of Analysis

Predictive modeling algorithms assume that each record is an independent observation. Independence in this context merely means that the algorithms do not consider direct connections between records. For example, if records 1 and 2 are customers who are husband and wife and frequently travel together, the algorithms treat these two no differently than any two people with similar patterns of behavior; the algorithms don't know they are related or connected.

If the rows are not independent and in some way are connected to each other, the data itself will not be representative of the broad set of patterns in the complete set of records the model may encounter after it is deployed.

Consider hospitality analytics where each record is a customer who visits the property of a hotel. The assumption of independence is satisfied because (presumably) each customer behaves as a separate, independent entity. This is plausible, although there are exceptions to this assumption in the case of business conventions and conferences.

But what if the modeling data is not truly independent and is built from a single organization that has a contract with the hotel. Assume also that the organization has travel procedures in place so that reservations are always placed in the same way, and visits can only be made for business purposes. These records have a connection with each other, namely in the procedures of the organization. A model built from these records is, of course, biased because the data comes from a single organization, and therefore may not apply well to the general population. In addition, however, models built from this data will identify patterns that are related to the organization's procedures rather than the visitors' behavior had they decided on their own to visit the hotel.

Which Unit of Analysis?

It isn't always clear which unit of analysis to use. Suppose you are building models for the same hospitality organization and that the business objectives include identifying customer behavior so they can customize marketing creative content to better match the type of visitor they are contacting. Assume also that the objective is to increase the number of visits a customer makes to the property in the next quarter.

The first obvious choice is to make the unit of analysis a person: a customer. In this scenario, each record will represent an individual customer, and columns will describe the behavior of that customer, including their demographics and their behavior at the hotel. If the customer has visited the hotel on multiple occasions, that history must be represented in the single record. Some possible variables relate to the most recent visit, how frequently the customer visits, and how much the customer spends on the reservation, restaurant, and other amenities at the hotel. These derived variables will take some effort to define and compute. However, each record will contain a complete summary of that customer's behavior, and no other record will relate to that particular customer—each record is an independent observation.

What is obscured by this representation is detail-related to each individual visit. If a customer has undergone a sequence of visits that is trending toward higher or lower spend, rolling up the visits into a single record for the customer

will obscure the details unless the details are explicitly revealed as columns in the data. To capture specific detail, you must create additional derived variables.

A second unit of analysis is the visit: Each record contains only information about a single visit and the behavior of the customer during that visit. If a customer visited ten times, that customer will have ten records in the data and these records would be considered independent events without any immediate connection to one another; the modeling algorithms would not know that this particular customer had visited multiple times, nor would the algorithms know there is a connection between the visits. The effect of an individual customer visiting multiple times is this: The pattern of that customer is weighted by the number of times the customer visits, making it appear to the algorithms that the pattern exists more often in the data. From a visit perspective, of course, this is true, whereas from a customer perspective it is not true.

One way to communicate the connection between visits to individual customers is to include derived variables in the data that explicitly connect the history of visits. Derived variables such as the number of visits, the average spend in the past three visits, and the number of days since the last visit can all be added. You must take care to avoid leaking future information from the derived variable into the record for the visit, however. If you create a derived variable summarizing the number of visits the customer has made, no future visit can be included, only visits whose end dates precede the date of the visit for the record. This precludes creating these derived variables from simple "group by" operations with an appropriate "where" clause.

Ultimately, the unit of analysis selected for modeling is determined by the business objectives and how the model will be used operationally. Are decisions made from the model scores based on a transaction? Are they made based on the behavior of a single customer? Are they made based on a single visit, or based on aggregate behavior of several transactions or visits over a time period? Some organizations even build multiple models from the same data with different units of analysis precisely because the unit of analysis drives the decisions you can make from the model.

Defining the Target Variable

For models that estimate or predict a specific value, a necessary step in the Business Understanding stage is to identify one or more *target variables* to predict. A target variable is a column in the modeling data that contains the values to be estimated or predicted as defined in the business objectives. The target variable can be numeric or categorical depending on the type of model that will be built.

Table 2-5 shows possible target variables associated with a few projects from the list of projects in the section "Business Objectives" that appears earlier in the chapter.

Table 2-5: Potential Target Variables

PROJECT	TARGET VARIABLE TYPE	TARGET VARIABLE VALUES
Customer Acquisition	Binary	1 for acquired, 0 for non-acquired
Customer Value	Continuous	Dollar value of customer
Invoice Fraud Detection	Categorical	Type of fraud (5 levels)
Days to Next Purchase	Continuous	Number of days
Days to Next Purchase <= 7	Binary	1 for purchased within 7 days, 0 for did not purchase within 7 days

The first two items in the table are typical predictive modeling problems: the first a classification problem and the second an estimation problem. These will be defined in Chapter 8. The third item, Invoice Fraud Detection, could have been defined with a binary target variable (1 for "fraud," 0 for "not fraud") but was instead defined with five levels: four types of fraud and one level for "not fraud." This provides not only an indication as to whether or not the invoice is fraudulent, but also a more specific prediction of the type of fraud that could be used by the organization in determining the best course of action.

Note the last two items in the table. Both target variables address the same idea, predicting the number of days until the next purchase. However, they are addressed in different ways. The first is the more straightforward prediction of the actual number of days until the next purchase although some organizations may want to constrain the prediction to a 7-day window for a variety of reasons. First, they may not care if a customer will purchase 30 or 60 days from now because it is outside of the window of influence they may have in their programs. Second, binary classification is generally an easier problem to solve accurately. These models do not have to differentiate between someone who purchases 14 days from now from someone who purchases 20 days from now: In the binary target variable formulation, these are the same (both have value 0). The predictive models may predict the binary target variable more accurately than the entire distribution.

Defining the target variable is critically important in a predictive modeling project because it is the only information the modeling algorithms have to uncover what the modeler and program manager desire from the predictions. Algorithms do not have common sense and do not bring context to the problem in the way the modeler and program manager can. The target variable definition therefore must describe or quantify as much as possible the business objective itself.

Temporal Considerations for Target Variable

For most modeling projects focused on predicting future behavior, careful consideration for the timeline is essential. Predictive modeling data, as is the case with all data, is historical, meaning that it was collected in the past. To build a model that predicts so-called future actions from historic data requires shifting the timeline in the data itself.

Figure 2-2 shows a conceptualized timeline for defining the target variable. If the date the data is pulled is the last vertical line on the right, the "Data pull timestamp," all data used for modeling by definition must precede that date. Because the timestamp for the definition of the target variable value must occur *after* the last information known from the input variables, the timeline for constructing the modeling data must be shifted to the left.

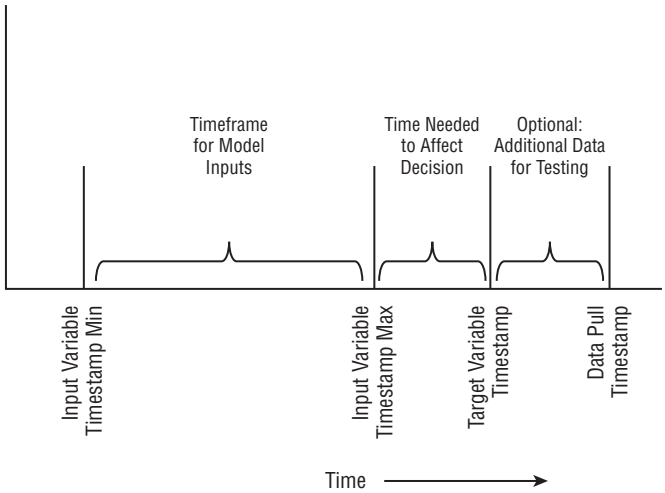


Figure 2-2: Timeline for defining target variable

The "Time Needed to Affect Decision" time gap can be critical in models, allowing time for a treatment to mature. For example, if you are building models to identify churn, the lead time to predict when churn might take place is critical to putting churn mitigation programs in place. You might, therefore, want to predict if churn will occur 30–60 days in the future, in which case there must be a 30-day gap between the most recent input variable timestamp and the churn timestamp.

The "Timeframe for Model Inputs" range has two endpoints. The "Input Variable Timestamp Max" is defined by the target variable and time needed to affect the decision. However, the minimum time is defined by the business objectives and practical considerations. Sometimes maintaining a long temporal

sequence of events is difficult and costly. Other times, the perception of domain experts is that information more than several years old is too stale and unlikely to be valuable in predicting the target variable.

The time gap at the right, "Additional Data for Testing," can sometimes be valuable during model validation. The best validation data are data that have not been seen by the modeling algorithms and emulate what the model will be doing when it goes live. Data collected subsequent to the target variable timestamp is certainly held-out data. If the model is biased by the particular timeframe when the target variable was computed or created, predictions for data collected after the target variable definition will reveal the bias.

Defining Measures of Success for Predictive Models

The determination of what is considered a good model depends on the particular interests of the organization and is specified as the business success criterion. The business success criterion needs to be converted to a predictive modeling criterion so the modeler can use it for selecting models.

If the purpose of the model is to provide highly accurate predictions or decisions to be used by the business, measures of accuracy will be used. If interpretation of the business is what is of most interest, accuracy measures will not be used; instead, subjective measures of what provides maximum insight may be most desirable. Some projects may use a combination of both so that the most accurate model is not selected if a less accurate but more transparent model with nearly the same accuracy is available.

Success criteria for classification and estimation models is described in more detail in Chapter 9.

Success Criteria for Classification

For classification problems, the most frequent metrics to assess model accuracy is Percent Correct Classification (PCC). PCC measures overall accuracy without regard to what kind of errors are made; every error has the same weight. Another measure of classification accuracy is the confusion matrix, which enumerates different ways errors are made, like false alarms and false dismissals. PCC and the confusion matrix metrics are good when an entire population must be scored and acted on. For example, if customers who visit a website are to be served customized content on the site based on their browsing behavior, every visitor will need a model score and a treatment based on that score.

If you are treating a subset of the population, a *selected population*, sorting the population by model score and acting on only a portion of those entities in the selected group can be accomplished through metrics such as Lift, Gain, ROC,

and Area under the Curve (AUC). These are popular in customer analytics where the model selects a subpopulation to contact with a marketing message, or in fraud analytics, when the model identifies transactions that are good candidates for further investigation.

Success Criteria for Estimation

For continuous-valued estimation problems, metrics often used for assessing models are R^2, average error, Mean Squared Error (MSE), median error, average absolute error, and median absolute error. In each of these metrics, you first compute the error of an estimate—the actual value minus the predicted estimate—and then compute the appropriate statistic based on those errors. The values are then summed over all the records in the data.

Average errors can be useful in determining whether the models are biased toward positive or negative errors. Average absolute errors are useful in estimating the magnitude of the errors (whether positive or negative). Analysts most often examine not only the overall value of the success criterion, but also examine the entire range of predicted values by creating scatterplots of actual target values versus predicted values or actual target values versus residuals (errors).

In principle, you can also include rank-ordered metrics such as AUC and Gain as candidates to estimate the success criteria, although they often are not included in predictive analytics software for estimation problems. In these instances, you need to create a customized success criterion.

Other Customized Success Criteria

Sometimes none of the typical success criteria are sufficient to evaluate predictive models because they do not match the business objective. Consider the invoice fraud example described earlier. Let's assume that the purpose of the model is to identify 100 invoices per month to investigate from the hundreds of thousands of invoices submitted. If the analyst builds a classification model and selects the model that maximizes PCC, the analyst can be fooled into thinking that the best model as assessed by PCC is good, even though none of the top 100 invoices are good candidates for investigation. How is this possible? If there are 100,000 invoices submitted in a month, you are selecting only 0.1 percent of them for investigation. The model could be perfect for 99.9 percent of the population and miss what you care about the most, the top 100.

In situations where there are specific needs of the organization that lead to building models, it may be best to consider customized cost functions. In the fraud example, you want to identify a population of 100 invoices that is maximally productive for the investigators. If the worst scenario for the investigators is to pursue a false alarm, a case that turns out to not be fraudulent at all, the model

should reflect this cost in the ranking. What modeling metric does this? No metric addresses this directly, although ROC curves are close to the idea. You could therefore select models that maximize the area under the ROC curve at the depth of 100 invoices. However, this method considers true alerts and false alarms as equally positive or negative. One solution is to consider the cost of false alarms greater than the benefit of a true alert; you can penalize false alarms ten times as much as a true alert. The actual cost values are domain-specific, derived either empirically or defined by domain experts.

Another candidate for customized scoring of models includes Return On Investment (ROI) or profit, where there is a fixed or variable cost associated with the treatment of a customer or transaction (a record in the data), and a fixed or variable return or benefit if the customer responds favorably. For example, if you are building a customer acquisition model, the cost is typically a fixed cost associated with mailing or calling the individual; the return is the estimated value of acquiring a new customer. For fraud detection, there is a cost associated with investigating the invoice or claim, and a gain associated with the successful recovery of the fraudulent dollar amount.

Note that for many customized success criteria, the actual predicted values are not nearly as important as the rank order of the predicted values. If you compute the cumulative net revenue as a customized cost function associated with a model, the predicted probability may never enter into the final report, except as a means to threshold the population into the "select" group (that is to be treated) and the "nonselect" group.

Doing Predictive Modeling Out of Order

While CRISP-DM is a useful guide to follow for predictive modeling projects, sometimes there are advantages to deviating from the step-by-step structure to obtain insights more quickly in a modeling project. Two useful deviations are to build models first and to deploy models before they are completed.

Building Models First

One of the biggest advantages of predictive modeling compared to other ways to analyze data is the automation that occurs naturally in many algorithms. The automation allows you to include many more input variables as candidates for inclusion in models than you could with handcrafted models. This is particularly the case with decision trees because, as algorithms, they require minimal data preparation compared to other algorithms.

Building models before Data Preparation has been completed, and sometimes even before Data Understanding has been undertaken, is usually problematic. However, there are also advantages to building models. You can think of it as an additional step in Data Understanding.

First, the predictive models will give the modeler a sense for which variables are good predictors. Some variables that would be good predictors if they were prepared properly would not, of course, show up. Second, building predictive models early gives the modeler a sense of what accuracy can be expected without applying any additional effort: a baseline.

Third, and perhaps one of the biggest insights that can be gained from building models early, is to identify models that predict too well. If models predict perfectly or much better than expected, it is an indication that an input variable contains future information subsequent to the target variable definition or contains information about the target variable itself. Sometimes this is obvious; if the address of a customer is only known after they respond to a mailing and the customer's state is included as an input, state values equal to NULL will always be related to non-responders. But sometimes the effect is far more subtle and requires investigation by the modeler to determine why a variable is inexplicably a great predictor.

Early Model Deployment

Model deployment can take considerable effort for an organization, involving individuals from multiple departments. If the model is to be deployed as part of a real-time transactional system, it will need to be integrated with real-time data feeds.

The Modeling stage of CRISP-DM is iterative and may take weeks to months to complete. However, even early in the process, the modeler often knows which variables will be the largest contributors to the final models and how those variables need to be prepared for use in the models. Giving these specifications and early versions of the models to the deployment team can aid in identifying potential obstacles to deployment once the final models are built.

Case Study: Recovering Lapsed Donors

This case study provides an example of how the business understanding steps outlined in this chapter can be applied to a particular problem. This particular case study, and data from the case study, will be used throughout the book for examples and illustrations.

Overview

The KDD-Cup is a data competition that became part of the annual Knowledge Discovery and Data Mining (KDD) conference. In the competition, one or more data sets, including record ID, inputs, and target variables, are made available to any who wish to participate. The 1998 KDD-Cup competition was a non-profit donation-modeling problem.

Business Objectives

The business objective was to recover lapsed donors. When donors gave at least one gift in the past year (0–12 months ago), they were considered active donors. If they did not give a gift in the past year but did give 13–24 months ago, they were considered lapsed donors. Of the lapsed donors, could you identify characteristics based on their historical patterns of behavior with the non-profit organization? If these donors could be identified, then these lapsed donors could be solicited again and lapsed donors unlikely to give again could be ignored, increasing the revenue to the organization.

The test mailing that formed the basis for the competition had already been completed so considerable information about the recovery of lapsed donors was already known. The average donation amount for all lapsed donors who were mailed was $0.79 and the cost of the mail campaign was $0.68 per contact. So soliciting everyone was still profitable, and the amount of profit was approximately $11,000 per 100,000 lapsed donors contacted.

But could this be improved? If a predictive model could provide a score, where a higher score means the donor is more likely to be a good donor, you could rank the donors and identify the best, most profitable subset.

Data for the Competition

The recovery concept was to mail to a random subset of lapsed donors (191,779 of them). Approximately 5.1 percent of them responded to the mailing. The unit of analysis was a donor, so each record was a lapsed donor, and the attributes associated with each donor included demographic information and historic patterns of giving with the non-profit organization. Many derived variables were included in the data, including measures that summarized their giving behavior (recent, minimum, maximum, and average gift amounts), RFM snapshots at the time of each of the prior contacts with each donor, their socioeconomic status, and more.

The Target Variables

Two target variables were identified for use in modeling: TARGET_B and TARGET_D. Responders to the recovery mailing were assigned a value of 1 for

TARGET_B. If the lapsed donor did not respond to the mailing, he received a TARGET_B value of 0. TARGET_D was populated with the amount of the gift that the lapsed donor gave as a result of the mailing. If he did not give a gift at all (i.e., TARGET_B = 0), the lapsed donor was assigned a value of 0 for TARGET_D.

Thus, there are at least two kinds of models that can be built. If TARGET_B is the target variable, the model will be a binary classification model to predict the likelihood a lapsed donor can be recovered with a single mailing. If TARGET_D is the target variable, the model predicts the amount a lapsed donor gives from a single recovery campaign.

Modeling Objectives

The next step in the Business Understanding process is translating the business objective—maximizing net revenue—to a predictive modeling objective. This is a critical step in the process because, very commonly, the business objective does not translate easily or directly to a quantitative measure.

What kinds of models can be built? There are two identifying measures that are candidate target variables. First, TARGET_B is the response indicator: 1 if the donor responded to the mailing and 0 if the donor did not respond to the mailing. Second, TARGET_D is the amount of the donation if the donor responded. The value of TARGET_D is $0 if the donor did not respond to the mailing.

If a TARGET_B model is built, the output of the prediction (the Score) is the probability that a donor will respond. The problem with this number, however, is that it doesn't address directly the amount of a donation, so it won't address directly net revenue. In fact, it considers all donors equally (they all are coded with a value of 1 regardless of how much they gave) and it is well known that donation amounts are inversely correlated with likelihood to respond. These models therefore would likely favor low-dollar, lower net revenue donors.

On the other hand, a TARGET_D model appears to address the problem head on: It predicts the amount of the donation and therefore would rank the donors by their predicted donation amount and, by extension, their net revenue (all donors have the same fixed cost). But TARGET_D models have a different problem: You only know the amount given for those donors who gave. The vast majority of donors did not respond and therefore have a value of TARGET_D equal to $0. This is a very difficult distribution to model: It has a spike at $0 and a much smaller, heavily skewed distribution for values greater than $0. Typically in these kinds of problems, you would create a model for only those who donated, meaning those with TARGET_B = 1 or, equivalently, those with TARGET_D greater than 0.

This introduces a second problem: If you build a model for just the subset of those who donated, and the model would be applied to the entire population of lapsed donors, are you sure that the predicted values would apply well to those who don't donate? The model built this way is only built to predict the donation amount of donors who actually gave.

This problem therefore is the classic *censored regression* problem where both the selection stage (TARGET_B) and the amount stage (TARGET_D) need to be modeled so that a complete prediction of an expected donation amount can be created.

One solution, though certainly not the only one, would be to build both TARGET_B and TARGET_D models, and then multiply their prediction values to estimate the expected donation amount. The TARGET_B prediction is the likelihood that the donor will give, and the TARGET_D prediction is the amount the donor will give. After multiplying, if a donor is very likely to give and is likely to give in a large amount, that donor will be at the top of the list. Donors will be considered equally likely to give if, for example, their propensity to give is 0.1 (10 percent likelihood) and the predicted donation amount is $10 or their propensity is 0.05 (5 percent likelihood) and their predicted donation amount is $20. In both cases, the score is 1.0.

Score = P(TARGET_B = 1) × Estimated TARGET_D

Model Selection and Evaluation Criteria

After building models to predict TARGET_B or TARGET_D, how do you determine which model is best? The answer is to use a metric that matches as closely as possible to the business objective. Recall that the business objective is to maximize cumulative net revenue. If you build a model to predict TARGET_B, computing percent correct classification is a measure of accuracy, but won't necessarily select the model that best matches the business objective. Computing gain or lift is closer, but still doesn't necessarily match the business objective. But if you instead compute the cumulative net revenue, the business objective itself, and select the model that maximizes the cumulative net revenue, you will have found the best model according to the business objectives.

The procedure would be this:

1. Score all records by the multiplicative method, multiplying the TARGET_B prediction by the TARGET_D prediction.
2. Rank the scores from largest to smallest.
3. Compute the net revenue for each donor, which is the actual TARGET_D value minus $0.68.
4. Sum the net revenue value as you go down the ranked list.
5. When the net revenue is maximized, this is by definition the maximum cumulative net revenue.
6. The model score associated with the record that generated the maximum cumulative net revenue is the score you should mail to in subsequent campaigns.

Note that once you compute a score for each record based on the model predictions, the scores themselves are not used at all in the computation of cumulative net revenue.

Model Deployment

In subsequent mailings to identify lapsed donors to try to recover, you only need to generate model scores and compare the scores to the threshold defined in the model evaluation criterion defined earlier.

Case Study: Fraud Detection

This case study provides an example of how the business understanding steps outlined in this chapter can be applied to invoice fraud detection. It was an actual project, but the name of the organization has been removed upon their request. Some details have been changed to further mask the organization.

Overview

An organization assessed invoices for payment of services. Some of the invoices were submitted fraudulently. The organization could afford to investigate a small number of invoices in detail. Predictive modeling was proposed to identify key invoices to investigate.

Business Objectives

In this invoice fraud example, the very definition of fraud is key to the modeling process. Two definitions are often considered in fraud detection. The first definition is the strict one, labeling an invoice as fraudulent if and only if the case has been prosecuted and the payee of the invoice has been convicted of fraud. The second definition is looser, labeling an invoice as fraudulent if the invoice has been identified as being worthy of further investigation by one or more managers or agents. In the second definition, the invoice has failed the "smell test," but there is no proof yet that the invoice is fraudulent.

Note that there are advantages and disadvantages to each option. The primary advantage of the first definition is clarity; all of those invoices labeled as fraud have been proven to be fraudulent. However, there are also several disadvantages. First, some invoices may have been fraudulent, but they did not meet the standard for a successful prosecution. Some may have been dismissed on technicalities. Others may have had potential but were too complex to prosecute efficiently and therefore were dropped. Still others may have shown potential, but the agency did not have sufficient resources to complete the investigation.

On the other hand, if you use the second definition, many cases labeled as fraud may not be fraudulent after all, even if they appear suspicious upon first glance. In other words, some "fraudulent" labels are created prematurely; if we had waited long enough for the case to proceed, it would have been clear that the invoice was not fraudulent after all.

In this project, the final determination was made that the strict definition should be used.

Data for the Project

Hundreds of thousands of invoices were available for use in modeling, more than enough for building predictive models. However, there were relatively few labeled fraud cases because of the strict definition of fraud. Moreover, just because an invoice was not labeled as being fraudulent didn't mean that the invoice was definitively not fraudulent. Undoubtedly, there were invoices labeled as "not fraud" that were actually fraudulent. The problem is that the modeling algorithms don't know this and believe the labels of fraud and non-fraud are completely accurate.

Let's assume that the invoice fraud rate was 1 percent. This means that 1 of every 100 invoices is fraudulent. Let's assume, too, that only half of the fraudulent invoices are identified and prosecuted, leaving the other half unprosecuted. For every 100,000 invoices, 500 are identified as fraudulent and 500 are left in the data with the label "0" even though they are fraudulent. These mislabeled cases can at best confuse models and at worst cause models to miss patterns associated with fraud cases because so many invoices with the same pattern of behavior are also associated with non-fraud invoices.

Modelers, during the Business Understanding stage of the project, determined that the non-fraud population should be sampled so that the likelihood that a mislabeled invoice would be included in the data was greatly reduced.

The Target Variables

The most obvious target variable is the indicator that an invoice is fraudulent (1) or not fraudulent (0). The organization decided, however, to be more precise in their definition by identifying six different kinds of fraud, each of which would be investigated differently. After an initial cluster analysis of the fraud types, it was determined that two of the six fraud types actually overlapped considerably with other fraud types, and therefore the number of fraud types used in modeling was reduced to four. The final target variable therefore was a five-level categorical variable.

Modeling Objectives

The models for this project were to be multi-valued classification models with four fraud classes and one non-fraud class. Misclassification costs were set up when possible—not all algorithms support misclassification costs—so that misclassifying non-fraud invoices as fraud received four times the weight as the converse. Moreover, misclassifying one fraud type as another fraud type did not generate any additional penalty.

Model Selection and Evaluation Criteria

Models were selected to identify the 100 riskiest invoices each month. One key concern was that the workload generated by the model had to be productive for the investigators, meaning that there were few false alarms that the investigators would ultimately be wasting their time pursuing. Therefore, a customized cost function was used to trade off true alerts with false alarms. False alarms were given four times the weight of true alerts. The model that was chosen was the one that performed best on the top 100 invoices it would have selected. Scores were computed according to the following process:

1. Compute a score for each invoice where the score is the maximum of the four probabilities.
2. Sort the scores from high to low.
3. Select the top 100 scores.
4. If the invoice in each record was fraudulent, give the record a score of +1. If the record was not fraudulent, give the record a score of –4.
5. Add the +1 or –4 values for each record. The highest score wins.

With a 4:1 ratio in scoring false alarms to true alerts, if 80 of the top 100 invoices scored by the maximum probability were identified as fraudulent, the weighted score is 0. The actual score, however, does not indicate how well the models predicted fraud, although you could look at the individual probabilities to report which fraud type the invoice was flagged for and how large the probability for the record was.

Model Deployment

Deployment of the model was simple: Each month, the model generated scores for each invoice. The 100 top-scoring invoices were flagged and sent to investigators

to determine if they were potentially fraudulent, and if so, they were prosecuted judicially.

Summary

Setting up predictive modeling problems requires knowledge of the business objectives, how to build the data, and how to match modeling objectives to the business objectives so the best model is built. Modelers need to be a part of setting up the problem to ensure that the data and model evaluation criteria are realistic. Finally, don't be afraid to revisit and modify business objectives or modeling objectives as the project unfolds; lessons learned during the modeling process can sometimes reveal unexpected problems with data or new opportunities for improving decisions.

CHAPTER

3

Data Understanding

Now that data has been collected for a modeling project, it needs to be examined so the analyst knows what is there. In many situations, the analyst is the first person to even look at the data as compiled into the modeling table in-depth. The analyst, therefore, will see all of the imperfections and problems in the data that were previously unknown or ignored. Without Data Understanding, the analyst doesn't know what problems may arise in modeling.

Data Understanding, as the first analytical step in predictive modeling, has the following purposes:

- Examine key summary characteristics about the data to be used for modeling, including how many records are available, how many variables are available, and how many target variables are included in the data.
- Begin to enumerate problems with the data, including inaccurate or invalid values, missing values, unexpected distributions, and outliers.
- Visualize data to gain further insights into the characteristics of the data, especially those masked by summary statistics.

What the Data Looks Like

After data has been imported into the predictive analytics software tool, the data can be summarized and assessed: variables in the columns and records in the rows. Variables can be numeric, strings, and dates. Before using data, you need to ensure that it is typed properly, especially if data is read into the predictive analytics software as a flat file where the software must guess the data types from the data itself. For example, ZIP codes are numbers, but are not numeric; they should be considered strings so that leading zeros are not stripped out.

Dates are particularly cumbersome in software because so many different formats can be used. Some software doesn't handle date operations particularly well. Nevertheless, dates can provide tremendous amounts of information in predictive modeling so they should be handled with care and precision.

Continuous variables are numbers whose values can, in principle, range from negative infinity to positive infinity. These numbers can be integers or real values, sometimes called *ints* and *doubles* respectively. For example, Age, Income, Profit/Loss, Home Value, Invoice Amount, Visit Count, and DaysSinceLastVisit are all continuous variables. Some of the values of the variables may be restricted in some way. Some may be positive only, some positive or zero, some negative, some limited in magnitude (at least practically).

Predictive modelers and predictive analytics software usually think of continuous variables as one group of values, without differentiating subgroups of continuous variables, such as interval variables and ratio variables.

Categorical variables have a limited number of values whose intent is to label the variable rather than measure it. Examples of categorical variables include State, Title Code, SKU, and target variables such as Responder. Categorical variables are sometimes called nominal variables or discrete variables. Flag variables or binary variables are often used as names for categorical variables having only two values, like Gender (M,F), responses to questions (Yes and No), and dummy variables (1 and 0).

Single Variable Summaries

After data has been loaded into any software tool for analysis, it is readily apparent that for all but the smallest data sets, there is far too much data to be able to make sense of it through visual inspection of the values.

At the core of the Data Understanding stage is using summary statistics and data visualization to gain insight into what data you have available for modeling. Is the data any good? Is it clean? Is it representative of what it is supposed to measure? Is it populated? Is it distributed as you expect? Will it be useful for

building predictive models? These are all questions that should be answered before you begin to build predictive models.

The simplest way to gain insight into variables is to assess them one at a time through the calculation of summary statistics, including the mean, standard deviation, skewness, and kurtosis.

Mean

The mean of a distribution is its average, simply the sum of all values for the variable divided by the count of how many values the variable has. It is sometimes represented by the Greek symbol mu, μ.

The mean value is often understood to represent the middle of the distribution or a typical value. This is true when variables match a normal or uniform distribution, but often this is not the case.

Standard Deviation

The Standard Deviation measures the spread of the distribution; a larger standard deviation means the distribution of values for the variable has a greater range. Most often in predictive modeling, the standard deviation is considered in the context of normal distributions. The Greek symbol sigma, σ, is often used to denote the standard deviation.

The Normal Distribution

Normal distributions are fundamental to data analysis. The shape of the normal distribution is shown in Figure 3-1 and because of the shape, the normal distribution is often called the *bell curve*. In some disciplines, it is called the *Gaussian* distribution. Many algorithms assume normal distributions either explicitly or implicitly. Considerable effort is expended by some analysts to find transformations that change the variables so that they become normally distributed.

Normal distributions have some additional properties that are worth noting:

- The distribution is symmetric.
- The mean value is the most likely value to occur in the distribution.
- The mean, median, and mode are all the same value.
- Approximately 68 percent of the data will fall between the mean and +/– 1 standard deviation from the mean.
- Approximately 95 percent of the data will fall between the mean and +/– 2 standard deviations from the mean.

- Approximately 99.7 percent of the data will fall between the mean and +/– 3 standard deviations from the mean.
- The distribution does not have a definitive, fixed range. However, because data is so unlikely to fall outside of three standard deviations from the mean, these values are often considered to be outliers.

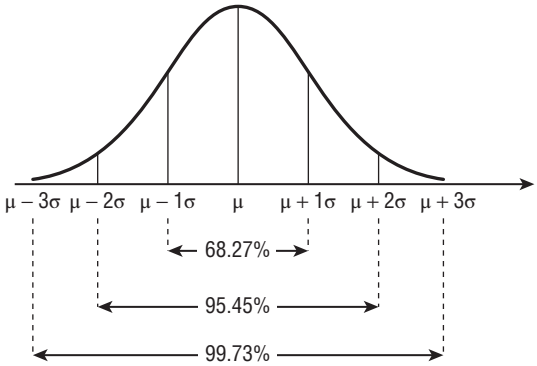


Figure 3-1: Standard deviation

Uniform Distribution

The uniform distribution assumes that a variable has a fixed, finite range. Figure 3-2 shows a conceptualization of a uniform distribution. The mean value is in the center of the range of the variable. Standard deviations can be computed but are rarely used in predictive modeling algorithms.

True uniform distributions are rare in behavioral data, though some distributions are close enough to be treated as uniform.

In uniform distributions, the mean can also be computed by simply computing one-half of the difference between the maximum minus the minimum of the values of the variable.

Some interesting properties of uniform distributions include:

- The distribution is symmetric about the mean.
- The distribution is finite, with a maximum and minimum value.
- The mean and midpoint of the distribution are the same value.
- Random number generators create uniform random distributions, most often in the range 0 to 1 as their default.

Sometimes the distribution of a variable is primarily uniform but will have gaps between the denser part of the distribution and the maximum or minimum. These outliers can cause problems for some algorithms that are sensitive to outliers and therefore should be noted during Data Understanding.

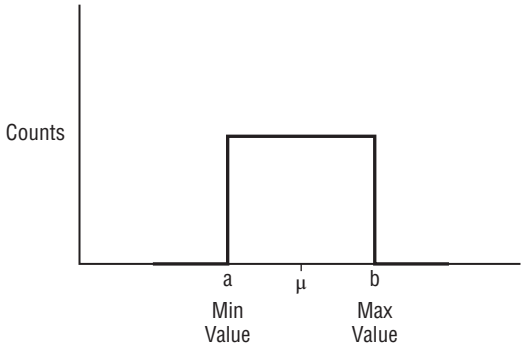


Figure 3-2: Uniform distribution

Applying Simple Statistics in Data Understanding

Consider computing simple statistics for some key variables in the KDD Cup 1998 data shown in Table 3-1: As a part of data understanding, there are several measures to examine to understand the characteristics of the data. The purpose in examining these statistics is to verify the data is distributed as expected and to identify potential problems in the data that should be cleaned or corrected during the Data Preparation stage.

Table 3-1: Summary Statistics for a Subset of Continuous Variables for KDD Cup 1998 Data

VARIABLE	MINIMUM	MAXIMUM	MEAN	STANDARD DEVIATION	NUMBER OF MISSING VALUES
AGE	1	98	61.61	16.66	23665
RAMNTALL	13	9485	104.48	118.58	0
NGIFTALL	1	237	9.60	8.55	0
LASTGIFT	0	1000	17.31	13.96	0
LASTDATE	9503	9702	9548.10	49.24	0
FISTDATE	0	9603	9135.65	320.39	0
NEXTDATE	7211	9702	9151.02	294.26	9973
TIMELAG	0	1088	8.09	8.21	9973
AVGGIFT	1.2857	1000	13.35	10.77	0
CONTROLN	1	191779	95778.18	55284.60	0
TARGET_B	0	1	0.05	0.22	0
TARGET_D	0	200	0.79	4.43	0

First, look at the minimum and maximum values. Do these make sense? Are there any unexpected values? For most of the variables, the values look fine. RAMNTALL, the total donation amount given by a donor to the non-profit organization over his or her lifetime, ranges from $13 to $9,485. These are plausible.

Next, consider AGE. The maximum value is 98 years old, which is high, but believable. The minimum value, however, is 1. Were there 1-year-old donors? Does this make sense? Obviously, 1-year-olds did not write checks or submit credit card donations to the organization. Could it mean that donations were given in the name of a 1-year-old? This could be, but will require further investigations by someone to provide the necessary information. Note, however, that we only know that there is at least one 1-year-old in the data; this summary does not indicate to us how many 1-year-olds there are.

Third, consider FISTDATE, the date of the first gift to the organization. The maximum value is 9603, meaning that the most recent date of the first gift in this data is March 1996. Given the timeframe for the data, this makes sense. But the minimum value is 0. Literally, this means that the first donation date was the year of Christ's birth. Obviously this is not the intent. The most likely explanation is that the value was unknown but at some time a value of 0 was used to replace the missing or null value.

Next consider the standard deviations. For AGE, the standard deviation is about 17 years compared to the mean of 61.61 years old. If AGE were normally distributed, you would expect 68 percent of the donors to have AGE values between 44 years old and 78 years old, 95 percent of the donors to have AGE values between 27 and 95 years old, and 99.7 percent of the AGE values to range between 10 and 112 years old. It is obvious from these values, where the maximum value is less than the value assumed by the normal distribution for three standard deviations above the mean, that the data is not normally distributed and therefore the standard deviation is not an accurate measure of the true spread of the data. This can be seen even more clearly with RAMNTALL where one standard deviation below the mean is negative, less than the minimum value (and a value that makes no operational sense).

The target variables, TARGET_B and TARGET_D, are also shown in the table. TARGET_B, the indication whether a donor responded to the recapture mailing (a value of 1) or didn't respond to the recapture mailing (a value of 0) is a dummy variable, but because it was coded with values 0 and 1, it can be treated as a numeric variable for the purposes of calculating summary statistics. The minimum and maximum values make sense and match what you expect. The mean value is 0.05, which is the response rate to the mailing. The standard deviation doesn't apply here.

For TARGET_D, notice that the maximum gift in response to the mailing was $200 and the average was $0.79 (79 cents). The minimum value is interesting, however. Were there individuals who gave $0 in response to the mailing? A

quick investigation would reveal the answer is "no," but non-responders were assigned TARGET_D values of 0. This isn't exactly correct: There could be a difference between a lapsed donor not responding and a lapsed donor responding to the mailing but not giving any money at all (perhaps indicating their support for the organization even while not being able to donate at this time). However, assigning the value 0 to these is not unreasonable and is done frequently in these situations; it is a decision that has to be made during Business Understanding or Data Understanding by analytics and domain experts so the appropriate values are assigned to TARGET_D.

The last column in Table 3-1 summarizes how many missing values occurred in the variable. A missing value could have been coded as NULL, an empty cell, or some other indicator of missing. How missing values are interpreted is software dependent. Most predictive analytics software will handle null values and treat them as missing. But other codings of missing, such as 99 or 999, –1, an empty string, and others could cause deceptive values. You have already seen this in FISTDATE: The 0 for the minimum value, the recoding of missing to the value 0, distorts the mean value. AGE is likewise distorted because of the 1-year-olds in the data. These are all values that need to be investigated and possibly corrected before modeling begins.

Skewness

Skewness measures how balanced the distribution is. A normal distribution has a skewness value of 0. But so does a uniform distribution or any balanced distribution. Positive skew—skew values greater than zero—indicates that the distribution has a tail to the right of the main body of the distribution. Negative skew indicates the converse: The distribution has a tail to the left of the main body. Examples of the possible shape of distributions with positive and negative skew are shown in Figure 3-3.

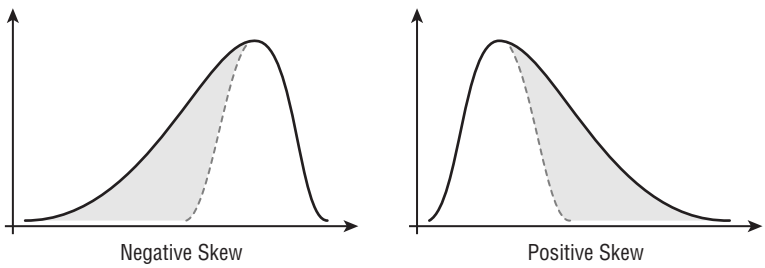


Figure 3-3: Positive and negative skew

Why does measuring skew matter? Some algorithms assume variables have normal distributions, and significant deviations from those assumptions affect

the model accuracy or at least the model interpretation. Significant deviations from normal result in the mean and standard deviation having deceptive values. Examining how skewed a distribution is provides one way you can determine how much the variable deviates from normal.

How much skew indicates significant deviation from normal? There is no fixed rule on what values of skewness are too high. However, several rules of thumb are often used. The first indication of positive skew is when the mean is larger than the median (to be defined in the next section). An additional indicator is when the mode is less than the median, which is also less than the mean. In Table 3-2, RAMNTALL has the class indicators of positive skew: The mode is less than the median, and the median is less than the mean. Note the skewness value is 13.9, a large positive value.

Some software also provides a measure of the standard error for skewness. The larger the sample, the smaller the standard error is, generally.

Table 3-2: Skewness Values for Several Variables from KDD Cup 1998

VARIABLE	MODE	MEDIAN	MEAN	SKEWNESS	SKEWNESS STANDARD ERROR
RAMNTALL	20	78	104.48	13.91	0.008
MAXRAMNT	15	17	20.00	99.79	0.008
LASTGIFT	15	15	17.31	16.29	0.008
NUMPRM12	13	12	12.86	2.99	0.008
CARDPROM	6	18	18.44	0.15	0.008
NUMPROM	13	47	46.97	0.44	0.008

Often, values of skew greater than a magnitude of 2 or 3 are considered significant. Some even tie significantly large skew to a value twice the skewness standard error; of course, when the sample size is large, even skewness values considered small would show as significant. In Table 3-2, the standard error is only 0.008 due to the sample size (95,412), so any skew value greater than 0.016 could be considered significant.

Positive skew is observed more often than negative skew, especially in applications with monetary variables. Home values, income, credit card transaction amounts, and loan values are typically positively skewed monetary variables. The positive skew in these variables results in their mean values no longer representing a typical value for the variable. This is the reason that home values, for instance, are reported using the median rather than the mean; the mean is inflated by the large values to the right of the distribution.

Because of the adverse effect skew has on how some algorithms behave, many analysts consider identifying large skew (positive or negative) so that during Data Preparation the variable can be corrected to remove the skew.

Kurtosis

Kurtosis measures how much thinner or fatter the distribution is compared to the normal distributions. Figure 3-4 shows three distributions: On the left is a normal distribution; the center is a skinnier-than-normal distribution (called leptokurtic); and on the right is a fatter-than-normal distribution (called platykurtic). A variable that is normally distributed will have a kurtosis value equal to 3, a leptokurtic distribution less than 3, and a platykurtic distribution greater than 3.

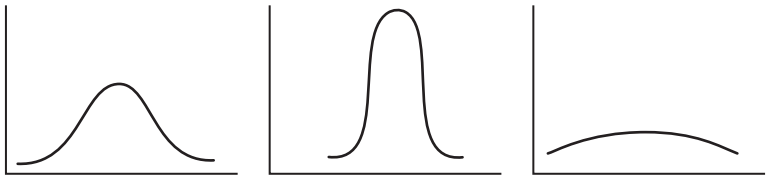


Figure 3-4: Types of skewness

Kurtosis is always positive. However, kurtosis is sometimes reported in software as *excess kurtosis* even if the label of the statistic is still kurtosis. Excess kurtosis is the difference between the kurtosis value assumed for normal distributions (3) and the value computed. A value of excess kurtosis that is greater than 0 indicates a platykurtic distribution (fat), whereas an excess kurtosis value that is less than 0 indicates a leptokurtic distribution (skinny).

If a variable is uniformly distributed, it will appear to be leptokurtic because there is not a sharp peak in the distribution. If a variable has outliers both positive and negative, such as a profit/loss variable, the distribution will have a high peak with big tails and therefore have high kurtosis.

Kurtosis is rarely used in predictive analytics for anything more than providing insight into the shape of the distribution of a variable. However, there are some situations where understanding kurtosis can be valuable. For instance, consider a distribution that is skinny with long symmetric tails on both sides of the distribution, like the shape of Figure 3-5. The skew will be 0 in this distribution because the tails are symmetric and balance each other out. The kurtosis, however, will be very large (platykurtic) because of the large tails (in spite of the tall spike in the center of the distribution). This is evidence therefore that the variable's standard deviation will not be representative of the true spread in the data for algorithms that use standard deviation or variance in the model. The modeler should consider transformations that would correct this problem.

The kurtosis values for the same variables from Table 3-2 are shown in Table 3-3. The first three variables show a strong indication that not only is there positive skew—a tail in the distribution to the right—but there is also high excess kurtosis, so the distribution appears to be much fatter than a normal distribution.

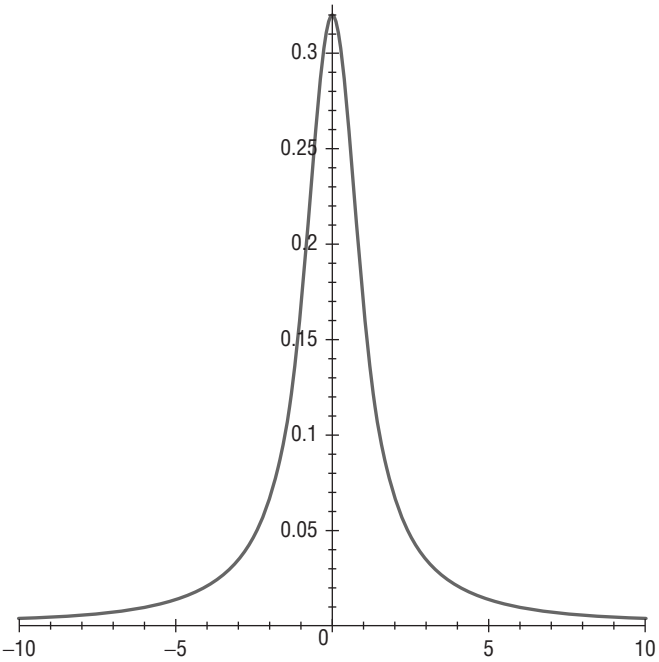


Figure 3-5: Big-tailed distribution

Table 3-3: Kurtosis Values for Several Variables from KDD Cup 1998

VARIABLE	MEAN	SKEWNESS	EXCESS KURTOSIS	KURTOSIS STANDARD ERRORR
RAMNTALL	104.48	13.91	652.64	0.016
MAXRAMNT	20.00	99.79	17673.12	0.016
LASTGIFT	17.31	16.29	728.44	0.016
NUMPRM12	12.86	2.99	13.14	0.016
CARDPROM	18.44	0.15	–0.93	0.016
NUMPROM	46.97	0.44	0.02	0.016

Rank-Ordered Statistics

The statistics described above assumed a normal distribution—the mean, standard deviation, skew, and kurtosis. Outliers can have a large influence on each of these values, misleading an analyst about the true characteristics of the variables.

However, there are other statistics that do not share this problem. If the variable is sorted from smallest to largest variable and each value is associated with

the percentile of its rank-ordered value, insights can be gained regardless of the shape of the distribution, regardless of how many or how extreme the outliers are. The most common rank statistics are listed in Table 3-4.

Table 3-4: Rank-Ordered Metrics

PERCENTILE	METRIC
0th	Minimum
25th	1st quartile
50th	2nd quartile; the median
75th	3rd quartile
100th	Maximum

These are considered *robust statistics* because small changes in the data do not produce large changes in the statistics. This is not the case with the mean. The variable LASTGIFT has a mean value of 17.3 (from Table 3-1). A single additional LASTGIFT value of $10,000 would change the mean from $17.3 to $19.3 (if there were 5,000 records in the data). However, the median would not change at all even with this single large outlier, which is an additional reason to use median home value as the measure of a typical home price rather than the mean. If a rich celebrity would build a new 30 million dollar home in Los Angeles county, the median home value in the county would not change, though the mean will tick upward.

The median is defined as the value that is exactly 50 percent of the way from the minimum to maximum value of the variable (or *vice versa*). If there are an even number of rows, the median is the average of the two middle numbers. If, for example, you have the list of sorted numbers 1, 4, 8, 21, 34, 45, and 50, the median is 21, the fourth of seven numbers in the list. For the list 1, 4, 8, 21, 34, and 45, the median is the average of 8 and 21, or 14.5 mean

The median is by far the most well-known of the rank-ordered statistics and is commonly included in summary statistics in software, though often with a disclaimer. Because computing the median, in particular, and quartiles, in general, requires the data first be sorted, computing the median can be computationally expensive when there are large numbers of rows. Additionally, this operation must be repeated separately for each and every variable whose summary statistics are being computed. The mean, standard deviation, skewness, and kurtosis do not require sorting the data; only summing operations are needed.

Quartiles are a special case of the general rank-ordered measure called *quantiles*. Other commonly used quantiles are listed in Table 3-5. The consensus of predictive analytics software is to use quartile, quintile, decile, and percentile. However, there is no consensus for the name of the label for 20 bins; three of the names that appear most often are shown in the table. Of course, if you know

the deciles, you also know the quintiles because quintiles can be computed by summing pairs of deciles. If you know the percentiles, you also know all of the other quantiles.

Table 3-5: Quantile Labels

QUANTILE LABEL	NUMBER OF BINS	PERCENT OF POPULATION
Quartile	4	25
Quintile	5	20
Decile	10	10
Demi-decile		
Vingtile	20	5
Twentile		
Percentile	100	1

The analogous measure to standard deviation using quartiles is the *Inter-Quartile Range* (IQR): the difference between the 3rd quartile value and the 1st quartile value, or in other words, the range between the 25th and 75th percentiles. This, too, is a robust statistic as individual outliers have little effect on the range; each data point shifts the values of the 1st and 3rd quartiles (Q1 and Q3 respectively) by, at most, the difference in value of the next variable value in the sorted list.

This doesn't mean that outliers cannot be labeled using the IQR. Some practitioners use a rule of thumb that values more than 1.5 times the IQR from the upper or lower quartiles are considered outliers. Table 3-6 shows the same variables shown in Tables 3-2 and 3-3, this time with quantile information. For example, LASTGIFT, while having a huge 1000 maximum value, has a median of 15, a 3rd quartile value of 20, a 1st quartile value of 10, and therefore an IQR (not in the table) equal to 10 (20–10). Any value of LASTGIFT greater than 35 may be considered an outlier because it exceeds 1.5 times the IQR plus the 3rd quartile value. As it turns out, that includes 3,497 records of the 95,412, or 3.7 percent of the population. No records are 1.5 times the IQR below the 1st quartile because outliers would have to be negative, less than the minimum value of LASTGIFT. Only NUMPRM12 has lower outliers, those values below 8.

The fewest IQR outliers were for CARDPROM and NUMPROM, in large part because they both have small values of excess kurtosis (refer to Table 3-3).

Table 3-6: Quartile Measures for Several Variables from KDD Cup 1998

VARIABLE	MINIMUM	Q1	MEDIAN	Q3	MAXIMUM	–1.5*IQR OUTLIERS	+1.5*IQR OUTLIERS
RAMNTALL	13	40	78	131	9485	-96.5	267.5
MAXRAMNT	5	14	17	23	5000	0.5	36.5
LASTGIFT	0	10	15	20	1000	–5	35
NUMPRM12	1	11	12	13	78	8	16
CARDPROM	1	11	18	25	61	–10	46
NUMPROM	4	27	47	64	195	–28.5	119.5

Categorical Variable Assessment

Categorical or nominal variables are assessed by counting the number of occurrences for every level, informing the modeler which values occur frequently and infrequently in the data.

There are several issues typically addressed with frequency counts.

- Do the values make sense? Are there values that are unexpected, erroneous, or indicators of missing or unknown values?
- Are there missing values? How many? How are they coded?
- How many levels are there? Is there only one value? Are there more than 50 or 100 levels?
- What is the mode of the levels?

Table 3-7 shows the counts for variable RFA_2 with 14 levels in the 95,412 rows of data. You would expect that each of these categorical values would be populated with approximately 6815 rows (95,412 ÷ 14). While the values in the Count column are not equal, each value is populated with at least 900 records: a good amount.

Table 3-7: Frequency Counts for Variable RFA_2

RFA_2	% OF POPULATION	COUNT
L1E	5.2	4,911
L1F	31.8	30,380

Continues

Table 3-7 (*continued*)

RFA_2	% OF POPULATION	COUNT
L1G	13.0	12,384
L2E	5.2	4,989
L2F	11.5	10,961
L2G	4.8	4,595
L3D	2.6	2,498
L3E	8.1	7,767
L3F	3.7	3,523
L3G	1.6	1,503
L4D	5.2	4,914
L4E	4.2	3,972
L4F	2.2	2,100
L4G	1.0	915

Or consider a second example, with the variable STATE in Table 3-8. These are only the first 15 of 58 values for STATE—obviously the variable contains more than states, but also contains U.S. territories. Part of the task of the analyst in this stage is to confirm that values AA, AE, AP, and AS are correct. Of additional concern, however, are the low counts for values AA, AE, and CT, but especially AS, DC, and DE. The last three have so few examples (1 or 3) that no reasonable inference can be made about their relationship to a target variable. This problem will need to be considered during data preparation.

Table 3-8: Frequency Counts for Variable STATE

STATE	% OF POPULATION	COUNT
AA	0.0	18
AE	0.0	15
AK	0.3	282
AL	1.8	1,705
AP	0.1	81
AR	1.1	1,020
AS	0.0	1
AZ	2.5	2,407

CA	18.2	17,343
CO	2.1	2,032
CT	0.0	23
DC	0.0	1
DE	0.0	3
FL	8.8	8,376
GA	3.6	3,403

A third example is shown in Table 3-9. This categorical variable, PEPSTRFL, has one value that is populated, with the remaining missing. On the surface, it appears that this variable may be uninteresting with only one value, but can be cleaned during Data Preparation.

Table 3-9: Frequency Count for Variable with One Level

PEPSTRFL	% OF POPULATION	COUNT
	52.6	50,143
X	47.4	45,269

A convenient summary of categorical variables is shown in Table 3-10, including most of the key measures that indicate data cleaning must occur with the variable. These key measures are the following: existence of missing values, high-cardinality, and low counts for some levels.

High cardinality presents two problems for predictive modeling. First, some software will automatically convert categorical variables to numeric variables by creating a dummy variable for every level of the categorical variable. If the number of levels is large, the number of dummy variables is equally large and can result in an overwhelming number of input variables. For example, in the KDD Cup 1998 data, there are 23 RFA variables: RFA_2, RFA_3, RFA_4, and so on, through RFA_24. Twenty-two (22) of these (all except for RFA_2) have 70 to more than 100 levels each. Therefore, if these are converted to dummy variables by some software, there are no longer just 23 variables but more than 2,000 variables that become inputs to models, a huge number of variables to consider in modeling. Some data prep will likely be needed to help with this problem.

Second, the more levels in the categorical variable, the fewer the number of records that can populate each level. If there are too few (for example, fewer than 30 records per level), inferences made by the modeling algorithms about the relationship between that level and the target variable are noisy at best, and can lead to model overfit. Low counts, of course, can occur even with relatively

few levels, but the more levels you have in the variable, the more likely it is that several or even most levels will have low counts. Rules of thumb for handling high-cardinality variables are described in Chapter 4.

Table 3-10: Frequency Count Summary Table for Several Variables

VARIABLE	LEVELS	MODE	POPULATED	MISSING	MISSING (%)
STATE	57	CA	95,412	0	0.0
GENDER	7	F	92,455	2,957	3.1
PEPSTRFL	2		45,269	50,143	52.6
RFA_2	14	L1F	95,412	0	0.0
RFA_3	71	A1F	93,462	1,950	2.0
RFA_2A	4	F	95,412	0	0.0

Data Visualization in One Dimension

Data visualization is a graphical representation of the data. Like statistical summaries, visualization also summarizes data but does so in a way that conveys much more information and context of the data. Some ideas that are difficult to convey in numbers from a table but can be seen easily in graphical form are distances and density. Seeing distances between some data values and others reveals if data points or groups of data points are outliers or unexpectedly different.

Visualizing density shows how and where data is distributed, revealing if the data is normal, uniform, or neither. Visualization can be particularly effective in identifying high skew or excess kurtosis, answering questions such as, "Is skew high because of a small group of outliers or due to a long tail?"

Visualization is done primarily in two stages of a modeling project. First, it can be done effectively in the beginning of a project, during the Data Understanding stage to help the modeler verify the characteristics of the data. Some one-dimensional data visualization methods can be done for all variables in the data as a part of the data verification step. Other methods—two or more dimensional visualization methods—require manipulation of settings that render them difficult to use in any automated manner.

Data visualization is also done after building the models to understand why the most important variables in the models were effective in predicting the target variable. Pre-modeling, there may be dozens, hundreds, or even thousands of candidate input variables. Post-modeling, even if all the candidate

input variables were used in the model, only a small subset are typically good predictors, reducing the number of variables the analysts needs to visualize to understand the most important patterns in the data.

Histograms

Histograms operate on a single variable and show an estimate of the shape of the distribution by creating bins of the data and counting how many records of the variable fall within the bin boundaries. The x-axis typically contains equal-width bins of the variable, and the y-axis contains the count or percentage of count of records in the bins. The analyst can usually specify the number of bins in the software, though the default tends to be 10.

The first reason to examine a histogram is to examine the shape. Figure 3-6 shows a standard histogram with a shape that appears to be normal or close to it. Figure 3-7, on the other hand, shows a histogram that is bi-modal; there are two peaks in the distribution. The mean is actually at the trough between the peaks and therefore is not a typical value at all. Moreover, the standard deviation in the summary statistics will be much larger than the spread in either of the two subdistributions of the data.

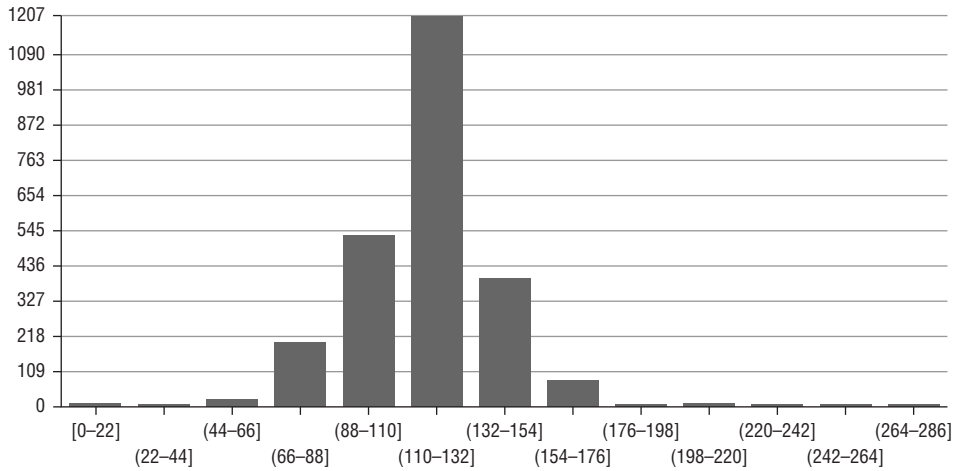


Figure 3-6: Standard histogram

The third histogram, Figure 3-8, shows a spike in the data that dwarfs any other bin. This variable is clearly not normal or uniform, and is a good candidate for a variable transformation if you are building models using a numeric algorithm such as regression or K-Means clustering.

Figure 3.7: Bi-modal distribution

Histograms can be customized in many software packages, including these three, which are the most common:

- **Change the number of bins.** This can be particularly helpful in identifying data shape that can be distorted by using a particular number of bins or when data clumps near bin boundaries. Some software allows you to click through the number of bins as a part of the graphic, making this very easy to do and visually effective. Other software requires you to exit the histogram and rebuild the graphic completely, which makes changing the number of bins tedious at best.
- **Change the y-axis scale.** Another common modification is to make the counts on the y-axis of the histogram log scale rather than linear scale when a few bins have large bin counts—spikes in the histogram—that overwhelm the size of counts in all other bins. Some software also includes options to show the mean value of a second variable on the y-axes, such as the target variable.
- **Overlay a second categorical variable.** Some software allows the histogram to be color coded by a categorical variable, such as the target variable. This is helpful to determine if there is an obvious, visible relationship between the input variable and the target, as shown in Figure 3-9. However, when bin sizes vary considerably, judging the relative proportion of the values of the target variable represented in each bar can be difficult. In these cases, if the software also allows you to change the scale of each bar so that each is full height, it is easy to see the relative proportions of the target in each bin. This is essentially a visual crosstab.

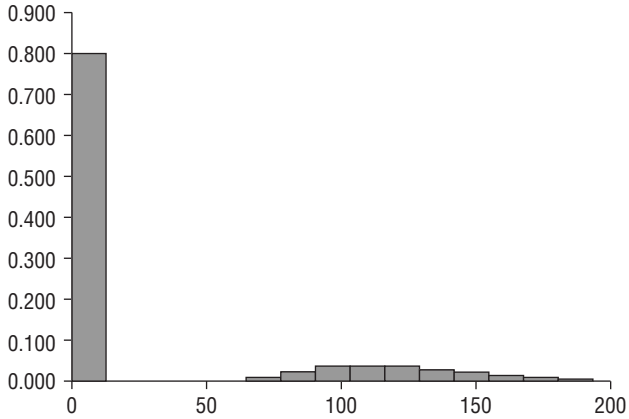


Figure 3-8: Spike in distribution

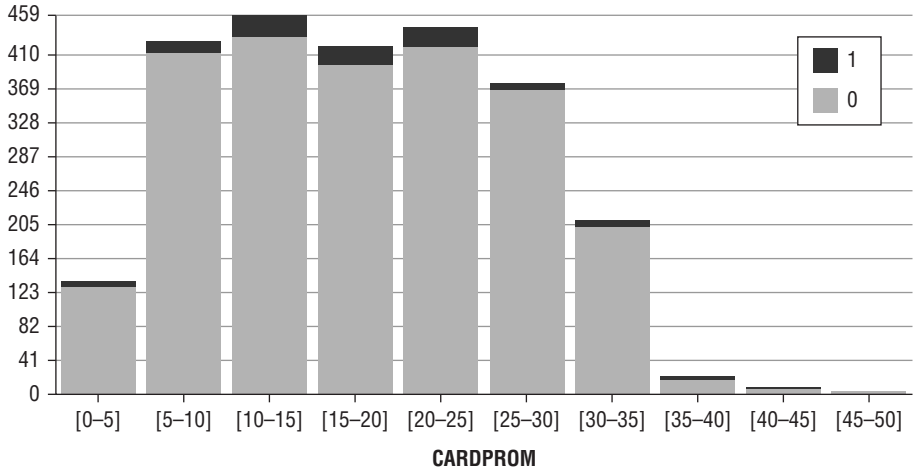


Figure 3-9: Histogram overlaid with Target variable

The box plot is a graphical representation of the quartile statistics of a variable and is an excellent method to gain quick insight into the characteristics of numeric data. In the box plot, the "box" is the IQR; the median is typically represented by a line cutting the IQR box in half. Some software also shows *whiskers* by drawing lines between the IQR box and some other value in the data. Most commonly, this other value is an upper quartile outlier boundary (+1.5 * IQR from the 3rd quartile) and the lower quartile outlier boundary (–1.5*IQR from

the 1st quartile), though it can also represent the minimum and maximum of the range in the data or even a small and large percentile in the data (1 percent and 99 percent for example).

Figure 3-10 shows a box plot for NUMPROM with the whiskers representing the minimum and maximum of the distribution. However, this form can obscure the main density of the data if the maximum is an extreme outlier. This can be seen with the box plot of LASTGIFT in Figure 3-11. A different depiction of the quartiles in LASTGIFT is shown in Figure 3-12, though this time the whiskers show outliers in the data, and all values exceeding the extent of the whiskers are identified as outliers. This plot has had all values greater than 200 removed from the plot to make the IQR box easier to see.

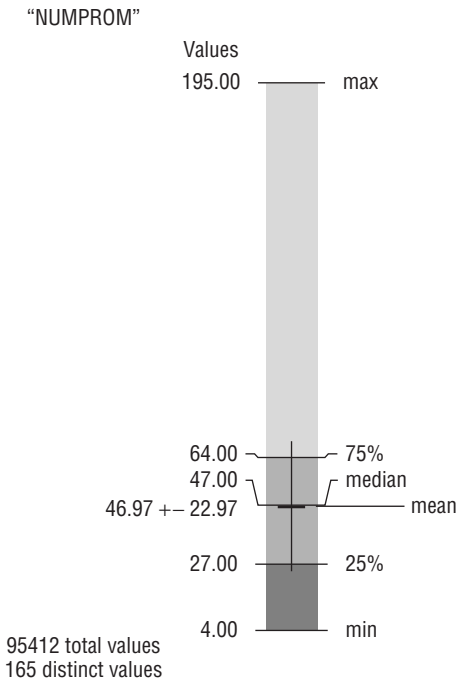


Figure 3-10: Box plot of NUMPROM

Box plots are sometimes represented horizontally in predictive analytics and statistics software. Unfortunately, not all predictive modeling software contains box plots, although it is becoming much more commonplace.

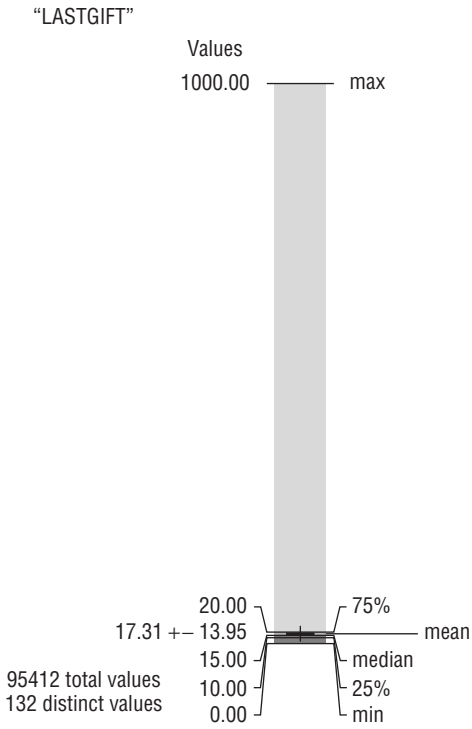


Figure 3-11: Box plot with outliers squashing IQR

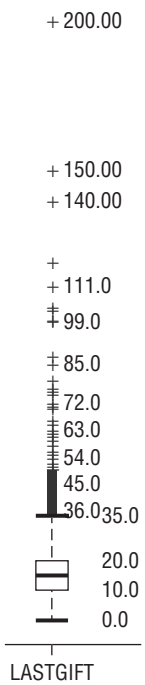


Figure 3-12: Box and whiskers plot showing outliers

Multiple Variable Summaries

Examining single variables provides valuable information to assess the quality and value of the variables for use in predictive modeling. However, the value of a variable in predicting a target variable is sometimes hidden and can only be revealed through interactions with other variables.

Hidden Value in Variable Interactions: Simpson's Paradox

One of the most intriguing examples demonstrating the value of variable interactions can be seen in a phenomenon known as *Simpson's Paradox* where a trend is seen in individual variables but is reversed when variables are combined.

Consider a patient who needs to go to a hospital for surgery. The patient considers two hospitals, Hospital A and Hospital B, by examining their mortality rates. The data used to make the decision is shown in Table 3-11.

Table 3-11: Simpson's Paradox Example, Aggregate Table

RESULT	HOSPITAL A	HOSPITAL B	TOTAL
Died	63	16	79
Survived	2037	784	2821
Total	2100	800	2900
Mortality Rate	3.0%	2.0%	2.7%

Based on the information, you would obviously choose Hospital B because its mortality rate is 33 percent lower than Hospital A.

But what if you know an additional piece of information before selecting the hospital, namely the condition of the patient prior to surgery? Table 3-12 shows these values.

Table 3-12: Simpson's Paradox Example, The Interactions

	GOOD CONDITION			BAD CONDITION		
CONDITION	HOSPITAL A	HOSPITAL B	TOTAL	HOSPITAL A	HOSPITAL B	TOTAL
Died	6	8	14	57	8	65
Survived	594	592	1186	1443	192	1635
Total	600	600	1200	1500	200	1700
Mortality Rate	1%	1.3%	1.6%	3.8%	4.0%	3.8%

Now Hospital A has the lower mortality rate for both subgroups: those in good condition *and* those in bad condition. But how can this be when Hospital A was higher overall?

The key is in the counts: Hospital A not only treats more patients (2100 vs. 800, from Table 3-10), but it treats a higher percentage of patients in bad condition, whereas Hospital B actually treats fewer patients in bad condition than in good condition. Since the mortality rate for patients arriving in bad condition is three to four times worse, Hospital A's overall mortality rate is biased and appears worse. In other words, Hospital A appears to have a worse mortality rate because it treats the most difficult patients.

Simpson's Paradox isn't just an academic oddity; it happens in many real data sets. One of the most famous is the case against the University of California at Berkeley, which was accused of gender bias in admissions to graduate schools. In 1973, 44 percent of male applicants were accepted, whereas only 35 percent of female applicants were accepted, a statistically significant difference very unlikely to occur by chance. It seemed like a clear case of discrimination.

However after separating the admissions rates by department, it turned out that women were admitted at a higher rate than men in most departments. How can this be when the overall rates were so overwhelmingly in favor of men? Women applied to departments that were more competitive and therefore accepted lower percentages of applicants, suppressing their overall acceptance rates, whereas men tended to apply to departments with higher overall acceptance rates. It was only after considering the interactions that the trend appeared.

The Combinatorial Explosion of Interactions

Interactions therefore sometimes provide critical information in assessing variables and their value for predictive modeling. However, assessing all possible combinations can be problematic because of the combinatorial explosion. Table 3-13 shows how many combinations must be considered if all possible interactions are considered.

Table 3-13: Number of Interactions as Number of Variables Increases

NUMBER OF VARIABLES	NUMBER OF POSSIBLE TWO-WAY INTERACTIONS	NUMBER OF POSSIBLE THREE-WAY INTERACTIONS	NUMBER OF POSSIBLE FOUR-WAY INTERACTIONS
5	10	10	5
10	45	120	210
50	1,225	19,600	230,300
100	4,950	161,700	3,921,225
500	124,750	20,708,500	2,573,031,125
1000	499,500	166,167,000	41,417,124,750

For only five variables, there isn't a problem in considering all possible combinations. However, if you have even 50 input variables, not unusual in most predictive modeling data sets, you would have to consider 1,225 two-way, 19,600 three-way, and 230,300 four-way interactions. With 1000 variables, there are over 40 billion four-way interactions.

As a result, any analysis of interactions that is not automatic must focus on particular interactions expected to be interesting. This will be the case in considering crosstabs of variables and visualization of pairs of variables in scatterplots.

Correlations

Correlations measure the numerical relationship of one variable to another, a very useful way to identify variables in a modeling data set that are related to each other. This doesn't mean one variable's meaning is related to another's. As the well-known phrase goes, "Correlation does not imply causation." Or, in other words, just because two variables are numerically related to one another doesn't mean they have any informational relationship.

Spurious Correlations

Spurious correlations are relationships that appear to be related but in fact have no direct relationship whatsoever. For example, when the AFC wins the Super Bowl, 80 percent of the time the next year will be a bull market. Or butter production in Bangladesh produces a statistically significant correlation to the S&P 500 index.

One such rule is the so-called Redskin Rule. The rule is defined in this way. Examine the outcome of the last Washington Redskins home football game prior to a U.S. Presidential Election, and between 1940 and 2000, when the Redskins won that game, the incumbent party won the electoral vote for the White House. On the other hand, when the Redskins lost, the incumbent party lost the election. Interesting, it worked perfectly for 17 consecutive elections (1940 –2000). If you were to flip a fair coin (a random event with odds 50/50), the likelihood of flipping 17 consecutive heads is 1 in 131,072, very unlikely.

However the problem with spurious correlations is that we aren't asking the question, "How associated is the Redskins winning a home game prior to the presidential election to the election outcome" any more than we are asking the same question of any other NFL team. But instead, the pattern of Redskin home wins is found *after the fact* to be correlated as one of thousands of possible patterns to examine. If you examined enough other NFL outcomes, such as examining the relationship between Presidential elections and all teams winning their

third game of the year, or fourth game of the year, or first pre-season game of the year, and so on until you've examined over 130,000 possible patterns, one of them is likely to be associated.

This problem of over-searching to find patterns that match your target is a real problem in predictive analytics and will be addressed during the discussion on sampling. For now, you can merely chuckle at the myriad of spurious correlations you find in data.

However, some spurious correlations are related, albeit not directly. For example, consider a town where it is noticed that as ice cream cone sales increase, swimsuit sales also increase. The ice cream sales don't cause the increased sales in swimsuits, but they are related by an unobserved variable: the outdoor temperature. The correlation is spurious but not a random relationship in this case.

Back to Correlations

Nevertheless, correlations are very helpful ways to assess numeric variables in a pairwise analysis. Finding pairs of variables with high correlation indicates there is redundancy in the data that should be considered for two main reasons. First, redundant variables don't provide new, additional information to help with predicting a target variable. They therefore waste the time spent building models. Second, some algorithms can be harmed numerically by including variables that are very highly correlated with one another.

Most predictive analytics software provides a way to produce a correlation matrix, one correlation for each pair of variables. If there are N numeric variables in the data, you can create a correlation matrix that is $N \times N$. This shows an advantage of the correlation matrix as a method of Data Understanding: All numeric variables can be assessed in a single report.

Table 3-14 contains one such correlation matrix, from the same variables included in Tables 3-2, 3-3, and 3-5 except for NUMPRM12. The values above the diagonal of the matrix are equivalent to those below the diagonal: The correlation of CARDPROM to NUMPROM is identical to the correlation of NUMPROM to CARDPROM. The values of correlation range from –1 to +1, where +1 indicates perfect positive correlation and –1 indicates perfect negative correlation. Perfect correlation doesn't necessarily mean the values are identical, only that they all fall along a line; the relationship between values of the two variables have constant positive slope. However, it is obvious that every variable is perfectly correlated with itself, which is why the diagonal in the correlation matrix must all have the value 1.

Perfect negative correlation indicates the slope is negative and constant. A correlation value of 0 indicates there is no relationship between the two variables; they appear to be random with respect to one another.

Table 3-14: Correlations for Six Variables from the KDD Cup 98 Data

CORRELATIONS	CARDPROM	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT
CARDPROM	1.000	0.949	0.550	0.023	–0.059
NUMPROM	0.949	1.000	0.624	0.066	–0.024
RAMNTALL	0.550	0.624	1.000	0.557	0.324
MAXRAMNT	0.023	0.066	0.557	1.000	0.563
LASTGIFT	–0.059	–0.024	0.324	0.563	1.000

Another way to understand correlation is that the higher the magnitude of correlation, the more you can determine the value of one variable once you know the value of the other. If two variables are highly correlated and one value is at the upper end of its range of values, the value of the second variable in the same record must have its value at the upper end of its range of values as well. If the values are highly correlated, greater than 0.95 as a rule of thumb, they are essentially identical, redundant variables.

Note, too, that in Table 3-14, NUMPROM and CARDPROM have very high correlation (0.949), although CARDPROM is nearly unrelated numerically to LASTGIFT as measured by their correlation (–0.059).

You must beware of how correlations are interpreted, however, for several reasons:Correlations are linear measures. Just because two variables are not linearly related to one another doesn't mean that they are not related at all to one another. The correlation between x and x^2 is 0 because when x is positive, x^2 is positive, but when x is negative, x^2 is still positive with the same magnitude as when x was positive. Correlations are affected severely by outliers and skewness. Even a few outlier value pairs can change the correlation between them from something small, like 0.2, to something large, like 0.9. Visualization is a good idea to verify that high correlation is real.

Crosstabs

Crosstabs, short for cross-tabulations, are counts of the intersection between two variables, typically categorical variables. As was already noted in the combinatorial explosions shown in Table 3-13, you can only specify so many of these crosstabs to compute, so usually only those interactions suspected to be interesting are considered during Data Understanding.

Consider the crosstab of RFA_2F and RFA_2A in Table 3-15. As a reminder, RFA_2F indicates how many gifts were given in the past time period of interest (1, 2, 3, or 4 or more), and RFA_2A indicates the dollar amount given during this time period of interest, D being the lowest and G the largest.

Table 3-15: Crosstab of RFA_2F vs. RFA_2A

RFA_2A / RFA_2F	D	E	F	G	TOTAL	PERCENT OF TOTAL
1	0	4,911	30,380	12,384	47,675	50.0%
2	0	4,989	10,961	4,595	20,545	21.5%
3	2,498	7,767	3,523	1,503	15,291	16.0%
4	4,914	3,972	2,100	915	11,901	12.5%
Total	7,412	21,639	46,964	19,397	95,412	100.0%
Percent of Total	7.8%	22.7%	49.2%	20.3%	100.0%	

Interestingly, the two cells in the upper left of the crosstab are 0: There are no donors who gave small gifts (RFA_2A=D) 1 or 2 times during the past year (RFA_2F = 1 or 2).

Is this unexpected? You can compute the number of gift records expected to be given if the interactions were truly random by multiplying the Percent of Total values for RFA_2A=D and RFA_2F = 1: 50.0% * 7.8% = 3.9%. You would expect 3.9 percent of the donors to fall in this cross-tab bin, which, for 94,412 donors is 3,721. However, none fall there. Could this be by chance? Very unlikely.

Undoubtedly there is some other reason for these zeros. These cells describe donors who give infrequently and in low amounts. Apparently, when donors give less often, they give in larger amounts, or if they give in smaller amounts, they give more frequently. But, apparently for this non-profit, donors never give both infrequently and in low amounts together.

Data Visualization, Two or Higher Dimensions

Visualizing data in two or more dimensions provides additional insights into data, but because of the combinatorial explosion shown in Table 3-13, the number of plots can become unmanageable.

In this section, all plots will use the Nasadata data set because the value of higher-dimensional visualization is seen more readily.

Scatterplots

Scatterplots are the most commonly used two-dimensional visualization method. They merely plot the location of data points for two variables. A sample scatterplot between Band 1 and Band 2 is shown in Figure 3-13. Note that there is

a strong relationship between the two variables—their correlation coefficient is +0.85. Scatterplots are effective ways to show visually how correlated pairs of variables are: The more cigar-shaped the distribution and the narrower that cigar is, the stronger the correlation.

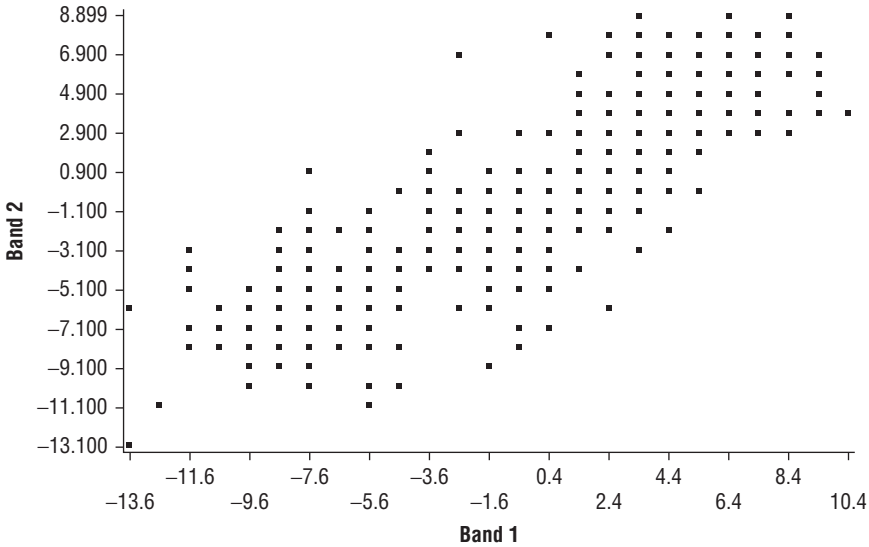


Figure 3-13: Scatterplot of Band 2 vs. Band 1

For highly correlated NUMPROM and CARDPROM (0.949), the scatterplot looks like Figure 3-14.

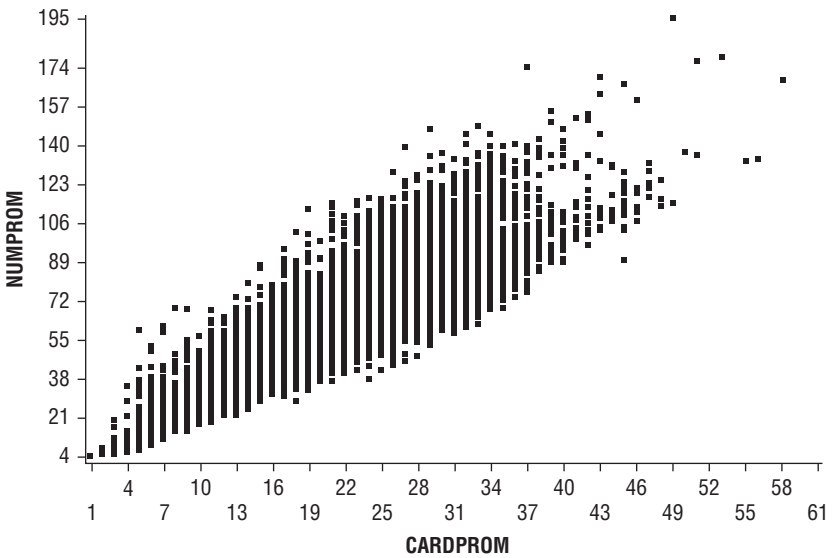


Figure 3-14: NUMPROM vs. CARDPROM scatterplot

There are many practical limitations to the effective use of scatterplots. Some of the problems include the following:

- **Too many data points:** If too many data points are plotted, you won't be able to even see them all on the computer screen or a piece of paper, even if each data point is represented as a single pixel. Moreover, when the points lie on top of each other or too near each other for distinct points to be seen, the plot contains indistinct blobs and ceases to be informative.

  Most predictive modeling software limits the number of data points that can be used in a scatterplot to 2,000 or so. One way to present the data points more effectively is, rather than using small squares or circles to represent data points, use open circles. When data points differ slightly, the rings will still show themselves as distinct open circles. However, with too many data points, this is still not enough.

  Yet another solution some software vendors have for this problem is that rather than plotting individual data points, plot regions in the scatterplot space and color code those regions with the relative density of counts within the region. It is, in effect, a two dimensional histogram of the data or a heat map plot.

- **Outliers or severe skew pushed data into a small area of the plot:** When there is severe skew of severe outliers in the data, the data is sparse in the tails of the distribution. However, the plot is presented using a linear scale, which means that much, if not most, of the number line has little data in it; most of the data ends up in a small area of the plot so that there is little information that can be gleaned from the scatterplot itself. Figure 3-15 shows highly skewed data plotted on a linear scale.

  One solution to the problem is to scale the data before plotting it so that there aren't big outliers or large sparse ranges of input data. A second solution is to change the scale of the plot axes. For positively skewed data, log scales are effective in changing the data so that the analyst can see the data resolution and patterns better. However, the analyst must keep in mind that the data is no longer in its natural units but is in log units.

Anscombe's Quartet

Statistics can be deceiving. If data is not normally distributed and statistics that assume normal distributions are used, the statistics may not represent the summary you expect and can therefore be misleading. One famous example of this phenomenon is captured in four scatterplots known as Anscombe's Quartet, created in 1973 by the statistician Francis Anscombe. The data plotted in the quartet of scatterplots is shown in Table 3-16.

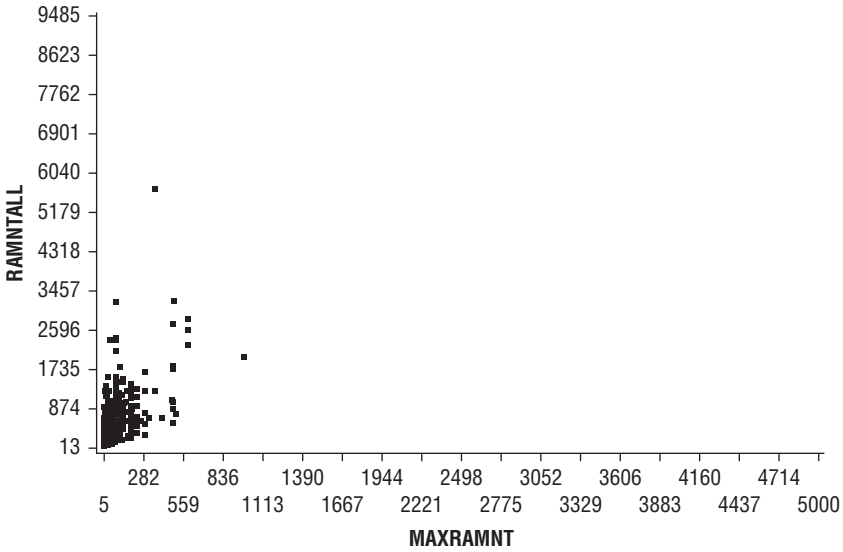


Figure 3-15: Highly skewed data in scatterplot

Table 3-16: Anscombe's Quartet Data

SET 1		SET 2		SET 3		SET 4	
X	Y	X	Y	X	Y	X	Y
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.1	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.1	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

Interestingly, each of the four pairs of data points, has the properties shown in Table 3-17.

Table 3-17: Anscombe's Quartet Statistical Measures

MEASURE	VALUE
Mean X	9.0
Variance X	11.0
Mean Y	7.50
Variance Y	4.12 to 4.13
Correlation X vs. Y	0.816 to 0.817
Regression Equation	Y = 3.0 + 0.500*X
R^2 for fit	0.666 to 0.667

After seeing these measures and values for all four of the data sets, especially the correlation, regression equation, and *r*-squared values, an image usually emerges in the minds of analysts. However, a different story appears after seeing the four scatterplots in Figure 3-16.

The first plot is the one that probably was in the minds of most analysts. However, the remaining three plots fit the data just as well. The upper-right plot (data set 2) shows a quadratic structure to the data. If you built a regression model or trend line that included both linear and quadratic terms for x, the fit would be perfect. In the lower-left plot, data set 3 shows a perfect linear relationship except for one outlier. If this outlier, the third data point with coordinates (13,12.74), was removed from the data, the trend line would be a nearly perfect linear fit, with the equation:

$$y = 4.0056 + 0.3454x,\ R^2 = 0.99999$$

so the removal of this one data point changes the slope from 0.5 in Table 3-2 to 0.3454 without the outlier.

The final plot (data set 4) is a pathological data set where every point has the same x value except one, creating an extreme outlier. Extreme outliers can become leverage points, meaning that because they are so far from the general trend of the data, the regression line will pass through or near the data point. Figure 3-17 shows the original value of Y, 12.5, three other values for Y, 2.5, 6.5, and–-4, and the linear fit of the data that results from these values. The original value of Y just so happens to produce the summary statistics that match the summary statistics of the other three data sets.

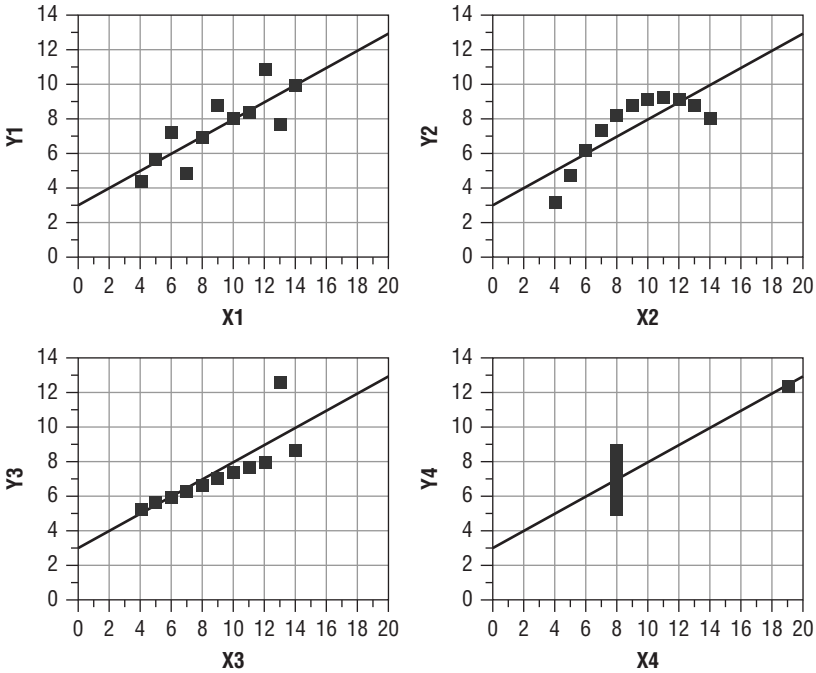


Figure 3-16: Anscombe's Quartet scatterplots

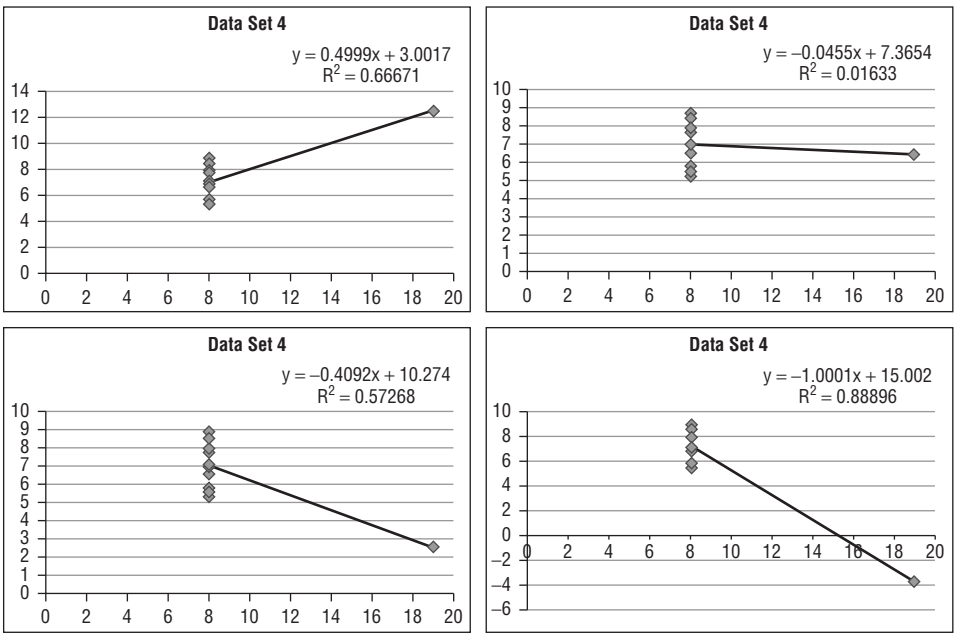


Figure 3-17: Data Set 4, four different leverage points

The conclusion therefore is that summary statistics are useful but don't always tell the whole story.

Scatterplot Matrices

Just like a correlation matrix shows all pairwise correlations in the data, the scatterplot matrix shows all pairwise scatterplots for variables included in the analysis. They provide a good way to show several relationships between pairs of variables in one plot.

However, scatterplots take significant amounts of space to display, and significant resources to draw, so typically, the number of variables included in the scatterplot is kept to less than 10. In the scatterplot matrix of Figure 3-18, only six variables are shown, the first of which is the categorical target variable. Note that you can see immediately which pairs of variables are highly correlated: Band 1 and Band 3, Band 3 and Band 6, and Band 9 and Band 11 (negatively).

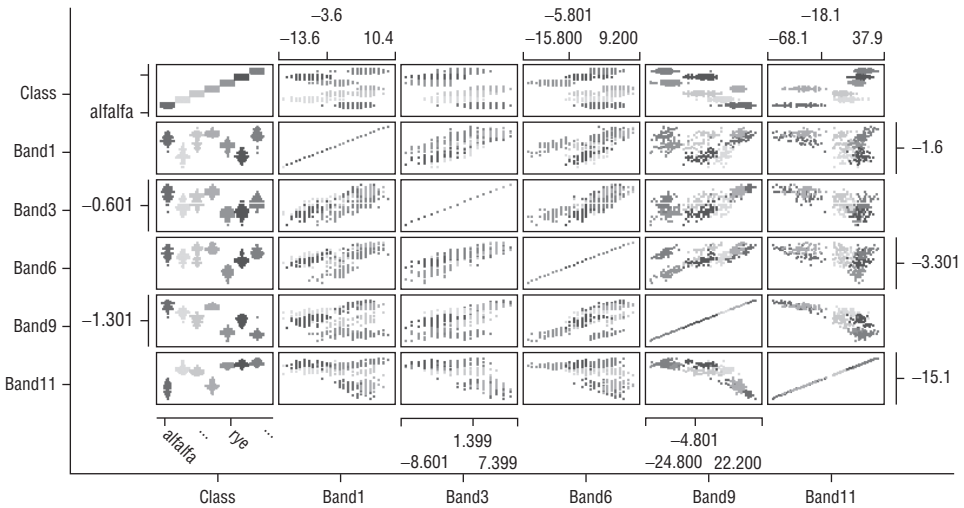


Figure 3-18: Scatterplot matrix

Plotting many variables in one plot is usually a problem. One solution to that is the idea of creating plots of *parallel coordinates*. In this plot, shown in Figure 3-19, each variable is represented in a column with its range independently scaled to its minimum to maximum range. The points on the vertical axes are the values of that variable. The lines connect data points for a single record, so in principle, you could trace an entire record from left to right in a parallel coordinate plot.

The advantage of the plot is that all, or at least many, variables can be shown in one diagram. However, there are many issues that make parallel coordinates cumbersome to use practically: When there are large numbers of records, the

lines blur together making trends exceedingly difficult to see. Parallel coordinates work best when the number of records is low—hundreds, not thousands or more. Some implementations overcome this problem by plotting density of records rather than individual records so that you can see darker colors for more common patterns and lighter for less common patterns. Patterns may be clear between neighboring variables, but are difficult to see when variables are separated. Even with fewer records, patterns are difficult to see because the order of the variables matters in the interpretation. One solution is to color code lines by a key categorical variable, such as the target, making the pattern more evident throughout the range. In Figure 3-19, the Class values wheat, soy, and rye are all in the upper range of Band11, whereas clover and alfalfa are at the lower range of Band11. However, the plot shows that for Band9, soy is larger than wheat and rye. Too many variables make the patterns difficult to see. You must limit the number of variables to the number that can fit legibly on a single page on the screen, PowerPoint presentation, or on a sheet of paper.

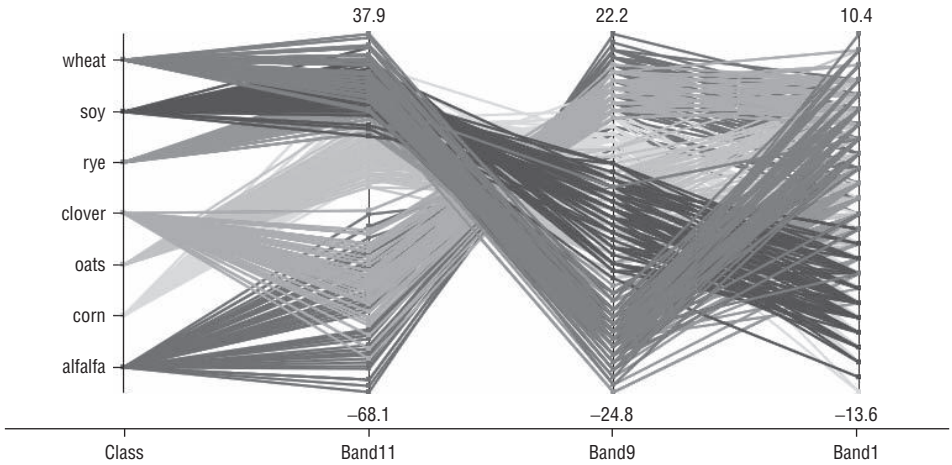


Figure 3-19: Parallel coordinates

Overlaying the Target Variable in Summary

One simple but powerful extension to any data visualization method is a simple overlay of an addition variable, usually a categorical variable or the target variable. Consider a binary target variable as the overlay. For scatterplots and scatterplot matrices, the data points are color coded by the value of the target variable showing regions in the scatterplots that have stronger or weaker relationships to the target variable. In a histogram, the counts in the bar attributed to each value of the target variable are represented by a color.

Adding this additional dimension provides a first look-ahead to the predictive power of the input variables in the chart. They can also provide critical information about target-variable leakage. If the relationship between an input variable and the target is too strong, often the explanation is that the target variable information is somehow held within the input variable. For example, if you built a histogram of area codes overlaid by a 1/0 target variable, and if you found that for all area codes except for "000" the target variable was always "1," whereas for "000" the target variable was always "0," you would not conclude that an area code is a perfect predictor, but rather that the area code is only known or coded *after* the target variable value is known. For the area code, this could mean that telephone numbers are only known *after* the donor has responded and therefore it is not a good predictor of response.

Figure 3-20 shows a histogram of Band 1 overlaid with the target variable: the seven crop types. Note that you can see that wheat, clover, oats, and alfalfa are present only for high values of Band 1, whereas corn and soy are present only for low values of Band 1.

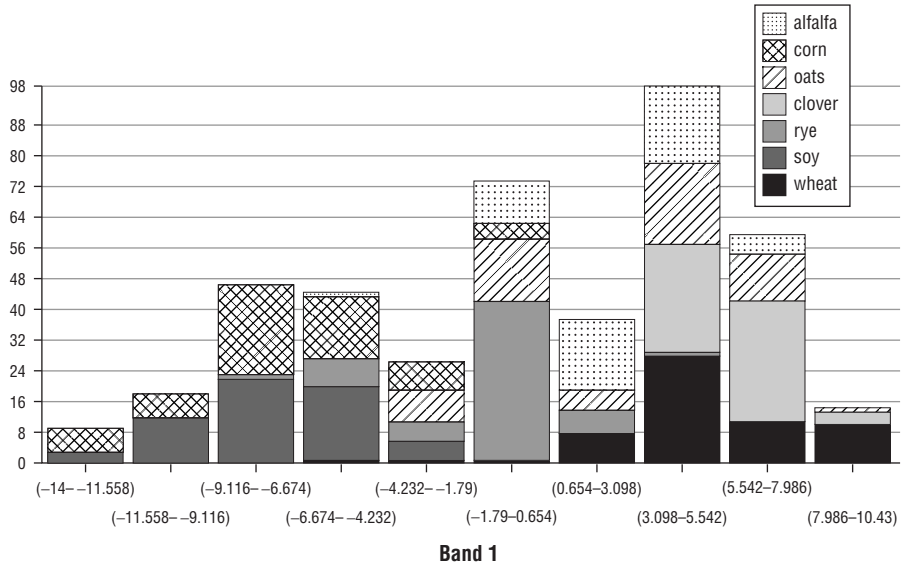


Figure 3-20: Histogram color coded with target

With box plots, you can split the box plot into one for each value of the target variable, as shown in Figures 3-21 and 3-22. From these plots, you can see even more clearly the trends identified in the histogram.

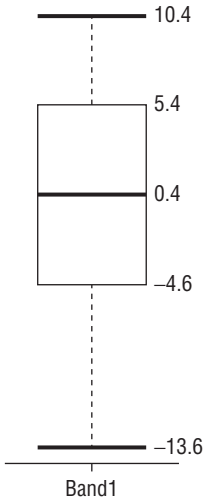


Figure 3-21: Box plot for Band 1

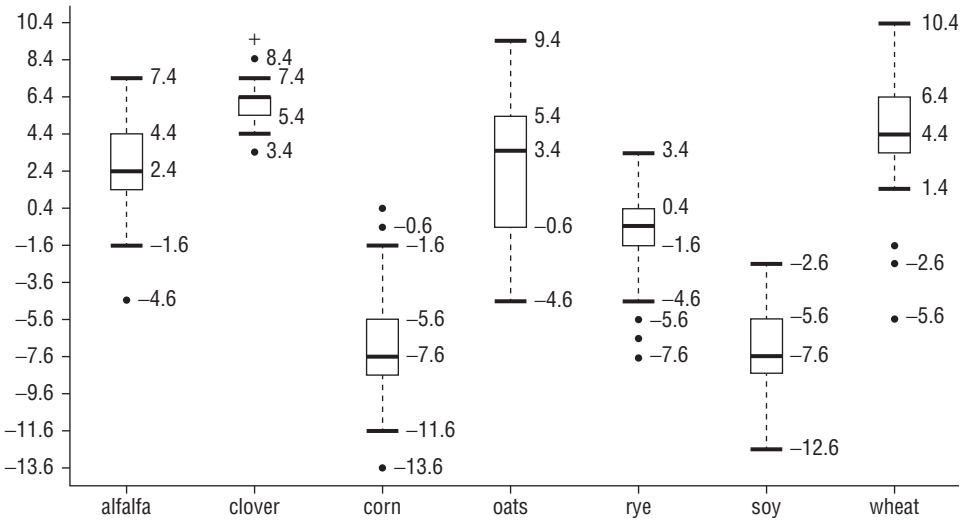


Figure 3-22: Box plot for Band 1 split by a target variable

Scatterplots in More Than Two Dimensions

Scatterplots can be enhanced to represent more than two dimensions in two ways. The first method is through projection; higher dimensional space can

be projected onto two dimensions for visualization. The projection space must always be two dimensions because the medium onto which the data is projected is two dimensional, whether it be a computer screen, monitor, or sheet of paper.

The simplest of the higher dimensional projections is a scatterplot in three dimensions. However, any snapshot summary of the three-dimensional plot is just one of the possible 2-D projections of the three-dimensional data. The value of 3-D plots comes from rotating the plot and observing the changes as you rotate it, which isn't easy to replicate on paper. As the dimensions increase to 4, 5, and more, the projection is even more troublesome because the combinatorial explosion makes picking the right projection ever more difficult.

Some algorithms can help select the project so that the interesting information in the higher dimensional projection is retained. Kohonen Self-Organizing Maps, discussed in Chapter 6 as an unsupervised learning technique, is one algorithm often used to project high-dimensional data onto two dimensions. Others algorithms belonging to a class of *multidimensional scaling* (MDS) algorithms, such as the Sammon Projection, try to retain the relative distances of data points in the 2-D projection as was found in the higher dimensional space. Of course, there are limitations; it is rare that analysts are able to effectively project more than 20 or 30 dimensions down to 2 without the help of algorithms to find interesting projections. Few predictive modeling software packages implement MDS algorithms and the grand tour because of the computational problems with them as the number of records and fields grows.

A simpler way to visualize higher-dimensional data is to use other means to modify the individual data points to provide the information found in the higher dimensions. Three of these are color, size, and shape. If each data point represents two dimensions spatially (the x and y axes), and also is represented by these three additional modifiers, five dimensions can now be visualized. Figure 3-23 shows the same two spatial dimensions as was found in Figure 3-13, but this time color coded by Class (the target variable), shape also coded by Class, and icon size coded by variable Band 9.

Note that the size shows the square shapes representing alfalfa have larger values of Band 9 (larger icon size), large values of Band 1, but small values of Band 12. The same is true for the reverse triangles representing clover.

This plot shows revealing patterns in the five dimensions plotted (only four unique dimensions), but finding these relationships can be quite difficult and take exploration on the part of the analyst to use these dimensions effectively. Sometimes, these dimensions are only found after modeling has revealed which variables are the most effective in predicting the target variable.

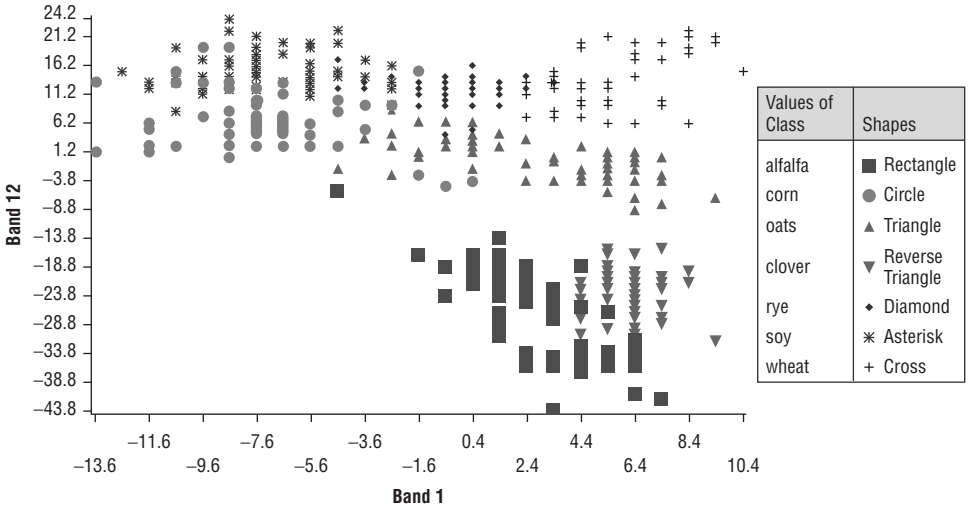


Figure 3-23: Scatterplot with multidimensional overlay

The Value of Statistical Significance

The idea of the significance of a measured effect is not commonly considered in predictive modeling software except for those tools that have a strong connection with the field of Statistics. Describing and explaining these statistics is outside the scope of this book; any statistics textbook will describe them in detail. However, on the most basic level, understanding these statistics well enough to interpret them can provide a modeler with additional insight. A short list of the significance measures described so far in this chapter is shown in Table 3-18.

The sample size, N, has a large effect on the significance of the statistical measures. Significance was devised primarily for small data (small N) to aid in understanding the uncertainty associated with interpreting the statistics. For data sets with tens of thousands of records or more, even when there is statistical significance, there may not be operational differences that matter. For example, in Table 3-2, the skewness value for CARDPROM is statistically significant, but it does not indicate a problem with skew, only that there were so many records in the data that even a skew of 0.15 is statistically significant.

Table 3-18: Measures of Significance in Data Understanding

BASE STATISTIC	SIGNIFICANCE MEASURE
Mean	Standard Error of Mean
Skewness	Standard Error of Skew
Kurtosis	Standard Error of Kurtosis
Crosstabs	Chi-square Statistic and p value
Correlations	Pearson correlation coefficient and p value
Binomial Test	Error at 95% Confidence Level
Anova, Difference of Means	F-statistic

Pulling It All Together into a Data Audit

Putting the Data Understanding steps together—computing summary statistics, examining trends in the data, identifying problems in the data, and visualizing the data—forms what you might call a *data audit*. You should take note of the following items for correction during Data Preparation.

A list of data audit items might include the following:

- How many missing values are there? Do any variables have mostly or all missing values?
- Are there strange minimum or maximum values?
- Are there strange mean values or large differences between mean and median?
- Is there large skew or excess kurtosis? (This only matters for algorithms that assume normal distributions in the data.)
- Are there gaps in the distributions, such as bi-modal or multi-modal distributions?
- Are there any values in the categorical variables that don't match the dictionary of valid values?
- Are there any high-cardinality categorical variables?
- Are there any categorical variables with large percentages of records having a single value?

- Are there any unusually strong relationships with the target variable, possibly indicating leakage of the target into a candidate input variable?
- Are any variables highly correlated with each other, possibly indicating redundant variables?
- Are there any crosstabs that show strong relationships between categorical variables, possibly indicating redundant variables?

Summary

Data Understanding provides the analytic foundation for a project by summarizing and identifying potential problems in the data. Summaries of the data can confirm the data is distributed as expected, or reveal unexpected irregularities that need to be addressed during Data Preparation. Problems in the data, such as missing values, outliers, spikes, and high-cardinality, should be identified and quantified so they can be fixed during Data Preparation.

Data visualization provides the analyst insights that are difficult or even impossible to discover through summary statistics. It must be done with care prior to modeling when the number of candidate modeling variables is high. Visualization should be revisited after modeling is finished to show graphically how the most important variables relate to the target variable.

This stage in the predictive modeling process cannot be rushed; Problems missed during Data Understanding will come back to haunt the analyst during Modeling.

CHAPTER

4

Data Preparation

Data Preparation is the third stage of the predictive modeling process, intended to convert data identified for modeling into a form that is better for the predictive modeling algorithms. Each data set can provide different challenges to data preparation, especially with data cleansing, rendering recipes for data preparation exceedingly difficult to create; deviations from any recipe will certainly be the norm. The approach taken in this chapter is to provide a set of techniques one can use to prepare data and the principles to consider when deciding which to use.

The key steps in data preparation related to the columns in the data are variable cleaning, variable selection, and feature creation. Data preparation steps related to the rows in the data are record selection, sampling, and feature creation (again).

Most practitioners describe the Data Preparation stage of predictive modeling as the most time-intensive step by far, with estimates of the time taken during this stage ranging between 60 and 90 percent. Conversations with recruiters who know and understand predictive modeling reveal a similar story, even to the point where if the candidate does not give this answer, the recruiter knows that the candidate has not been actively engaged in predictive modeling. This also differentiates "real-world" modeling data from the data used in many competitions and sample data sets made available in software: These data sets are often cleaned extensively first because the time and domain expertise needed to clean data is beyond what can be accomplished in a short period of time.

A new predictive modeling hire, upon receiving the data for his first modeling project, was quoted as complaining, "I can't use this data; it's all messed up!" Precisely. Data almost always has problems, and fixing them is what a predictive modeler has to do.

One reason that data cleaning in particular takes so long is that the predictive modeler is often the first person who has *ever* examined the data in such detail. Even if the data has been scrubbed and cleaned from a database storage perspective, it may never have been examined from a predictive modeling perspective; database cleanliness is not the same as predictive modeling cleanliness. Some values may be "clean" in a technical sense because the values are populated and defined. However, they may not communicate the information effectively to modeling algorithms in their original form. Additionally, some values that to a human are completely understandable—a value of –1 in a profit variable that should always be positive is known to mean the company report hasn't come out yet—are confounding and contradictory, and could be damaging to a predictive model; predictive modeling algorithms know only what is explicitly contained in the data.

The 60–90 percent guideline is a good rule of thumb for planning the first predictive modeling project using data that has never been used for modeling before. However, once the approaches for cleaning and transforming data have been defined, this number will drop to less than 10 percent of the overall time because the most difficult work has already been done.

Variable Cleaning

Variable cleaning refers to fixing problems with values of variables themselves, including incorrect or miscoded values, outliers, and missing values. Fixing these problems can be critical to building good predictive models, and mistakes in variable cleaning can destroy predictive power in the variables that were modified.

The variables requiring cleaning should have been identified already during Data Understanding. This section describes the most common solutions to those data problems.

Incorrect Values

Incorrect values are problematic because predictive modeling algorithms assume that every value in each column is completely correct. If any values are coded incorrectly, the only mechanism the algorithms have to overcome these errors is to overwhelm the errors with correctly coded values, thus making the incorrect values insignificant.

How do you find incorrect values and fix them? For categorical variables, the first step is to examine the frequency counts as described in Chapter 3 and identify values of a variable that are unusual, occurring very infrequently. If there are not too many values of a variable, every value can be examined and verified. However, for some values that are incorrect, there is no easy way to determine what the actual value is. The analysis may require domain experts to interpret the values so the analyst better understands the intent of the values saved in the data, or database experts to uncover how and why the incorrect values made their way into the data in the first place.

If the values cannot be interpreted, they should be treated as missing if they are infrequent, but if they occur in significant numbers, they can be left as their incorrect value for the purposes of modeling, only to be interpreted later.

For continuous values, incorrectly coded values are most often detected as outliers or unusual values, or as spikes in the distribution, where a single repeated value occurs far more often than you would otherwise expect. If the incorrectly coded values occur so infrequently that they cannot be detected easily, they will most likely be irrelevant to the modeling process.

Consistency in Data Formats

A second problem to be addressed with variable cleaning is inconsistency in data. The format of the variable values within a single column must be consistent throughout the column. Most often, mismatches in variable value types occur when data from multiple sources are combined into a single table. For example, if the purchase date in a product table comes from several sources—HTTP, mobile, and .mobi, for instance—the date format must be consistent for predictive modeling software to handle the dates properly. You cannot have some dates with the format mm/dd/yyyy and others with the format mm/dd/yy in the same column. Another common discrepancy in data types occurs when a categorical variable (like ZIP code) is an integer in some data tables and a string in others.

Outliers

Outliers are unusual values that are separated from the main body of the distribution, typically as measured by standard deviations from the mean or by the IQR. Whether or not you remove or mitigate the influence of outliers is a critical decision in the modeling process. Outliers can be unusual for different reasons. Sometimes outliers are just data coded wrong; an age value equal to 141 could really be an age equal to 41 that was coded incorrectly. These outliers should be examined, verified, and treated as incorrectly coded values.

However, some outliers are not extremes at all. An outlier defined as an unusual value rather than a value that exceeds a fixed number of standard

deviations from the mean or a multiple of the IQR can be anywhere in the range of the variable, provided there aren't many other values in that range of the variable's values. For example, a profit/loss statement for large organizations could typically have large profits or large losses in the millions of dollars, but magnitudes less than $100 could be very rare. These outliers are not extremes at all; they are at a trough in the middle of a multi-modal distribution. Without data visualization, the analyst may not even know these are unusual values at all because the summary statistics will not provide immediately obvious clues: The mean may be close to zero, the standard deviation reasonable, and there is no skew! These outliers don't cause bias in numeric algorithms, but can still either cause the algorithms to behave in unexpected ways, or result in large modeling errors because the algorithms essentially ignore the records.

If the outliers to be cleaned are examples of values that are correctly coded, there are four typical approaches to handling them:

- **Remove the outliers from the modeling data.** In this approach, the assumption is that the outliers will distort the models so much that they harm more than help. This can be the case particularly with the numeric algorithms (linear regression, k-nearest neighbor, K-Means clustering, and Principal Component Analysis). In these algorithms, outliers can dominate the solution to the point that the algorithms essentially ignore most of the data because they are biased toward modeling the outliers well.

  Removing the outliers can carry some risk as well. Because these values are not included in the modeling data, model deployment could be compromised when outliers appear in the data; records with any outliers will either have to be removed prior to scoring, or the records will be scored based on data never seen before by the modeling algorithms, perhaps producing scores that are unreliable. This will have to be tested during model evaluation to evaluate the risk associated with scoring outliers.

  Another problem with removing outliers occurs when the models are intended to be able to identify a subset of customer behavior that is particularly good or particularly bad. Fraud detection by its very nature is identifying unusual patterns of behavior. The very best customer lifetime value represents the very best of unusual behavior. Removing records with outliers in the inputs could remove those cases that are associated with the most egregious fraud or with the very best customer lifetime value.

- **Separate the outliers and create separate models just for outliers.** This approach is an add-on to the first approach. Records with outliers are pulled out into separate data sets for modeling, and additional models are built just on these data. Because outliers are relatively rare events, the models will likely be simpler, but building models specifically for these outliers will enable you to overcome the problem described in the first approach,

namely the problem of deploying models that did not have outliers in the training data. Some predictive modelers who build separate models for the outliers will relax the definition of outliers to increase the number of records available for building these models One way to achieve this is by relaxing the definition of an outlier from three standard deviations from the mean down to two standard deviations from the mean.

Decision trees can essentially incorporate this approach automatically. If outliers are good predictors of a target variable, the trees will split the data into the outlier/non-outlier data subsets naturally.

- **Transform the outliers so that they are no longer outliers.** Another approach to mitigating the bias caused by outliers that are extreme values is to transform the data so the outliers are no longer outliers. The discussion on skew will describe specific approaches that handle the extreme values smoothly; these techniques all reduced the distance between the outliers and the main body of the distribution. For multi-modal distributions with troughs that contain few records, even simple min-max transformations that reduce the distance between data points can help reduce the impact of outliers in the data.
- **Bin the data.** Some outliers are too extreme for transformations to capture them well; they are so far from the main density of the data that they remain outliers even after transformations have been applied. An alternative to numeric transformations is to convert the numeric variable to categorical through binning; this approach communicates to the modeling algorithms that the actual value of the outliers are not important, but it is the mere fact that the values are outliers that is predictive. This approach is also a good one when the outliers are not extremes and go undetected by automatic outlier detection algorithms. Binning to identify outliers may require some manual tuning to get the desired range of values for the outliers into a single bin. These binned variables will, of course, have to be converted to dummy variables for use in numeric algorithms.

  An alternative to binning the entire variable is to take a two-step approach to outlier mitigation. First, the modeler can create a single-column dummy variable to indicate that the variable value is an outlier. Next, the modeler can transform the outliers as described in the third bullet to mitigate the numeric affect of the outlier on the numeric algorithms.
- **Leave the outliers in the data without modification.** The last approach to handling outliers is to leave them in the data as they appear naturally. If this approach is taken, the modeler could decide to use only algorithms unaffected by outliers, such as decision trees. Or, in some cases, the modeler could use algorithms that are affected by outliers to purposefully bias the models in the direction of the outliers. For example, consider a model

to predict customer lifetime value (CLV) for a retail chain. Assume that there are large outliers at the upper end of CLV, indicative of the best customers for the retailer. The desire of the retailer is to build accurate CLV models, of course. But what kinds of errors are more acceptable than others? If customers with CLV values in the $500–$1,000 range are predicted accurately but at the expense of the $10,000 customers (who are predicted to have CLV values in the same range), is this acceptable?

Consider an example with the variable MAXRAMNT. Table 4-1 shows summary statistic values for MAXRAMNT, a variable with considerable positive skew: Note the mean is larger than the median and that the Maximum is many times larger than the upper bound of the IQR outlier threshold, and the values greater than 36 are considered outliers by the IQR standard (there are 5,023 "outliers" in the 95,412 records).

Table 4-1: Summary Statistics for MAXRAMNT

STATISTIC	VALUE
Minimum	5
Lower IQR outlier threshold	5
1st quartile	14
Median	17
Mean	19.999
3rd quartile	23
Upper IQR outlier threshold	36
Maximum	5000

The business objective will often dictate which of the five approaches in the preceding bullets should be used to treat the outliers. Consider, however, the fourth option: binning. Figure 4-1 shows the results from creating equal-count bins. If only the extreme outliers are flagged, any value greater or equal to 75 can be given the value 1 in a dummy variable, flagging 893 records, just under 1 percent of the population. The modeler could decide to lower the threshold under 50, depending on how many records you want to include in the dummy variable.

You may decide to deliberately bias the model toward the larger CLV values so that, even if there are errors in predicting $500 customers, as long as the $10,000 customers are identified well, the model is a success. In this case, leaving the data alone and allowing the models to be biased by the large magnitude may be

exactly what is most desirable for the retailer. The decision to mitigate outliers, therefore, depends on the business objectives of the model.

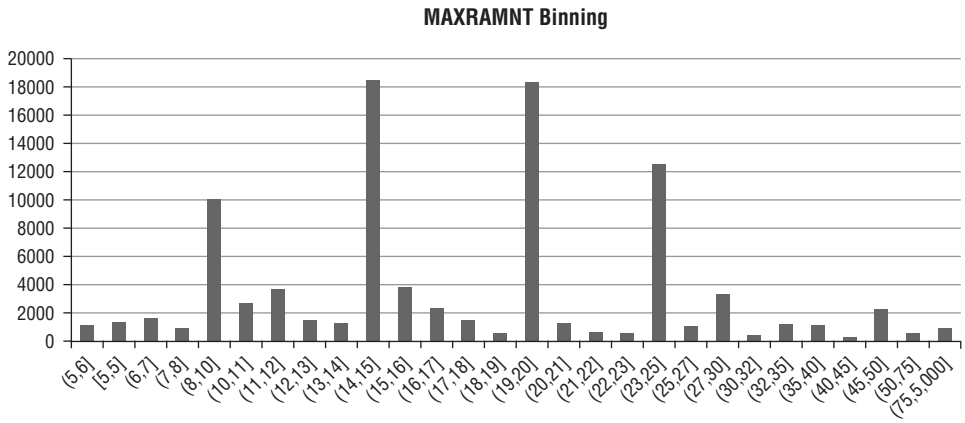


Figure 4-1: Binning of MAXRAMNT

Multidimensional Outliers

Nearly all outlier detection algorithms in modeling software refer to outliers in single variables. Outliers can also be multidimensional, but these are much more difficult to identify. For example, donors to a non-profit organization who have given lots of gifts historically are much more likely to give in small amounts, and those who donate infrequently are more likely to give in larger amounts. There may be interesting outliers in the data representing large-dollar, frequent donors. Neither single-dimension may be an outlier, but the combination is. The problems with these outliers are different than the single-variable outliers; the problem is not with numeric algorithm bias, but rather with the lack of data in the region with the outliers that makes good predictions difficult to achieve.

Predictive modelers rarely try to identify multidimensional outliers before modeling, but they sometimes uncover them after building and assessing the models. If you would like to identify potential multidimensional outliers, one way to do so is with unsupervised learning techniques such as clustering; clusters that have relatively few values in them are indicative of unusual groups of data, and therefore could represent multidimensional outliers.

To mitigate the effects of multidimensional outliers, the modeler could apply the same approaches described for single-variable outliers. The only difference is that in the fourth item in the previous list, rather than creating a dummy variable for each variable separately, you could create a dummy variable indicating the interaction is an outlier or indicating membership in a cluster that identifies the multidimensional outliers.

Missing Values

Missing values are arguably the most problematic of all data problems. Missing values are typically coded in data with a *null* value or as an empty cell, although many more representations can exist in data. Table 4-2 shows typical missing values you may encounter in data.

Table 4-2: Typical Missing Values Codes

POSSIBLE REPRESENTATION OF MISSING VALUES	DESCRIPTION
Null, empty string ("")	For numeric or categorical variables
0	For numeric variables that are never equal to zero
–1	For numeric variables that are never negative
99, 999, and so on	For numeric variables that have values less than 10, 100, and so on
–99, –999, and so on	For numeric variables that can be negative
U, UU	For categorical variables, especially 1- or 2-character codes
00000, 00000-0000, XXXXX	For ZIP Codes
11/11/11	For dates
000-000-0000, 0000000000	For phone numbers

Sometimes missing values are even coded with a value that looks like a legitimate value. For example, customer income code may have values 1 through 9, but the value 0 means the income code is missing. Or Gender of Head of Household could be coded "M" for male, "F" for female, "J" for joint (joint head of household, and "D" (did not respond). The "D" doesn't immediately look like a missing value though in this case it is.

One problem with understanding missing values is that the mere fact that the data is missing is only part of the story. Some missing values are due to simple data entry errors. Some missing values mean nothing more than that the values are completely unknown. This can be because the data has been lost inadvertently through data corruption or overwriting of database tables, or because it was never collected in the first place because of data collection limitations or even forgetfulness. Some data is missing because the data was deliberately withheld during data collection, such as with surveys and questionnaires. In the case of deliberate omission of values, there may be predictive information just from the value being missing.

In censored cases, data is not collected purposefully, often because organizations do not have the resources to collect data on every individual or business and have therefore decided to omit some from the data collection process. Marketing organizations do this routinely when they contact some customers but not others. Tax collection agencies do this when they audit some individuals or businesses and not others. The audit outcome for those individuals not audited, of course, is missing: We don't know what that outcome would have been.

MCAR, MAR, and MNAR

The abbreviations MCAR, MAR, and MNAR are used in many cases by statisticians but rarely in predictive modeling software. However, the meaning of these different types of missing values, when known by the modeler, can affect how one chooses to impute the missing values.

MCAR is the abbreviation for *missing completely at random* and means that there is no way to determine what the value should have been. Random imputation methods work as well as any method can for MCAR. MAR is the most confusing of the abbreviations, and means *missing at random*. While this seems to be the same as MCAR, MAR really implies a conditional relationship between the missing value and other variables. The missing value itself isn't known and cannot be known, but it is missing because of another observed value. For example, if answering "yes" for Question 10 on a survey means one doesn't answer Question 11, the reason Question 11 is missing is not random, but what the answer to Question 11 would have been had it been answered cannot be known.

MNAR is the abbreviation for missing not at random (not a very clear way of describing it). In this case, the values of the missing value can be inferred in general by the mere fact that the value is missing. For example, a responder in a survey may not provide information about a criminal record if they have one, whereas those without a criminal record would report "no record." MNAR data, if you know or suspect the data is MNAR, should not be imputed with constants or at random if possible because these values will not reflect well what the values should have been.

Fixing Missing Data

Missing value correction is perhaps the most time-consuming of all the variable cleaning steps needed in data preparation. Whenever possible, imputing missing values is the most desirable action. Missing value imputation means changing values of missing data to a value that represents a plausible or expected

value in the variable if it were actually known. These are the most commonly used methods for fixing problems associated with missing values.

Listwise and Column Deletion

The simplest method of handling missing values is *listwise deletion*, meaning one removes any record with any missing values, leaving only records with fully populated values for every variable to be used in the analysis. In some domains, this makes perfect sense. For example, for radar data, if a radar return has missing values, it most likely means there was a data collection problem with the particular example, leaving that record suspect. If the occurrence of missing values is rare, this will not harm the analysis.

However, many data sets have missing values in most if not all records, especially in domains related to the behavior of people, such as customer acquisition or retention and survey analysis. One may begin with more than 1 million records, but after listwise deletion, only 10,000 remain. Clearly, this isn't desirable and is rarely done in customer analytics.

Sometimes, you find that predictive modeling software performs listwise deletion by default without alerting the user that this is what is happening. You only discover this default when you examine reports on the models that have been built and can see that very few records were actually used in building the models.

An alternative to listwise deletion is *column deletion*: removing any variable that has any missing values at all, leaving only variables that are fully populated. This approach solves the listwise deletion problem of removing too many records, but may still be too restrictive If only a few of the values are missing in a column, removing the entire column is a severe action. Nevertheless, both listwise deletion and column deletion are practiced, especially when the number of missing values is particularly large or when the timeline to complete the modeling is particularly short.

Imputation with a Constant

This option is almost always available in predictive analytics software. For categorical variables, this can be as simple as filling missing values with a "U" or another appropriate string to indicate missing. For continuous variables, this is most often a 0. For some continuous variables, filling missing values with a 0 makes perfect sense. Bank account balances for account types that an individual does not have can be recoded with a 0, conveying information about the account that makes sense (no dollars on account). Sometimes, imputing with a 0 causes significant problems, however. Age, for example, could have a range of values between 18 and 90. Imputing missing values with a 0 clearly doesn't convey a value that is possible, let alone likely, for donors in a data set.

Mean and Median Imputation for Continuous Variables

The next level of sophistication is imputing with a constant value that is not predefined by the modeler. Mean imputation is perhaps the most commonly used method for several reasons. First, it's easy; mean values are easy to compute and often available without extra computational steps. Second, the idea behind mean imputation is that the value that is imputed should do the least amount of harm possible; on average, one expects the values that are missing to approach the mean value.

However, mean imputation has problems as well. The more values that are missing and imputed with the mean, the more values of the variable fall exactly at the mean, so a spike emerges in the distribution. The problem is primarily with algorithms that compute summary statistics; the more values imputed with the mean, the smaller the standard deviation becomes when you compare the standard deviation of the variable before and after imputation.

Consider the variable AGE, shown in Figure 4-2. The missing values, 25 percent of the population, are shown as a separate bar at the right end of the histogram.

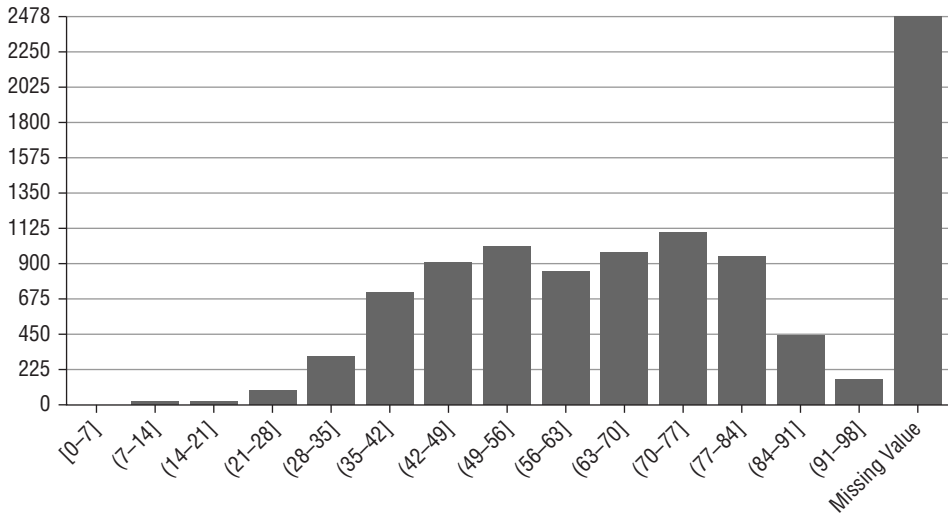


Figure 4-2: Histogram for AGE, including missing values

Table 4-3 shows the summary statistics for AGE: a mean of 61.64 and standard deviation with 16.62. If you were to impute missing values for AGE when AGE has 25 percent, 40 percent, and 62 percent missing, you would see the standard deviation shrink from 16.6 to 14.4, 12.9, and 10.2 respectively. What causes the shrinking of standard deviation?

Table 4-3: Shrinking Standard Deviation from Mean Imputation

VARIABLE	ORIGINAL MEAN	ORIGINAL STANDARD DEVIATION	MISSING	PERCENT MISSING	MEAN AFTER IMPUTATION	STANDARD DEVIATION AFTER IMPUTATION
AGE (25% missing)	61.6	16.6	23,665	24.8	61.64	14.4
AGE (40% missing)	61.6	16.6	37,984	39.8	61.63	12.9
AGE (62% missing)	61.6	16.6	59,635	62.5	61.67	10.2

Figure 4-3 shows the plots for AGE with 25 percent and 40 percent missing values imputed with the mean, 61.6. The shrinking of the standard deviation occurs because of the spike. The more missing values imputed with the mean, the larger the spike relative to the rest of the distribution.

Mean imputation is by far the most common method, but in some circumstances, if the mean and median are different from one another, imputing with the median may be better because the median will represent better the most typical value of the variable. However, median imputation can be more computationally expensive, especially if the number of records in the data is large.

Imputing with Distributions

When large percentages of values are missing, the summary statistics are affected by mean imputation. An alternative to this is, rather than imputing with a constant value, to impute randomly from a known distribution. For the variable AGE, if instead of imputing with the mean (61.6), you impute using a random number generated from a normal distribution with mean 61.6 and standard deviation 16.6 (see Table 4-4). These imputed values will retain the same shape as the original shape of AGE, though there will be small differences because AGE was not shaped exactly like a normal distribution originally. If the variable appears to be more uniformly distributed, you can use a random draw from a uniform distribution rather than a normal distribution for imputation.

Under most circumstances, imputing missing values from a distribution rather than from the mean is preferred, though sometimes the simplicity of mean imputation is preferred.

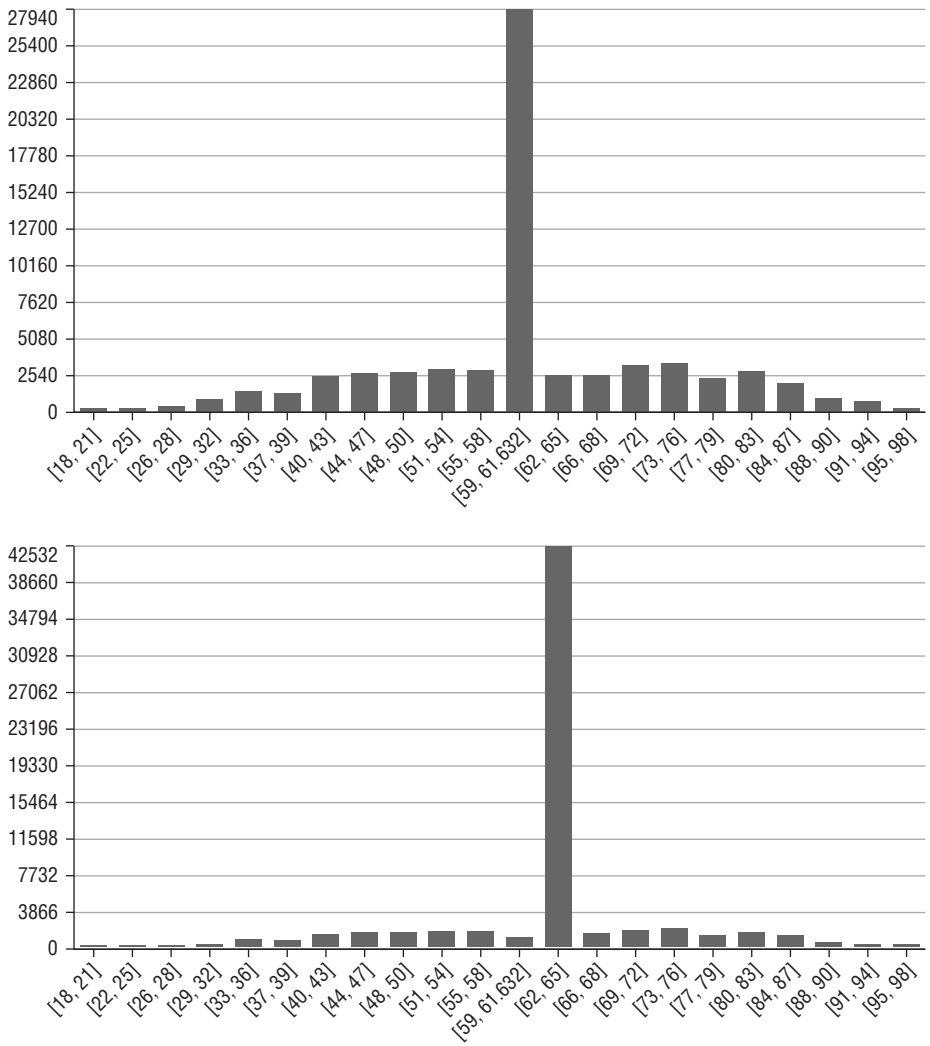


Figure 4-3: Spikes caused by mean imputation

Table 4-4: Comparison of Standard Deviation for Mean and Random Imputation

AGE	MEAN	STANDARD DEVIATION	STANDARD DEVIATION AFTER MEAN IMPUTATION	STANDARD DEVIATION AFTER RANDOM IMPUTATION
25% missing	61.6	16.6	14.4	16.6
40% missing	61.6	16.6	12.9	16.6
63% missing	61.6	16.6	10.2	16.6

Random Imputation from Own Distributions

A similar procedure is called "Hot Deck" imputation, where the "deck" referred originally to the days when the Census Bureau had cards for individuals. If someone did not respond, one could take another card from the deck at random and use that information as a substitute.

In predictive modeling terms, this is random imputation, but instead of using a random number generator to pick a number, a random *actual* value of the variable of the non-missing values is selected. Predictive modeling software rarely provides this option, but it is easy to do. One way to do it is as follows:

1. Copy the variable to impute (variable "*x*") to a new column.
2. Remove missing values and randomly scramble that column.
3. Replicate values of the column so that there are as many values as there were records in the original data.
4. Join this column to the original data.
5. Impute missing values of variable *X* with the value in the scrambled column.

The advantage of random imputation from the same variable is that the distribution of imputed values matches the populated data even when the shape of the data is not uniform or normal.

Imputing Missing Values from a Model

Imputing missing values as a constant (the mean or median) or with a random value is quick, easy, and often sufficient to solve the problem of missing data. However, better imputation can be achieved for non-MCAR missing values by other means.

The model approach to missing value imputation begins with changing the role of the input variable with missing values to now be a target variable. The inputs to the new model are other input variables that may predict this new target variable well. The training data should be large enough, and all of the inputs must be populated; listwise deletion is an appropriate way to remove records with any missing values. Keep in mind that even a moderately accurate model can still produce good imputations; the alternatives are either random or constant imputation methods.

Modelers have two problems with using models to impute missing values. First, if the software doesn't have methods to impute values from models automatically, this kind of imputation can take considerable time and effort; one must build as many models as there are variables in the data.

Second, even if the imputation methods are done efficiently and effectively, they have to be done again when the models are deployed; any missing values in data to be scored must first be run through a model, adding complexity to any deployment process.

Nevertheless, the benefits may outweigh the problems. It seems that the modeling algorithms most often included in predictive analytics software to

automatically impute missing values are decision trees and k-nearest neighbor, both of which can work well with imputing continuous and categorical values.

Dummy Variables Indicating Missing Values

When missing data is MAR or MNAR, the mere fact that the data is missing can be informative in predictive modeling. Capturing the existence of missing data can be done with a dummy variable coded as 1 when the variable is missing and 0 when it is populated.

Imputation for Categorical Variables

When the data is categorical, imputation is not always necessary; the missing value can be imputed with a value that represents missing so that no cell contains a null any longer. This approach works well when the missing value is MCAR or when "missing" itself is a good predictor.

If the data is MNAR, meaning that other variables could predict the missing value effectively, there may be advantages to building predictive models to predict the variable with missing values, just as was described for continuous variables. The model can be as simple as a crosstab or as complex as the most complex predictive model, but in either case, the predicted categorical value should be a better guess for what the missing value would have been than a random or constant imputed value.

How Much Missing Data Is Too Much?

When is it no longer worthwhile to try to impute missing values? If 50 percent of the values are missing, should the variable just be removed from the analysis? The question is complex and there is no complete answer. If you are trying to build a classification model with the target variable populated with 50 percent 1s and 50 percent 0s, but one of the candidate variables is 95 percent missing, that variable is very unlikely to be useful, and will usually be removed from the analysis. But what if the target variable is populated with 99 percent 0s and 1 percent 1s? The proportion of populated input values is five times that of the target; it may still be a useful predictor.

Software Dependence and Default Imputation

Some final recommendations to consider with missing values include:

- **Know what your software does with missing values by default.** Some software provides for no special handling of missing values. Algorithms will throw errors if there are any missing values at all in candidate inputs. Some software will perform listwise deletion automatically if there are any

missing values, and you will only discover this by looking at diagnostics or log files created by the software. Some software is more sophisticated and will automatically impute missing values, sometimes with the mean, sometimes with 0 (not a good default choice in general), sometimes even with the midpoint (also not a good default unless the data has already been normalized).

- **Know the options in your software for imputing missing values.** Most predictive analytics software provides options for imputing with the mean or a constant or your choice. Sometimes imputing with a model is possible but hidden behind advanced options.
- **Consider including dummy variables as missing value indicators.** Dummy variables can provide significant information for predictive models in and of themselves. These dummy variables don't have to take the place of the original variable or impute missing values for the variable; they can be additional candidate inputs. A second reason for including dummy variables to indicate missing values, especially for continuous variables, is that for some modeling algorithms, such as decision trees, the story associated with a missing value dummy can add insight into the predictions; sometimes the dummy variable can tell you more than an imputation.

Feature Creation

Features are new variables to add to data that are created from one or more existing variables already in the data. They are sometimes called *derived variables* or *derived attributes*. Creating features provides more value-added to the quality of data than any other step. They are also the great equalizer between algorithms: Good features reduce the need for specialization and extensive experimentation with algorithms because they can make the patterns that relate inputs to the target variable far more transparent.

Simple Variable Transformations

Skewness was defined in Chapter 3, and examples of positive and negative skew were shown in Figure 3-3. Many practitioners want to correct for skew because of the assumptions many algorithms make regarding the shape of the data, namely normal distributions. For example, when data has positive skew, algorithms such as linear regression, k-nearest neighbor, K-Means the positive tail of the distribution will produce models with *bias*, meaning that the coefficient and influence of variables are more sensitive to the tails of the skewed distribution than one would find if the data were normally distributed. Some analysts will dismiss the models as not being truly representative of the patterns in the data.

For example, consider a linear regression model that computes the square of the errors between the data points and the trend line. Figure 4-4 shows one example. The larger the values, the larger the error could be. In Figure 4-4, the error for the x-axis value equal to 500 is 150 units, 100x more than the actual values at the left end of the x-axis. But the problem is even worse than it appears. The Regression model computes the *square* of the error, so 150 units is 150^2 = 22,500 units, thousands of times greater than the errors at the left end of the plot.

Therefore, to minimize the square of the errors, the regression model must try to keep the line closer to the data point at the right extreme of the plot, giving this data point a disproportionate influence of the slope of the line. Of the 28 data points in the plot, the square of the error for the data point with an x-axis value equal to 500 is 21 percent of the total error of all the data points. The four data points with x-axis values 250 or above contribute more than 60 percent of the error (for 14 percent of the data). Clustering methods such as K-Means and Kohonen use the Euclidean Distance, which computes the square of the distances between data points, and therefore tails of distributions have the same disproportionate effect. Other algorithms, such as decision trees, are unaffected by skew.

The power to raise the distribution is not always known beforehand and may require some experimentation.

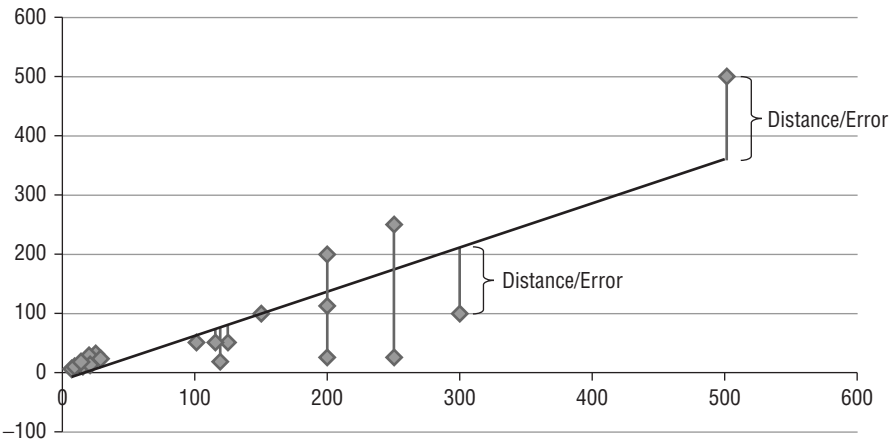


Figure 4-4: Effect of skew on regression models

Fixing Skew

When a skewed distribution needs to be corrected, the skewed variables are typically transformed by a function that has a disproportionate effect on the tails of the distribution. Ideally, for most modeling algorithms, the desired outcome of skew correction is a new version of the variable that is normally distributed.

However, even if the outcome is a variable that has a uniform distribution, as long as the distribution is balanced—skew equals a value close to 0—the algorithms will behave in a less biased way.

For positive skew, the most common corrections are the log transform, the multiplicative inverse, and the square root transform. Table 4-5 shows the formulas for these three transformations. Note that in all three cases, they operate by reducing the larger values more and reducing (or even expanding) the smaller values less, or, in the case of the inverse, actually increasing the smaller values.

Table 4-5: Common Transformations to Reduce Positive Skew

TRANSFORM NAME	TRANSFORM EQUATION
Log transform	log(*x*), logn(*x*), log10(*x*)
Multiplicative inverse	1/*x*
Square root	sqrt(*x*)

Of these, the log transform is perhaps the most often used transformation to correct for positive skew. A log transform of any positive base will pull in the tail of a positively skewed variable, but the natural log (base "e," 2.7182818) or the log base 10 are usually included in software. The log base 10 pulls the tail in more than the natural log and is preferred sometimes for this reason. Another reason to prefer the log base 10 is that translation from the original units to log units is simpler (see Table 4-6). The log base 10 increments by 1 for every order of magnitude increase, whereas the natural log has increments of 2.3.

Table 4-6: Log Unit Conversion

NATURAL UNITS	LOG10 UNITS	LOG*N* UNITS
0	Undefined	Undefined
1	0	0
10	1	2.3 . . .
100	2	4.6 . . .
1,000	3	6.9 . . .
10,000	4	9.2 . . .
Natural units	Log10 units	Log*n* units

The effect of the transformation can be seen in Figure 4-5. The histogram at the left shows original units and the one to the right shows the distribution after applying the log10 transform. The skew for the histogram on the left was 3.4, and on the right only 0.2; the log transform removed nearly all of the skew.

The square root and multiplicative inverse on this same data help remove skew but not as effectively for this data as the log transform.

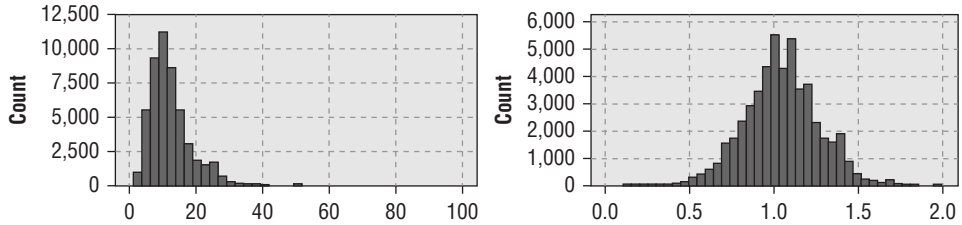


Figure 4-5: Positively skewed distribution normalized with log transform

Because the log transform is undefined for the values less than or equal to zero, you must take care that there are no 0 or negative values in the data, or you introduce undefined values into the transformed data, which are treated by most software as missing. Then you would have to follow the same process already discussed for imputing missing values.

However, a simple fix to the 0 values for data that is non-negative is to add 1 to the value that is transformed: $\log10(x + 1)$. This has two helpful effects. First, 0s, instead of becoming undefined, are now transformed to the value 0, exactly where they started, so 0s stay as 0s. Second, because the log transform of a value between 0 and 1 is negative, adding 1 to the original value keeps the log transform positive just like the original version of the variable. For this reason, practitioners often make adding 1 to the original variable a common practice.

On rare occasions, the original variable is positively skewed but contains negative values as well. In these situations, you can add the absolute value of the minimum value of the variable (the magnitude of the most negative value) plus 1 so that the log transform is never undefined:

$$\log10(|\min(x)| + 1 + x)$$

Negative skew is less common than positive skew but has the same problems with bias that positive skew has. Of course with negative values, the log transform cannot be used as it was with positive skew. One correction used often to transform negatively skewed variables is a power transform: square, cube, or raise the variable to a higher power.

If the variable has a large magnitude, raising that value to a high power will create very large transformed values and even cause numeric overflow. It is therefore advisable to scale the variable first by its magnitude before raising it to a high power.

Consider the four plots in Figure 4-6. The upper-left histogram is the negatively skewed original variable (skew = –3.4). The subsequent three histograms show the distributions when the power transform is applied using power 2, 20, and 80.

The last of these finally achieves the desired effect of creating a version of the variable with zero skew. The power to raise the distribution to is not always known beforehand and may require some experimentation.

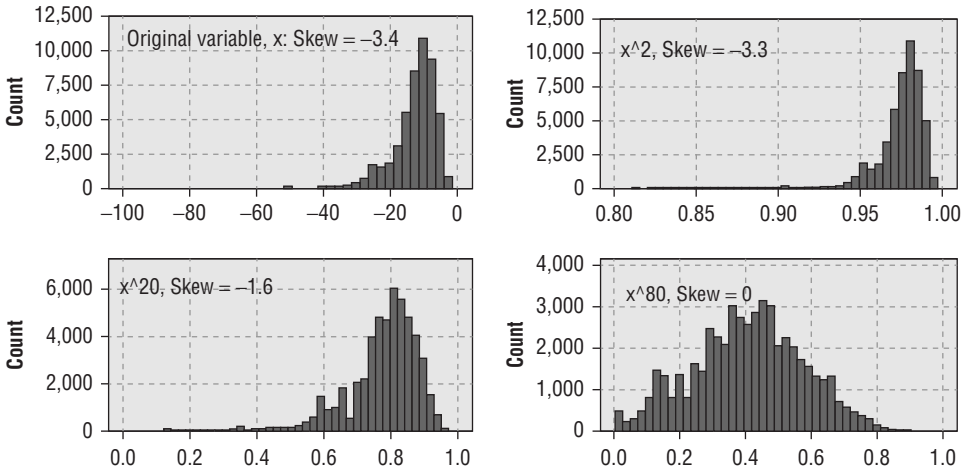


Figure 4-6: Negatively skewed distribution normalized with the power transform

An alternative approach is to take advantage of the same log transform already described. However, because the log transform is undefined for negative values, you must first ensure the values are positive before applying the log. Moreover, the log transform pulls in the tail of a positively skewed distribution. Therefore, you must first flip the distribution before applying the log transform and then restore the distribution to its original negative (now transformed) values. The equation for transforming either a positively or negatively skewed variable is as follows:

$$\text{Transformed\_}x = \text{sgn}(x) \times \log10(1 + \text{abs}(x))$$

Inside the log transform, an absolute value is applied first to make the values positive. One (1) is then added to shift the distribution away from zero. After applying the log transform, the sgn(x) function applies the original sign of the variable back to the distribution. If the original values were negative, this restores the negative sign. Note that if the original values are all positive, like the ones shown in Figure 4-5, the absolute value and `sgn` function have no effect, so this formula can be used for either positively or negatively skewed data.

In fact, this transformation can be used in yet another distribution, one with large tails in both the positive and negative directions but peaks at 0. This occurs for monetary fields such as profit/loss variables.

In summary, Table 4-7 lists transformations that are effective in converting positively or negatively skewed distributions to distributions that are at

least more balanced and often are more normally distributed. The particular transformation that is used by a predictive modeler will vary depending on personal preferences and the resulting distribution found after applying the transformation.

Table 4-7: Table of Corrective Actions for Positive or Negative Skew

PROBLEM	TRANSFORM
Positive skew	log10(1 + *x*), 1/*x*, sqrt(*x*)
Negative skew	x^n, –log10(1 + abs(*x*))
Big tails, both directions	sgn(*x*) x log10(1 + abs(*x*))

Binning Continuous Variables

Correcting skewed distributions is a straightforward process, but other more complex distributions require more interaction by the modeler. If the variable has long tails and outliers, is multi-modal, has spikes, or has missing values, a binned version of the variable may be a better representation for modeling rather than trying to convert the odd distribution to one that appears normally or uniformly distributed.

For example, consider the distribution in Figure 4-7. Note there are two main peaks to the distribution, and at the far right, there is also a small bin representing missing values. This population is difficult to represent as a continuous variable for algorithms that assume normal distributions. A different representation that may help modeling is one that captures the two groups of values, as shown in Figure 4-8.

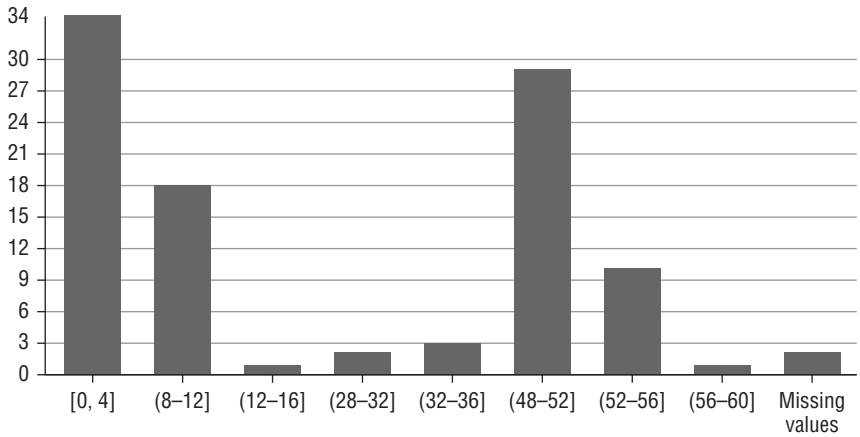


Figure 4-7: Multi-modal distribution

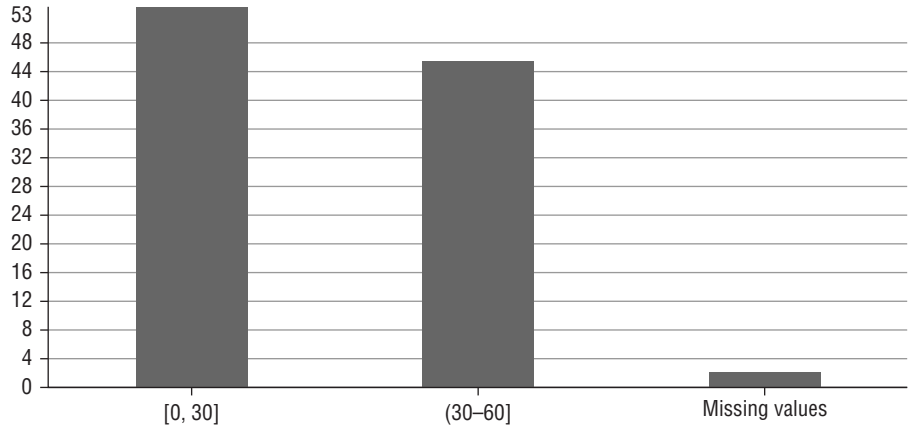


Figure 4-8: Binned version of multi-modal variable

Numeric Variable Scaling

Just as positive and negative skew can bias models toward the tails of the distribution, the magnitudes of variables can also bias the models created by some algorithms, especially those that are built based on Euclidean Distances such as K-Means clustering and k-nearest neighbor. Magnitudes affect these kinds of algorithms for the same reason pictured in Figure 4-1: Larger magnitudes produce larger distances. Scaling variables so that all have the same magnitudes removes this bias.

Linear regression coefficients are influenced disproportionately by the large values of a skewed distribution, but not by the magnitudes of normally distributed variables; the coefficients of a regression model scale the magnitudes of inputs appropriately. Most implementations of neural networks also scale the inputs before building the models. Scaling may need to be done even after transforming variables through the use of log transforms. Some software packages refer to this kind of scaling as normalization.

A list of commonly used normalization methods appears in Table 4-8, where the most common of these are min-max normalization and z-scores. The magnitude scaling method, just scaling the variable by its maximum magnitude, will create a maximum value of -1 or +1 depending on whether the maximum magnitude is negative or positive but is not guaranteed to fill the entire range from -1 to +1. The sigmoid, a transformation also used in neural networks and logistic regression, sometimes requires some experimentation to find a value for the scaling factor, c, that makes the sigmoid shape good for the particular variable.

Table 4-8: Commonly Used Scaling and Normalization Methods

SCALING METHOD	FORMULA	RANGE
Magnitude scaling	$x' = x / \max\|x\|$	[-1,1]
Sigmoid	$x' = 1 / (1 + e(-x/c))$	[0,1]
Min-max normalization	$x' = (x - x_{min})/(x_{max} - x_{min})$	[0,1]
Z-score	$x' = (x - x_{mean})/x_{std}$	mostly [–3,3]
Rank binning	100*rank order / # records	[0,100]

Min-max normalization merely changes the range of a variable to the range from 0 to 1, compressing variable magnitudes if the maximum is greater than 1. If the maximum of a variable is less than 1, min-max scaling actually expands the range of the variable. The min-max values therefore represent the percentage of the value of the variable relative to its maximum.

Z-scores are most appropriate when the data is normally distributed because the mean and standard deviation used in the scaling assume normal distributions. The interpretation of z-scores is straightforward: Each unit refers to one unit of standard deviation from the mean. Consider Table 4-9, showing 10 values of AGE with the corresponding min-max and z-score normalized values. The AGE value equal to 81 years old is 78 percent of the distance from the minimum to the maximum, but represents just over one standard deviation from the mean, where standard deviations assume AGE follows a normal distribution. The mean appears to be near 62 years old as this value is nearly 0 in z-scored units.

Table 4-9: Sample Z-Scored Data

AGE	AGE (MIN-MAX)	AGE (Z-SCORE)
81	0.7821	1.1852
62	0.5385	–0.0208
79	0.7564	1.0583
66	0.5897	0.2331
69	0.6282	0.4235
73	0.6795	0.6774
28	0.1026	–2.1790
50	0.3846	–0.7825
89	0.8846	1.6930
58	0.4872	–0.2747

Normalizing data can aid considerably in data visualization as well. In Figure 4-9, the variables AGE and MAXRAMNT are plotted. MAXRAMNT has a considerable outlier, a value equal to $1,000. The scale on the x-axis is linear, as is typical for scatterplots, and therefore the plot contains primarily white space, which provides very little visual information to the analyst.

Normalization of the data can help utilize more the space on the printed page available to view the data. If MAXRAMNT is log transformed to compress the effect of outliers and skew, and then is scaled to the range [0,1], and if the variable AGE is also scaled to the range [0,1], the plot in Figure 4-10 results. Now you can see several patterns that were compressed into the very left side of the natural units plot. First, there are stripes in the data indicating typical gift sizes in the MAXRAMNT variable (people tend to give in $5 or $10 increments). Also, the larger gifts tend to be given by older donors as evidenced by the fact that the lower-right region of the plot has few data points.

While structural information is gained about individual variables and their interactions, scaling and transforming data obscures the original variable values, forcing the analyst to keep the translations in his or her head or superimpose the original values of the variables on the plots. Some software provides the capability to redraw scatterplots with different scales, such as a log scale, but for other scaling methods, the analyst is on his or her own. In Figure 4-10, you know that the maximum age is 98 years old, and therefore the normalized age value 1.0 on the y-axis represents 98 years old. However, the log transform makes the other values more difficult to estimate. What age does the value 0.3 represent? It turns out to be about 45 years old, but this isn't obvious.

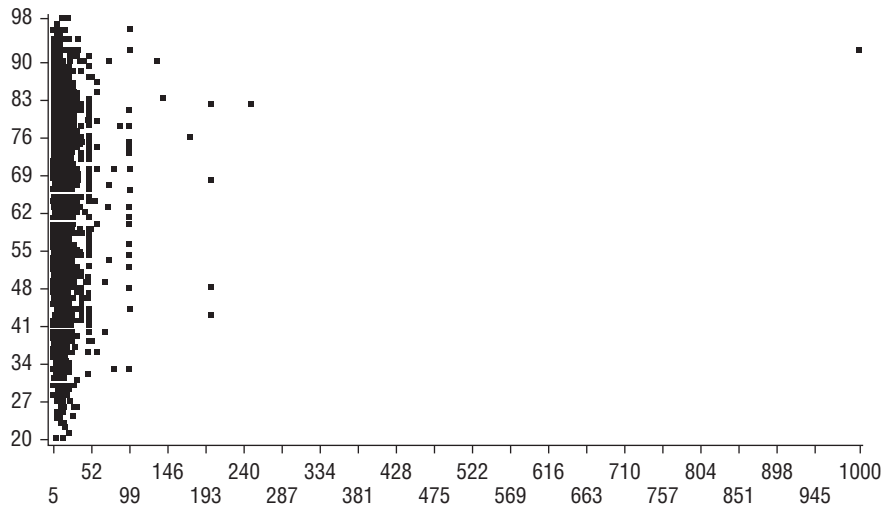


Figure 4-9: AGE versus MAXRAMNT in natural units

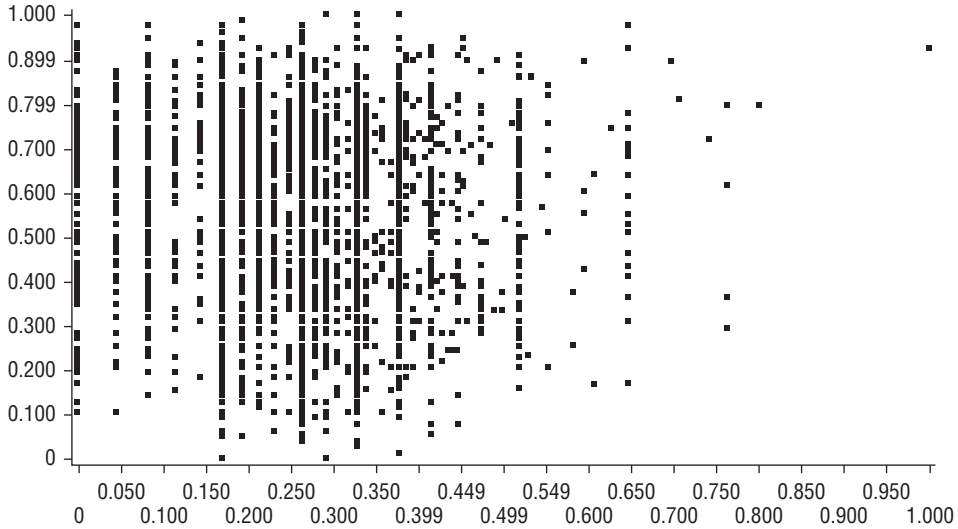


Figure 4-10: AGE versus MAXRAMNT in scaled and normalized units

Nominal Variable Transformation

Nominal variables present problems for numeric algorithms because they usually are not in numeric form, and even if they are numeric, they aren't properly interpreted as numbers in the way modeling algorithms need. The most common approach to using nominal variables is to explode them into *dummy variables*. Dummies are dichotomous variables (two values), usually coded with 0s and 1s, where the value 1 corresponds to "true" and 0 to "false." Recoding of categorical variables is accomplished by exploding a single column of categorical values having *N* values to *N* columns of dummy variables.

For example, in the simple example of Table 4-10, four state values are converted to four columns of dummy variables. With this representation, each record contains one and only one column of the individual states whose value equals 1. The actual numbers representing the dummy variables do not matter, though the 1/0 representation has advantages, including the ability to count for each state how many records hit.

Table 4-10: Exploding Categorical Variables to Dummy Variables

STATE	CA	TX	FL	NY
CA	1	0	0	0
TX	0	1	0	0
FL	0	0	1	0
NY	0	0	0	1

You must also beware of high cardinality—that is, when there are many values in the original categorical variable, especially if it results in one or more of the columns of dummy variables having very few 1s. These columns could be removed during a variable selection stage, however. Or, as an alternative, all columns of the same exploded categorical variable that have small counts (maybe less than 50) could be combined into a single bin. A simple "or" operation for these variables will result in the new binned column having value 1 when any of the columns have value 1, or 0 otherwise.

Ordinal Variable Transformations

Ordinal variables are problematic for predictive modeling and most algorithms do not have specific ways to incorporate ordinal variables in the algorithms. Algorithms that can use the information contained in an ordinal variable are often either not included in predictive modeling software or are not implemented in a way that takes advantage of ordinal variables.

The modeler must usually make a choice: Should the ordinal variable be treated as continuous or categorical? If it is treated as continuous, the algorithm will assume the variable is an interval or ratio variable and may be misled or confused by the scale. For example, if the ordinal variable is education level and is coded as 1 for high school graduate, 2 for some college, 3 for associate's degree, 4 for bachelor's degree, and so forth, a bachelor's degree will be considered twice as big as some college, and four times as big as a high school graduate. On the other hand, if the variable is considered to be categorical, a model could combine high school, bachelor's degree, and PhD in one group, a grouping that doesn't make intuitive sense.

Nevertheless, most often, ordinal variables are treated as categorical, but care should be taken to ensure that non-intuitive grouping or splitting of categories doesn't take place. If it is occurring in the model, consider making the ordinal variable continuous.

One additional consideration for ordinal variables is how to code them when they are used in numeric modeling algorithms. The typical treatment of categorical variables is to explode them into their corresponding dummy variables, a form that results in one and only one column having the value 1, and all others having the value 0.

Ordinal variables are different than nominal, however. For variables such as education, a thermometer scale may be more appropriate. Table 4-11 shows an example of the thermometer scale for education level (not all possible levels are shown to make the table easier to see). With this scale, modeling algorithms will be able to incorporate education levels already achieved in the models. For example, individuals with a master's degree or PhD also (presumably) have a bachelor's degree. If bachelor's degrees are predictive, including the individuals

with master's degrees or PhDs will strengthen the pattern associated with bachelor's degrees.

If the typical explosion of a nominal variable were to be used to measure the effect of a bachelor's degree on the target variable, the model would be trying to find individuals with bachelor's degrees who did not go on to higher degrees. Either one could be useful and predictive, but you don't know which is more predictive unless they are both constructed and tried in models.

Table 4-11: Thermometer Scale

EDUCATION LEVEL	HIGH SCHOOL	SOME COLLEGE	ASSOCIATE'S DEGREE	BACHELOR'S DEGREE	MASTER'S DEGREE	PHD
High school graduate	1	0	0	0	0	0
Some college	1	1	0	0	0	0
Associate's degree	1	1	1	0	0	0
Bachelor's degree	1	1	1	1	0	0
Master's degree	1	1	1	1	1	0
PhD	1	1	1	1	1	1

Date and Time Variable Features

Date and time variables are notoriously difficult to work with in predictive modeling and nearly always require some feature creation to be used effectively. Date features will be described in this section, but the same principles apply to time features as well.

The first problem with date variables is their representation. Dates can be coded in a myriad of ways: March 1, 2013, 1-March-2013, 1-Mar-99, 3/1/99, 3/1/2013, 03/01/2013, 03012013, and many more. Note that all of these formats are absolute dates. Assume, for example, the date refers to the date an individual visited a website. If the visit date is used in a predictive model to predict the likelihood for the visitor to purchase a product, the interpretation may be, "If a visitor visited the website on March 1, 2013, then the visitor is likely to purchase product X." If the model is used week after week, the pattern in the model is still literally checking for a visit on March 1, 2013, a pattern that is unlikely to be the intent of the visitor.

A more desirable interpretation is this: "If a visitor visited the website within the last week, then the visitor is likely to purchase product X." In this case, the feature of interest is a relative date, computed between the date in the

data and "today." Therefore, features such as "days since," "days between," "days until," and so forth need to be computed from the fixed dates in the data. In addition, because the date features are relative dates, the "today" date is critically important to get right, and may itself be relative to other dates in the data. The same principles can apply to calculations related to months, quarters, and years.

However, fixed dates are sometimes more valuable than relative dates. Was the visit on a holiday? Which one? Was the visit on Black Friday? These representations of the dates are domain-specific and require significant domain expertise to create in a way that captures visitor intent. One additional problem that occurs with dates and times but not with other numeric variables is their cyclical nature. An integer representation of the month of the year, a number between 1 and 12 (inclusive), can be an effective representation of month. However, every year, the rollover from 12 (December) to 1 (January) appears to be a huge leap numerically—a difference of 11 months—even though it is only a one-month difference on the calendar. Computing relative months from a fixed month can relieve this problem as well.

ZIP Code Features

ZIP codes, which represent a geographical region, can be good predictors of behavior. They can be represented as five-digit numbers, though they should always be stored as text so the predictive analytics software does not remove the leading zeros for some ZIP codes. There are approximately 43,000 ZIP codes in the United States, far too many for typical predictive modeling algorithms to handle effectively. Some software reduces the cardinality of ZIP codes by using only the first two or three digits to represent the region.

Other variables have the same characteristics as ZIP codes in that they may have leading zeros, such as IDs (customer ID, address ID) and codes (title codes, demographic codes, vendor codes). These should always be stored as text.

However, it is rarely the actual ZIP code itself that is the causal reason for the behavior being measured, but rather the characteristics about the people who live in the ZIP code or the businesses that operate in that ZIP code. An alternative, then, is to summarize characteristics of people or businesses in the ZIP code as the feature to use as a candidate input in predictive models. This is exactly what demographic variable appends do: They summarize age, income, education, race, and other characteristics of a ZIP code in a column of the data.

Which Version of a Variable Is Best?

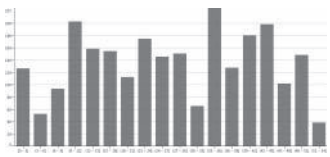
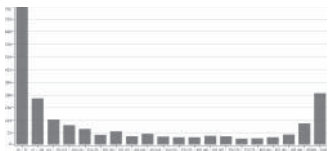
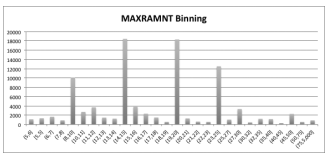
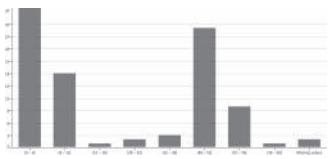
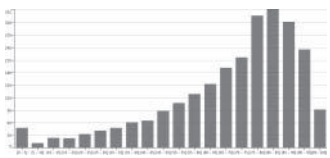
Computing transformed variables is easy: the original version, a log transformed version, one or more binned versions, a z-scored version, and more. But which of these should be used as candidates for the actual modeling? After all,

the original version of a variable and its related log10 transform will be very highly correlated: Their rank-ordering is identical.

Table 4-12 shows a list of histograms containing real-world data, along with the rationale for selecting one of the normalization or scaling methods described in this section.

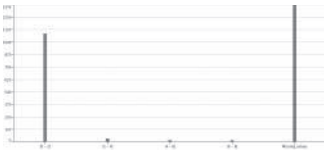
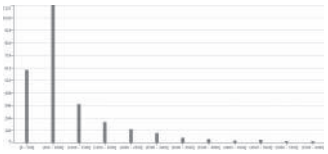
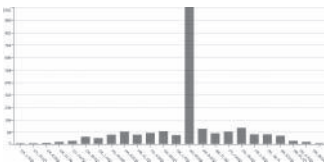
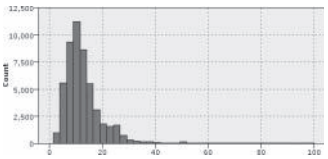
One solution to the problem of selecting which feature to keep is to use an empirical approach to select the best version of a variable for predictive modeling. If you use each version as a candidate input variable by itself to predict the target variable, you can score each version of the variable and pick the one that is the best predictor. This process can be extremely labor intensive if this variable feature search is not automated in the software, or is able to be scripted by the modeler.

Table 4-12: Distributions and Possible Corrective Action

DISTRIBUTION	POSSIBLE CORRECTIVE ACTIONS
	Maybe none; close enough to uniform.
	Primarily positive skew; consider log10 transform.
	Consider binning to capture four or five regions centered on spikes.
	Consider binning into two bins (for peaks) or four bins (two peak and two trough regions).
	Negative skew; consider power transform or flip transform and use log10.

Continues

Table 4-12 (*continued*)

	Consider dummy variables for spikes on left and right.
	Positive skew; consider a log transform.
	Spike in distribution, consider dummy indicator of the spike vs. non-spike. If spike generated through mean imputation, consider imputing with a distribution.
	Classic positive skew. Log10 transform.

Multidimensional Features

Creating features from several variables usually provides more improvement in accuracy than any single-variable transformation because of the information gain associated with these interactions. However, it is often very time-consuming to hypothesize good multidimensional features and then to build them. Several approaches to building multidimensional features are described in this section, including using domain experts, Principal Component Analysis (PCA), clustering, and supervised learning algorithms. The description of how the algorithms are used to create features is included here; a description of algorithms themselves will be described in Chapters 6 and 8.

Domain Experts

So far, the features described have all entailed converting a single variable from one form to a different, more convenient or predictive form. The most powerful features, however, are usually multidimensional features. Two common examples of multidimensional features are interactions (created by multiplying two variables together) and ratios (created by dividing one variable by another). Additive variables (created by adding or subtracting two variables) are not as

useful because most algorithms can compute these features as a part of their algorithms.

Ratios are particularly good features to create for several reasons. First, they can provide a normalized version of a variable that interprets the value of a variable within a record. A simple normalized version of a variable is a percentage; a feature could compute the percentage of visits by a visitor to a website that resulted in a purchase. The denominator, the total number of visits, varies from visitor to visitor, and thus, two purchases can take on a very different meaning depending on how many visits the visitor had. Ratios can also incorporate more complex ideas. The price-to-earnings ratio is a powerful representation of the potential earnings growth: Neither price nor earnings on their own provide this information.

Ratios are important as features because they are difficult for predictive modeling algorithms to uncover: Division is a difficult operator to estimate well. Creating ratios helps the algorithms find patterns more clearly and simply than if they had to approximate the ratio through a series of multiplies and adds.

Where do these powerful ratios come from? Nearly always, ratio features come from domain experts who understand which ratios are likely to be predictive, though sometimes modelers can use their own commonsense to hypothesize new ratio features that are worth examining. During the Business Understanding stage of the modeling project, interviews with domain experts can uncover interesting interaction and ratio features.

Modelers should be prepared to have to convert relatively vague descriptions of features into numeric form. The domain expert might say, "When the price is relatively high compared to earnings, the stock has good potential." Whenever the words "compared to" occur, the modeler can infer that a ratio is being described, price-to-earnings ratio in this case. Whenever the domain expert describes two variables with an "and" between them, multiplication can be inferred: for example, "If donors give frequently and in large amounts per gift, they are excellent donors." The feature in this case is AVGGIFT × NGIFTALL or AVGGIFT × RFA_2F.

Principal Component Analysis Features

Principal Component Analysis (PCA) finds linear projections of numeric data that maximize the spread or variables of the projections. Qualitatively speaking, PCA tries to find correlated variables, variables that have significant overlap with each other, and describe them as Principle Components (PCs). These PCs therefore can be understood as representing a broader idea in the data that several of the variables are representing. PCA models can identify how many real ideas are represented in the data rather than the modeler relying on the variables to indicate how many ideas there are in the data. A PCA model may find that

5 PCs describe most of the data even though there are actually 20 variables in the data. This could happen if the 20 variables are highly correlated with each other.

After building a PCA model, rather than using all 20 variables as inputs to a predictive model, you could use the 5 PCs instead and know that most of the information in the 20 variables is still present in the 5 PCs, but has been compressed into a more parsimonious form. The PCs therefore can be viewed as features of the original data, providing a different representation of the data for the models to use.

PCA can be very useful as a feature extraction algorithm, but also has drawbacks. First, it is only a linear projection of the data. If patterns in the data are nonlinear, the PCA algorithm could even destroy patterns that exist in the data, producing worse models than the original data would itself.

Clustering Features

Clustering can also be used to create one or more features of the original data. The algorithms for building clustering models are described in Chapter 6, but briefly, these algorithms find records that fall into groups from the variable values and assign a label for each record indicating which group the record has been assigned to. Therefore, the clustering algorithm has reduced the data from the number of variables included in the clustering model down to one dimension, the cluster label.

The most straightforward feature to create from a clustering model is the cluster label itself. This label can be converted into dummy variables if necessary. A second set of features you can create from the clustering model is a set of distances from each cluster. For each record, you can create a distance between the record and each of the cluster centers in the model. If there are 10 clusters, there will be 10 distances to compute. These distances can then be used as new features in predictive models. In fact, this is exactly what some predictive modeling algorithms do, like Radial Basis Function Networks.

Other Modeling Algorithms

Neural networks, decision trees, and other predictive modeling algorithms can be used to create features as well. Decision trees are perhaps the most commonly used algorithm for this purpose because they produce rules that are easy to understand and easy to implement. Neural networks can be used, especially if the modeler has a way to expose the outputs of hidden layer neurons in the network; these neurons are essentially feature extractors.

Insights into how to use algorithms for feature creation is deferred to Chapter 8 where the algorithms are described.

Tying Records Together

Predictive modeling algorithms assume each record in the data is an independent observation, unrelated to the other records. Because the algorithms assume this independence, they don't try to tie together records into higher-level features based on multiple records in the data.

In many cases, this is a good assumption. If the data represents customers of a retail chain of stores and their historic purchasing behavior, and if there is one and only one record per customer, the independence of records is a good assumption; in general, customers are not colluding with each other each time they visit the store.

But what if rather than having one record per customer, the modeling data instead represents each visit to any store of that retailer. Now the modeling data may contain multiple entries for each customer, and the records are no longer completely independent. The records are not independent, but the algorithms don't know this and have no mechanism to tie records together and uncover patterns that are only revealed when one examines the sequence of visits.

This kind of modeling data set occurs frequently, identifiable when a key record label, such as Customer ID, Business ID, or transaction code, is not unique in the modeling data. Frequently in these cases, there is also a date or time field, and the unique record ID is the customer ID *and* the date field.

The modeler, in these cases, should create the temporal relationship between records and code it explicitly in the data. For example, a government agency wanted to identify invoices sent to the federal government that might be improper or even fraudulent. Each record was an invoice, and the intent was to identify suspicious invoices and then connect them to the payee, and then investigate the payee. Each payee could be in the modeling data many times, depending on how many invoices he or she sent to the federal government.

One of the features that a domain expert used himself in identifying improper invoices was this: Label invoices that were mailed to the same address, and flag those that had different payees on the invoices. The feature name became "multiple_payees_to_same_address." For example, if there were five invoices paid to a particular address, and three of the five payees were John Smith but two of the five payees were John Smithers, the multiple_payees_to_same_address variable would be populated with a "1" indicating there was more than one payee for that address. This feature was a record-based feature, tying together information about several records, not just tying together values from multiple fields in the same records.

How does one create such a feature? First, the entire database of invoices had to be scanned, and all addresses that were identical were flagged. Then within each group of invoices that went to the same address, a second scan of the data

determined through a fuzzy match how many payees there were on invoices going to the address. If there was more than one payee going to the address, the invoice was flagged as having this multi-record characteristic.

One additional consideration when creating record-based features on temporal data is to ensure that in each record, no information for records with dates future to that record are included.

Time Series Features

Predictive modeling data is inherently two-dimensional, formatted into rows and columns because modeling algorithms work in two dimensions. Introducing a time dimension into modeling data increases the layout of data from two to three dimensions, requiring transformation of data into two-dimensional layouts.

For example, the KDD Cup 1998 donor data consists of rows of donors and columns of attributes. However, these attributes were not always organized in neat columns. Consider donor history tables. Each record is a transactional summary of a donor giving a gift. The unique identifier of these records is donor-date pairs, as shown in Figure 4-11, where CID is the donor ID, the Date is the date of the gift, and Amount is the amount of the gift.

CID	Date	Amount
13013413	2/13/2005	80
13013413	5/9/2005	59
13013413	1/16/2006	22
13013413	3/16/2006	59
13013413	3/28/2007	49
13013413	6/6/2007	189
13013413	7/12/2007	36
13013413	9/19/2007	87
23191348	8/25/2005	23
23191348	11/2/2005	123
23191348	3/14/2006	38
23191348	5/25/2006	26
23191348	11/16/2006	61
23191348	12/16/2006	193
23191348	1/6/2007	88
23191348	4/11/2007	157
23191348	6/5/2007	181
23191348	6/8/2007	90
31348313	1/31/2005	61
31348313	2/16/2006	110
31348313	5/16/2006	19
31348313	10/21/2006	163

CID	Amount_Sum	Amount_Mean	Amount_Max
13013413	$ 581	$ 72.63	$ 189
23191348	$ 980	$ 98.00	$ 193
31348313	$ 383	$ 95.75	$ 163

Figure 4-11: Creating features from time series data

In order to use this data in predictive modeling with each record representing a donor, the data must first be rolled up to the donor level (a groupby operation). Capturing any date or gift amount information can only be accomplished through feature creation. Three simple features are shown in Figure 4-11, but many more are usually created.

Some commonly computed features for donations are already in the KDD Cup 1998 data, including the mean donation amount (AVGGIFT), the most recent donation amount (LASTGIFT), the minimum donation amount (MINRAMNT), the maximum donation amount (MAXRAMNT), and the sum of all donation amounts (RAMNTALL). Common date variable rollups include the earliest donation date (FISTDATE) and the most recent donation date (LASTDATE).

Individual donation amounts are captured through a process often referred to as "pushing the data to the right," meaning new columns are created to capture data originally appearing in rows of the data. Time lags can be represented in absolute or relative form. In absolute form, separate columns are created for the first gift, second gift, third gift, and so on. In relative form, the new columns represent the last gift, the next to last gift, and so on.

The constraint on this process, however, is that the data must be rectangular. If donors give differing numbers of gifts, the modeler must find a way to represent these donors in a consistent manner. If, for example, one donor gives 4 gifts and another 20 gifts, the data cannot have 4 columns for the first donor and 20 for the second; the data must be coded consistently for both. This could be that 20 columns are created for both, but the 4-gift donor has only 4 of them populated, or that the modeling data contains only 4 columns so that the 20-gift donor has only 4 of the gifts shown in the record.

There is no theory to tell the modeler to build features based on fixed or relative time, how many lags to include in the historic data, and which summary statistics to create (min, max, mean, etc.). Domain expertise helps considerably in this process, but oftentimes modelers will create as many different kinds of features as can be handled by the modeling software.

Variable Selection Prior to Modeling

Now that data has been cleaned and new features have been created, one may think that it's now time to begin modeling. However, I recommend that an additional process be added: Reduce variables through variable selection. The goal of variable selection is to remove irrelevant variables, redundant variables, and variables that are unlikely to improve model accuracy.

Note that the techniques described here differ from variable selection in the context of predictive modeling. Forward and backward variable selection algorithms are often added to modeling algorithms to select variables in an efficient, albeit greedy, manner. Decision trees perform variable selection as a natural part of the learning algorithm. Stepwise Regression is a common variable selection technique

that has been integrated into linear regression. More sophisticated techniques such as using guided random search (genetic algorithms or simulated annealing) have been added to neural networks to improve variable selection.

These techniques are described in the context of the algorithms themselves, and while many are very time consuming, they have the advantage that the variable selection is done in the context of the modeling. When variable selection is wrapped around building neural networks, for example, the variables that are retained are those that have worked best for the neural network, not variables that appear to be good predictors using a different metric, such as the chi-square test or the F test. If there is sufficient time available for incorporating variable selection with the modeling algorithm, it is always the preferred method.

Removing Irrelevant Variables

The first variable selection step merely removes irrelevant variables. These are the variables that have either no relationship with the target variable or are improper to include in the models. The latter steps are usually based on regulatory concerns, such as removing race or ZIP code from the analysis. They can also include removing variables that cannot be replicated or are too expensive to collect on an ongoing basis; a company may have done a study that involved extensive interviews with their customers, but unless this can be done for all customers in the future, the variables cannot be used as inputs to a predictive model.

Other irrelevant variables include labels, such as customer ID. While there may be some predictive information possible in an ID—the order of assignment of IDs may indicate how long the customer has been a customer—there are usually more direct ways to represent any information they convey, and therefore they are routinely excluded. Finally, other variables that could be collected and used as candidate inputs have no known relationship with the target and are normally excluded. The phase of the moon is always known and *could* be included in the modeling data, but there is no known causal relationship between phase of the moon and someone's propensity to respond to a marketing campaign.

Removing variables with no possible connection to the target prevents algorithms from discovering happenstance relationships in the data—spurious correlations—as described in Chapter 3. Spurious correlations are largely a small-data phenomenon, but not completely. Even with larger data, if you search through enough variables, you may eventually happen upon a variable that predicts the outcome even though it is unrelated, sometimes described as *over-searching*. Over-searching may be one reason many published studies cannot be replicated even if they are shown to be statistically significant by seasoned modelers using good practices. The particularly dangerous part of over-searching for predictive modelers is that you often include variables in your analyses that you don't suspect will be good predictors, but "why not give it a shot?" Thus you should make sure that all input variables could have a real relationship with the target variable.

Removing Redundant Variables

Redundant variables, at a minimum, waste time in the modeling process, and at worst can cause model instability. They are variables whose information is conveyed by one or more other variables, and can occur with both numeric and categorical variables.

Correlation matrices (introduced in Chapter 3) identify highly correlated variables that are good candidates for removal. When a pair of variables have a high correlation, higher than 0.9 or smaller than –0.9, they are largely redundant and only one of the two is needed for use in the predictive models. The 0.9 threshold is a rule of thumb and should not be considered an absolute standard; depending on how aggressively the modeler would like to remove variables, the threshold could be reduced to a magnitude of 0.8 or even 0.7.

In Table 4-13, if the threshold magnitude of 0.9 is used, CARDPROM and NUMPROM are considered redundant predictors and only one or the other is needed: you can remove either CARDPROM or NUMPROM without removing information in the data. The analysis of correlation matrices is a manual process, though it is possible to automate it, keeping the first in your list of variables with correlations exceeding a threshold. Some software packages include functions or nodes specifically for this purpose.

Table 4-13: Correlations for Six Variables from the KDD Cup 98 Data

CORRELATIONS	CARDPROM	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT
CARDPROM	1.000	0.949	0.550	0.023	–0.059
NUMPROM	0.949	1.000	0.624	0.066	–0.024
RAMNTALL	0.550	0.624	1.000	0.557	0.324
MAXRAMNT	0.023	0.066	0.557	1.000	0.563
LASTGIFT	–0.059	–0.024	0.324	0.563	1.000

Selecting Variables When There Are Too Many

Many predictive modeling projects include hundreds of candidate input variables as a part of the analysis, including original variables and new features created to improve the predictive models. The inclusion of hundreds of variables as candidates for predictive models can cause problems, however:

- Some algorithms cannot reliably use hundreds or thousands of input variables.
- Algorithms that can reliably incorporate hundreds or thousands of variables as candidate inputs or actual inputs to models may take considerable time to train, slowing the iterative process of building and selecting good models.

- Hundreds or thousands of input variables in a model could cause a problem related to the *curse of dimensionality*, described later in this chapter.
- The size of the data with so many inputs may exceed the memory capacity of the computer.
- Some models must be built very quickly, leaving little time for extensive variable selection and variable assessment.

For these reasons, predictive modelers often perform variable selection prior to modeling to make the modeling process more efficient with, one hopes, minimal loss in model accuracy.

The simplest way to select variables for predictive modeling when there are too many is to first run an assessment to score the predictive power for each variable in predicting the target variable, one at a time. Single-variable models or statistical tests are fast and simple to do, and usually it is the case that variables that are poor predictors on their own are not good predictors even in combination with other inputs. This is not the case with Simpson's Paradox, as described in Chapter 3, so single-variable selection comes with risks.

Nevertheless, sometimes there is little else one can do but reduce the number of candidate inputs. Table 4-14 shows a list of techniques that can be used for variable selection. This is not an exhaustive list, and you can easily adapt the principle of assessing single variables to a wide variety of statistical tests and modeling algorithms.

Table 4-14: A Short List of Single-Variable Selection Techniques

TECHNIQUE	COMPARISON TYPE	INCLUSION METRIC
Chi-square test	Categorical input vs. categorical target	*p* value or top *N* variables
CHAID tree stump using Chi-square test	Continuous or categorical input vs. continuous or categorical target	*p* value or top *N* variables
Association Rules confidence, 1 antecedent	Categorical input vs. categorical target	Confidence, Support
ANOVA	Continuous input vs. categorical target	*p* value, top *N* variables
Kolmogorov-Smirnov (K-S) Distance, two sample test	Continuous input vs. continuous target	K-S test critical value, top *N* variables
Linear regression forward selection (1 step)	Continuous input (or dummy) vs. continuous target	*p* value, AIC, MDL, top *N* variables
Principle Component Analysis (select top loader for each PC)	Continuous input vs. continuous target	Top *N* variables

Each technique works for different comparison types, which, for predictive modeling, means the tests work for different combinations of continuous or categorical inputs and targets. There are also advantages to using several of these techniques and examining which input variables hove strong relationships to the target variables. You can include input variables that pass all of the tests or input variables that pass any one of the tests, depending on how many inputs pass the tests.

Consider the KDD Cup 1998 data, with nearly 500 inputs in the modeling data. After applying an ANOVA with TARGET_B as the grouping variable, Table 4-15 shows the top 15 variables in the list, sorted by the *F*-statistic. The obvious winner is TARGET_D, though this is related directly to TARGET_B. If TARGET_D = 0 then TARGET_B = 0 as well. If there is a variable that has a surprisingly large *F*-statistic, you should investigate the variable to make sure the target variable is not leaking into the definition of the input.

Table 4-15: Variable Selection Test Using the *F*-statistic

VARIABLE	*F* STATISTIC	*P* VALUE	MEAN TARGET_B = 0	MEAN TARGET_B = 1
TARGET_D	142,779	0	0.00	15.62
RFA_2F	501.52	0	1.89	2.25
CARDGIFT	279.31	0	5.00	6.12
NGIFTALL	247.79	0	9.50	11.48
LASTDATE	161.89	0	9547.63	9556.86
LASTGIFT	120.57	0	17.43	15.17
NUMPROM	105.04	0	46.80	50.27
FISTDATE	102.13	0	9138.07	9090.34
CARDPROM	100.68	0	18.37	19.64
AVGGIFT	100.53	0	13.43	11.84
MINRAMNT	91.82	0	8.00	6.76
RAMNT_16	72.26	0	14.17	11.73
MAXRDATE	66.85	3.33E-16	9442.92	9422.08
HV2	62.61	2.55E-15	1127.42	1237.94
HV1	59.72	1.10E-14	1056.39	1163.70

Of the 404 variables included in the full ANOVA table, 198 of them have a *p* value less than 0.05. There is no magic to this cutoff; frequently, you may use a cutoff related to selection of the top 50 variables rather than using a specific *p* value of *F* statistic.

Selecting Variable Interactions

Single variable selection is fast, easy, and usually effective at retaining the best predictors. There are situations where the selection techniques can be fooled, resulting in the removal of variables that, while not predicting well on their own, predict very well when interacting with one or more other variables. This is precisely why variable selection before building predictive models is to be done only when necessary for time or memory efficiency reason.

You can accomplish a second stage of variable selection by identifying highly predictive interactions. The number of interactions to test can become quite large, however, as shown in Table 3-12 and reproduced in part in Table 4-16. It is not unusual to have more than 500 candidate input variables, particularly after building features. A typical approach to searching for good interactions is through exhaustive searching in a factorial design, sometimes available in software. But trying 124,750 combinations may be impractical and the software often returns only the best interaction of these rather than listing all interactions.

Table 4-16: Number of Two-Way Interaction Combinations

NUMBER OF VARIABLES	NUMBER OF POSSIBLE TWO-WAY INTERACTIONS
5	10
10	45
50	1,225
100	4,950
500	124,750
1000	499,500

The only quick, automated way to assess all interactions efficiently and report the predictive accuracy of all of the interactions in most predictive analytics software available today is through the use of association rules. They work only with categorical variables, so any continuous variables must first be binned, but association rules are fast, efficient ways to find interesting interactions.

In summary, variable creation and selection follow these steps:

1. Clean single variables from problems such as incorrect values, missing values, and outliers.
2. Create new features from existing variables.
3. Reduce the number of candidate inputs by removing redundant and irrelevant variables and features.

Sampling

The biggest danger in predictive modeling stability and accuracy when models are deployed is overfitting the models. *Overfit* refers to models that are accurate on the data used to train the models, but are inherently too complex and therefore perform much worse on data not in the modeling set.

Figure 4-12 shows conceptually what happens with models that lead to overfit. Two target variable values are plotted in the figure represented with filled triangles and filled circles. On testing data, data not used in defining the model and decision boundary, the two target values are represented as unfilled shapes. In the left-hand plot, the model is a simple linear model so the decision boundary is also linear. On training data, only one point is misclassified: the filled triangle to the right of the line. Testing data overlaid on top of the plot, two triangles and two circles, are shown as well, and all four are classified correctly.

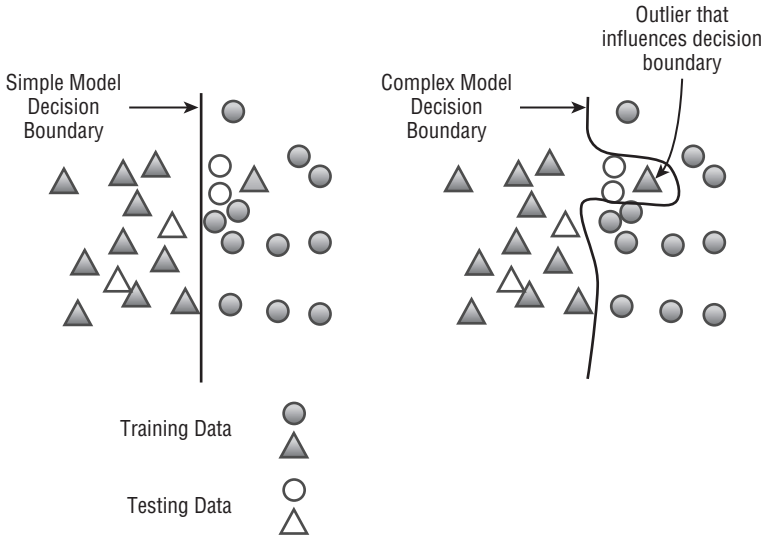


Figure 4-12: Example of overfitting data

Next, consider the plot on the right. This model is non-linear, with additional complexity introduced so that the filled triangle, misclassified in the linear model, is now classified correctly in the training data. The evidence that the algorithm has introduced additional complexity is the curved decision boundary that enables the model to achieve higher classification accuracy on the training data. However, the testing data tells a different story; both unfilled circles in the testing data are now misclassified in the class represented by the

triangles, to the left of the decision boundary. What happened? There was not sufficient evidence in the data to justify the additional complexity in the model, and sometimes it is difficult to determine whether or not the model complexity is too high without considering additional data. This is the role of sampling in building predictive models.

Partitioning Data

The most common sampling strategy is splitting the data into training and testing subsets. Some describe the testing subset as validation data or evaluation data, but the principle is the same: One builds the model on the training data and then assesses the model on the held-out, testing subset.

Training and testing subsets are usually created through a random selection of records without replacement; each record belongs to one and only one subset.

One way to think of the roles of sampling into training and testing subsets is to consider what you do when you are preparing to take a standardized test such as the GRE. To prepare for the exam, assume that you go to the store and buy a test book containing sample questions and an answer key in the back. After taking the practice exam, you grade yourself with the answers in the back of the book. Assume that after the first attempt, you have only 65 percent correct, indicating that you need to study a lot more to do better. So you study the material covered in the test harder and try again. This time, you get 82 percent correct: good, but not good enough. So you go back and study the material even harder. After several passes through testing and studying, you finally have a grade of 98 percent. After feeling quite good about yourself and how well you did on the test, a nagging feeling comes over you: Do you really know the material, or did you just memorize the answers to these particular questions?

One way to answer that question is to go to the store and buy a second book, the *test* or *validation* book. You haven't seen these questions before, so if you get 98 percent correct on this book, you can have a much higher degree of confidence that you really know the material. However, if you get only 65 percent correct on this book after getting 98 percent correct on the first book's test, you know that you simply memorized the answers to the questions in that book, and don't really know or understand the material.

Sampling is a lot like this. Predictive models can do very well on the data that was used to build the model. But the model could be too specific to those particular records, especially if there is noise in the data that may cause small differences in values of some inputs that the model can pick up and use to differentiate values of the target variable. This effect is sometimes even called *memorizing* the data.

Measuring model performance on data the algorithm didn't use in building the model is the best way to identify overfit. If the performance on the testing data is about the same as the performance on the training data, you have strong evidence that the model is not overfit. The typical process for building models therefore is this:

1. Build a model on training data.
2. Assess the model on testing data.
3. Change settings, parameters, or inputs to the model.
4. Re-build the model on training data.
5. Assess this new model on testing data.
6. Change settings, and repeat the process.

This iterative process could be done manually by a practitioner or automatically by the modeling software. Either way, you could iterate dozens, hundreds, or even thousands of times through the process.

Splitting data into these two data sets, training and testing, has a long history in statistical modeling. However, an additional nagging question should emerge. Because the testing data is informing the modeler or software on how to adjust the modeling algorithm parameters so that model performance on testing data is improved, is the testing data truly an independent measure of model accuracy? The answer is a definitive "maybe." It is certainly possible that the testing data becomes a driver for how the algorithm builds the model, leading to overfitting not based on the training data, but indirectly based on the testing data.

One solution to this potential problem is to have a third data set available, held out until the very end of the model assessment stage. This data, called *validation data* (as a part of the training, testing, validation labels for the three modeling data sets), is used to provide a final estimate of the predicted model performance or accuracy once it is deployed.

Each of these three sampled subsets of the entire data set must be large enough to be representative samples, but are separate from one another: Each record belongs to one and only one of the three subsets. Table 4-17 shows 20 records from a data set with each record labeled with the sampling subset to which it belongs: training, testing, or validation.

The idea of partitioning data into three subsets has been widely adopted by the software vendors; most predictive analytics software includes functions to split data into all three of these subsets. For those software packages that still only allow for partitioning into training and testing, you can include an additional split of the testing subset to create the testing and validation subsets.

Table 4-17: Sample Split into Sample Subsets

SAMPLE SUBSET	CONTROLN	STATE	AGE	GENDER	LASTGIFT
Training	24610	SC	81	M	3
Testing	118673	TX		F	10
Testing	82943	MN	62	F	5
Validation	73371	MI	79		10
Training	190313	TX		M	7
Training	76585	IA	66	F	5
Validation	156378	CA	69	F	10
Testing	25641	GA	73	M	12
Training	123822	TN		F	9
Training	31911	VA		F	5

The Curse of Dimensionality

Distance between data points is a key concept in building many modeling algorithms, including k-nearest neighbor and K-Means clustering models. Even if distances are not explicitly computed in the modeling algorithms, at its root, predictive modeling tries to find commonalities between values of variables, and in the case of supervised learning, the algorithms relate these values to a target variable. To find these patterns in the data, there must be enough of them to find commonalities and differences.

One way to appreciate potential problems with distances is to consider what happens to distances as the number of variables in a clustering model increase. What if you have 10 volleyballs in a room that is 20 feet by 20 feet and they were spread randomly throughout the room. How far apart are the balls? On average, they may be about 10 feet from any other ball. (Keep this measurement in mind.) But from a clustering standpoint, the more apt question is this: How far away is the nearest ball to each of the balls? It turns out that with 10 volleyballs in the room, it will be about 3 feet.

What happens if instead of the 10 volleyballs being thrown and landing on the ground in the two-dimensional area of the floor, you could suspend the laws of physics and allow the same 10 volleyballs to end up anywhere in the room, including in the air all the way up to the ceiling. Furthermore, assume that the ceiling is 20 feet high (the same size as the other two dimensions). How far apart is the smallest distance from each ball to its nearest neighbor now? It turns out that the average minimum distance may now *double* to more than 6 feet, depending on the random location of the balls.

These two spatial dimensions can be thought of as two inputs to a model, and the floor of the room like the canvas of a scatterplot (see Figure 4-13). In this plot, the ten randomly distributed data points are on average 10 units apart.

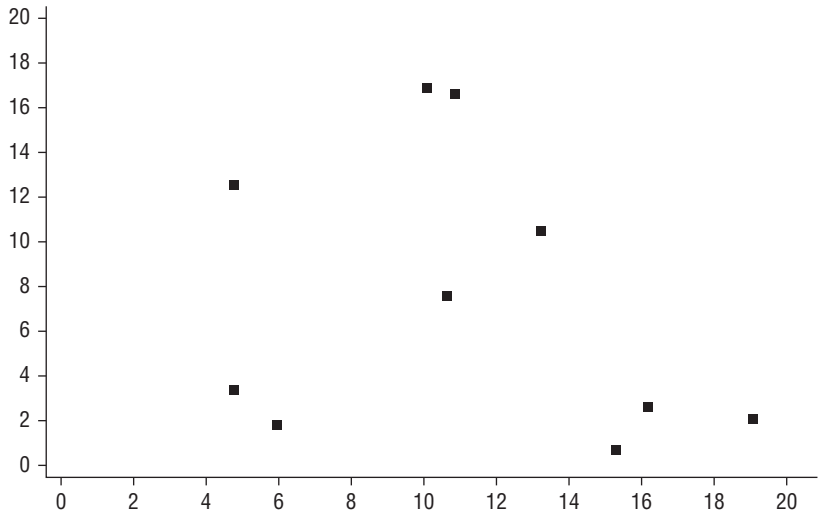


Figure 4-13: Random spread of data in 2D

If we could possibly imagine higher dimensional space, say four or five dimensions rather than just the three spatial dimensions, it would be a different story; the volleyballs will be further apart because of the exponentially increasing space to fill due to the higher dimensions. Or in a predictive analytics framework, four or five inputs rather than just two or three means a lot more empty space in the scatterplot.

Now consider Figure 4-14. With only two inputs and 10 data points (the point at the bottom left of the figure), the average minimum distance to the nearest data point is just under three units away. As you increase the number of inputs to five, this number rises to more than eleven units. Or think of the data in a different way: How many data points would you need to maintain the same minimum distance to the nearest point as you increase the number of inputs of the data? It turns out that to achieve the same distance between data points (about three) as the data in Figure 4-13 when you have three inputs, you need between 100 data points for three inputs and 500 data points in four inputs. With five inputs, even 1000 data points is not enough. In other words, as the number of inputs increases, the number of data points needed to fill the space comparably increases exponentially. This, in a nutshell, is the *curse of dimensionality*.

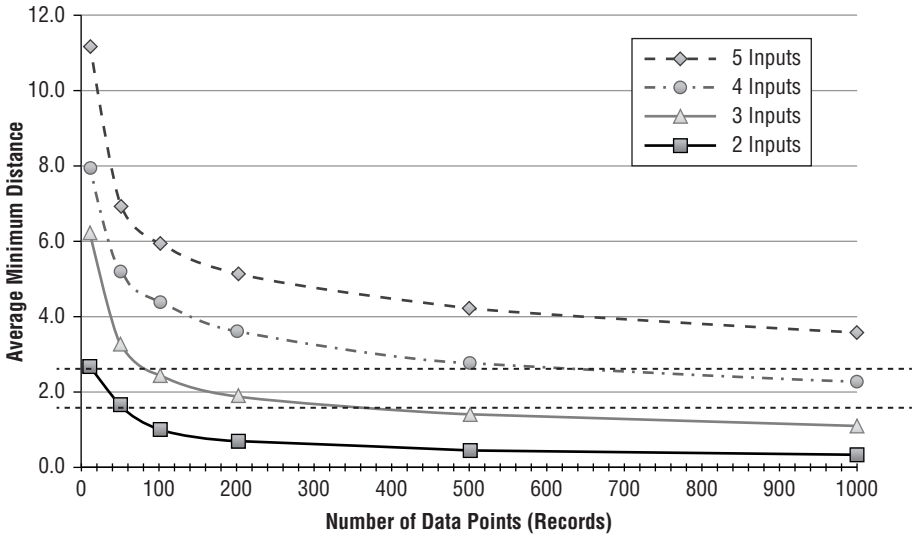


Figure 4-14: Increasing number of data points needed to populate data

If a sufficient number of records are not included in the data set, the data space is not populated densely enough and individual data points become further and further away from each other, appearing more like outliers than patterns of data.

In reality, the problem isn't as bad as described here; the data used for this experiment was randomly generated, and presumably, the patterns that exist in the data are not random and will clump in different regions of the data space. However, the curse of dimensionality provides a principle to keep in mind when determining how many records are needed for effective modeling: As you increase the number of candidate inputs for modeling, the number of records needed to capture the higher-dimensional patterns must increase, and certainly much more than linearly.

Rules of Thumb for Determining Data Size

All practitioners know that one needs *enough* records in the data to build reliable models. The actual number of records, however, is not easy to determine precisely and depends on patterns found in the data itself, not just how many inputs may be considered in the models. For example, if the patterns in the data are linear, and there is little noise in the data, you need only one or two records per input in the model, so merely dozens of records would be needed for modeling.

Unfortunately, few problems have these characteristics: linear and low-noise. The more nonlinear the patterns in the data, the more data needed to characterize

the patterns. And the more noise in the data, the more data needed to overcome the noise. Noise in this context means unobserved relationships in the data, not captured by the inputs. It appears as random fluctuations that models never can capture (unless they are overfit!).

There are two primary considerations for determining sample size. First, you must consider the inherent dimensionality of the data and the effect of the curse of dimensionality on the data. For smaller problems when you are expecting to consider only a few variables, 10–20 records per input may be sufficient. For more complex problems, at least 100 records per input may be needed to capture the multidimensional patterns in the data. Therefore, for models with 100 candidate inputs, as a general rule, you would want to have a minimum of 10,000 records in the data for each subset (training, testing, and validation).

The second consideration is of particular concern with classification problems. Not only do you need to have sufficiently populated inputs, but you need to have the target variable sufficiently populated. For each level in the target variable, as a rule of thumb, you would want to have at least 1,000 examples for each of the training, testing, and validation data sets. This is typically only a problem for the most interesting values: responders for customer acquisition models, fraud cases for fraud detection, churners for customer attrition models, and so forth. The under-represented class is the one that is in the most danger of having too few records for building stable models.

These rules of thumb can be used in two ways. First, they can help determine if there is enough data available for modeling. If there are not enough records to have at least training and testing subsets in sufficient numbers, you may have to consider other resampling techniques to ensure that stable models are being built.

If you have, from a practical standpoint, limitless data, these rules of thumb can also be used to reduce the size of the data in building models. From a practical standpoint, data size is often a limiting factor for the modeler for most algorithms; the time it takes to build models increases more than linearly with the number of records and the number of inputs. Iterating over training and testing of models to find good parameters and inputs for the models is a critical part of the modeling building process, so using smaller data helps the modeler build models more efficiently. If you have 10 million records, but the rule of thumb indicates that you need only 50,000 records, most of these records are not needed in the training set. If you need to reduce the time to build the models, most of these records would not be needed to include in the training and testing subsets. Of course if the hardware and software handles the 10 million records efficiently and without problems, there is no harm in using all the data.

Partitioning Data: How Much Data to Include in the Subsets

Once the sample size is determined, you must also determine how much data should be included in each of the training, testing, and validation subsets. Most software packages follow the approach taken historically as their defaults, either 50 percent of the data used for training and 50 percent for testing, or on some occasions, two-thirds of the data used for training and one-third for testing.

However these rules of thumb do not capture the key issue. Statistical significance and stability are determined not by percentages but by *N*: the number of records. If the number of records is just right, the percentages are fine to use. But if the there are too few records, no partitioning will be good: Each subset will have too few records. When there is more data than is needed for modeling, splitting by percentage will only include more data than is necessary for building good models in each subset.

Consider a data set with 1,000,000 records, and assume you have determined that 100,000 records are needed to build an effective model. Table 4-18 shows a summary of sample sizes based on the percentage defaults compared with a record-based default. If 50 percent of the records were selected for the training set, 500,000 records would be used compared with 100,000 for the record-based selection method. If 100,000 records are needed in the training data to build stable models, you should have 100,000 in the testing data and a minimum of 100,000 in the validation data. If only 100,000 are needed for training and testing, however, the remaining 800,000 can be used for validation to provide the final estimate of model performance once it is deployed.

Table 4-18: Sample Size of Partitioned Data Based on Rules of Thumb

PARTITION	RULE OF THUMB BASED ON PERCENTAGE OF RECORDS, PERCENT	SAMPLE SIZE FOR RULE OF THUMB BASED ON PERCENTAGE OF RECORDS	SAMPLE SIZE FOR RULE OF THUMB BASED ON SUFFICIENT NUMBER OF RECORDS
Training	50	500,000	100,000
Testing	25	250,000	100,000
Validation	25	250,000	800,000

Cross-Validation

Cross-validation is a sampling technique used primarily for small data sets, when data is too small to partition into training and testing subsets. The advantage of cross-validation is that all of the data is used in building the models, so all of the patterns represented in the training data help form the models.

Cross-validation is usually described as *k-fold cross-validation*, where the k refers to how many subsets are used for modeling. The procedure for k-fold cross-validation is as follows:

1. Create k distinct data subsets through random sampling.
2. Assign a role to each subset. k–1 of them will be used for training, 1 for testing. Begin by assigning subset 1 to testing, and subsets 2 through k for training.
3. Rotate roles so each subset is used for testing one time and training k–1 times.

Figure 4-15 shows a visualization of cross-validation.

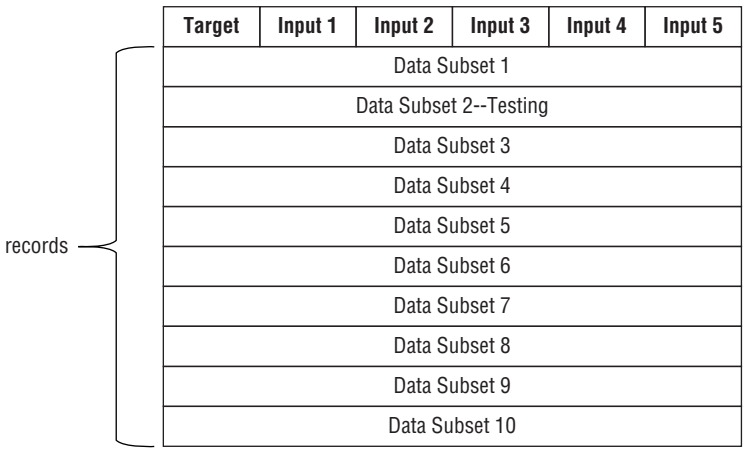


Figure 4-15: Cross validation sampling

A type value for k is 10, yielding 10-fold cross-validation. However, any number of folds can be used. The more folds you use, however, the smaller the held-out testing subset is, and the more error variance you will observe for the held-out data. As you increase the number of folds, the size of each fold becomes smaller and smaller until you have only one record in each fold. This is a special case of cross-validation sampling sometimes called "hold-one-out" sampling. The error for each fold isn't interesting, but rather the average error over each fold is what you compute to assess the models.

Statisticians use cross-validation to assess model stability. A model is built from each of the folds—a total of k models. The hold-out subset is used to test the model, and the errors are averaged over the subsets. The model accuracy on the k testing subsets can be used to assess how stable the models are: If the accuracy is similar for all folds, the modeling procedure is viewed as being stable and not overfit. This average error can be used as an estimate of how accurate a model will be on new data.

Figure 4-16 shows the average error from a linear regression model training using 10-fold cross-validation. Most folds demonstrate consistent behavior, which suggests that stable models have been built. Fold 7, and to a lesser degree Fold 5, has a much larger average error. Table 4-19 shows why Folds 5 and 7 have large average errors. This table shows the largest absolute errors in the testing data, and clearly there are outliers in the predictions that fall into these two folds; the evidence shows that the large cross-validation errors were due to outliers in the data rather than unstable models.

If the k models are all stable, a final model of comparable complexity to the models built from the folds can be built using all the data, and the modeler expects that the model will exhibit the problems due to overfitting.

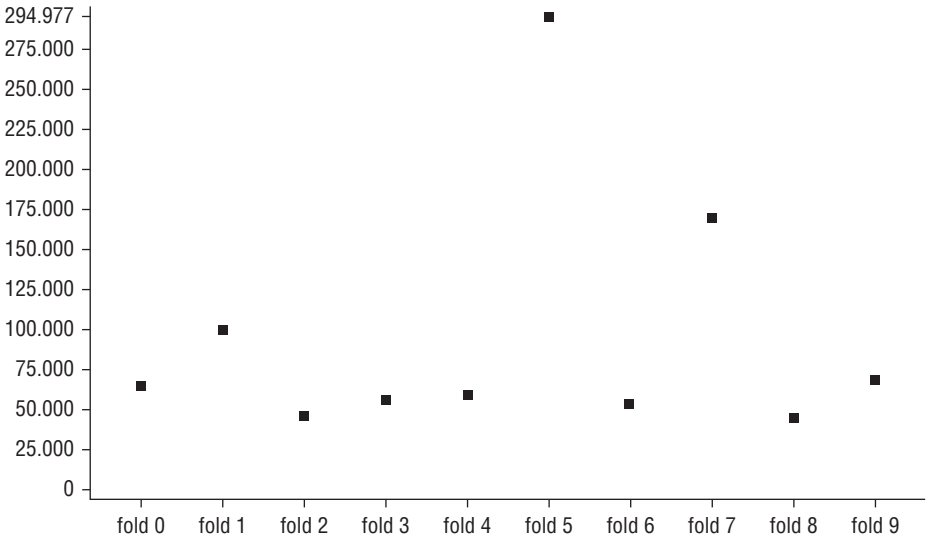


Figure 4-16: Errors from cross-validation out-of-sample data

Table 4-19: Largest Errors in 10-Fold Cross-Validation Models

FOLD	ACTUAL	PREDICTED	ABSOLUTE ERROR
5	20	329.1	309.1
7	200	20.0	180.0
1	200	37.1	162.9
7	150	21.9	128.1
5	100	9.8	90.2
9	100	19.8	80.2
6	100	20.3	79.7
5	200	121.5	78.5

Bootstrap Sampling

Another approach to building robust models on small data is the use of bootstrap sampling. The bootstrap sampling method samples the data *with replacement* in contrast to the random sampling methods described so far. For an illustration of the concept, consider a big ping pong ball hopper for picking lottery numbers. Each ping pong ball has a number on it representing a record number in the data to be sampled from. Assume there are 1,000 records in the data set, and the first ping pong ball that is selected is #143. Record 143 is then included in the bootstrap data set. Then that ping pong ball is returned to the hopper, or, in other words, the ping pong #143 is replaced into the hopper and is therefore eligible to be selected again. Now assume ball #879 is selected. Record 879 is now put into the bootstrap data set, and the ball is returned to the hopper so it is eligible to be selected again. Next, assume ball #143 emerged again. That record would be placed in the bootstrap data set for a second time. This process continues until, typically, the same number of records are selected for the bootstrap data set as were in the original data set.

Table 4-20 shows a small sample of 10 records and 3 bootstrap samples from the same 10-record data. In each case, only 6 or 7 of the original 10 records were included in the bootstrap sample. Moreover, some records were included multiple times (records 4, 7, and 10 in bootstrap sample 1), and some records not at all (records 1, 3, and 5 in bootstrap sample 1).

Table 4-20: Example of the Bootstrap Sampling Method.

ROW NUMBER	BOOTSTRAP SAMPLE 1	BOOTSTRAP SAMPLE 2	BOOTSTRAP SAMPLE 3
1	8	1	7
2	7	6	7
3	4	4	5
4	7	2	8
5	4	3	8
6	2	3	1
7	6	6	3
8	10	3	9
9	10	9	1
10	9	10	1
Summary Notes	Seven of ten records in sample No samples from 1, 3, 5 Two samples from 4, 7, 10	Seven of ten records in sample No samples from 5, 7, 8 Two samples from 6 Three samples from 3	Six of ten records in sample No samples from 2, 4, 6, 10 Two samples from 7, 8 Three samples from 1

You may ask at this point, "Why bootstrap sample at all? There are too many of some records and not enough of others?" The answer is that you don't sample only once or twice, but dozens or hundreds of times. Each sample is drawn from the original data but shifts the distribution randomly by some amount, providing a different emphasis in the data.

On average $1 \div e$ (or about 63 percent) of the records will be selected for inclusion in the bootstrap sample. This leaves a 37 percent hold-out sample for each bootstrap sample. One advantage bootstrapping has over cross-validation is that no matter how many samples one makes, there is always a 37 percent hold-out sample available to assess the models.

After creating the bootstrap samples, the procedure is the same as was done for cross-validation. One model is built for each sample and is assessed to ensure stability, consistency, and robustness.

Temporal Considerations in Sampling

So far, sampling descriptions have focused on creating unbiased data sets of sufficient size to avoid model overfit. The implicit assumption is that the time period of the historic data is irrelevant: Modelers assume future relationships between observed behavior, and the target variable will be consistent with the historical patterns stored in the data. For example, if you are predicting whether a donor to a non-profit will donate again, patterns such as how many donations the donor made (at any time in the past) and the amount the donor donated (at any time in the past) are all you need to predict future donation behavior. Put another way, the data does not expose the sequence of historic giving in order to predict if the donor will donate again.

However, some modeling problems have additional constraints that should be considered, such as data that is *non-stationary*. Non-stationary data changes behavior with time and therefore should be reflected in the modeling data and sampling strategies. For this kind of data, simple random sampling is not appropriate: It will inflate the expected accuracy of the model.

This phenomenon is similar to the difference between interpolation and extrapolation. Interpolation refers to estimating values that are bracketed by other values; predictive modeling usually involves interpolation, such as the prediction shown in Figure 4-17. The model provides the connection between historic data. Predictions for unseen data are assumed to be consistent with the model, falling in between data points observed historically in the data.

Extrapolation, on the other hand (see Figure 4-18), extends beyond what has been seen historically. These predictions are usually more difficult, and often less accurate, because the data available for predictions is limited to one side of the curve fit.

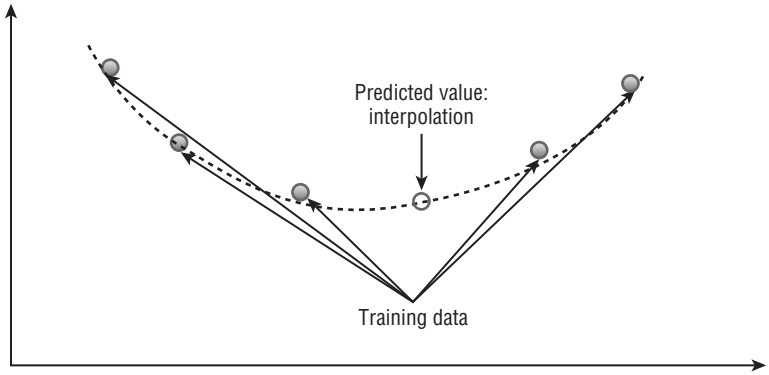


Figure 4-17: Interpolation of data points

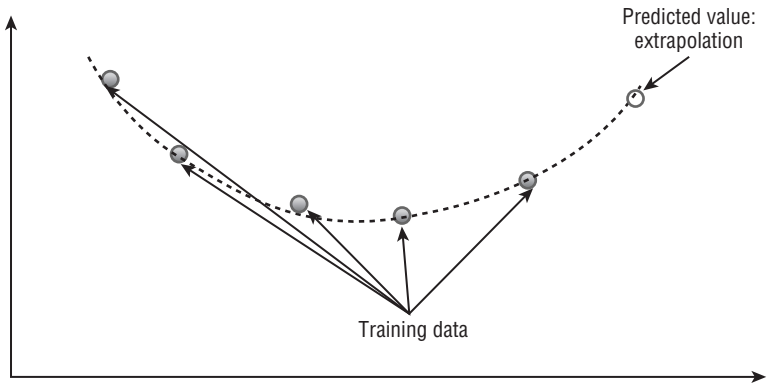


Figure 4-18: Extrapolation of data points

When temporal sequencing is important in predictions, some modifications in sampling need to be incorporated. Consider predicting data from a time series, such as stock market future value prediction. Training and validation data used for predictive modeling should be set up in chunks: training on one set of data and validating on data future to the data used in training. It is inherently an extrapolation problem. Figure 4-19 shows a representation of the idea. This figure is an extension of Figure 2-2 with changes made to the left of "now." If data were split into training and validation subsets randomly from any point in time, the validation data would measure the interpolation capabilities of the models in a way that is not representative of how the model will be used. It will be overly optimistic in estimates of accuracy.

Notice that testing data falls in the same temporal group as training data. Testing data in this context is used to avoid overfitting, whereas validation data is used to assess expected accuracy of the models when deployed. This is

the case with many algorithms: Temporal separation of training and testing data is difficult to achieve.

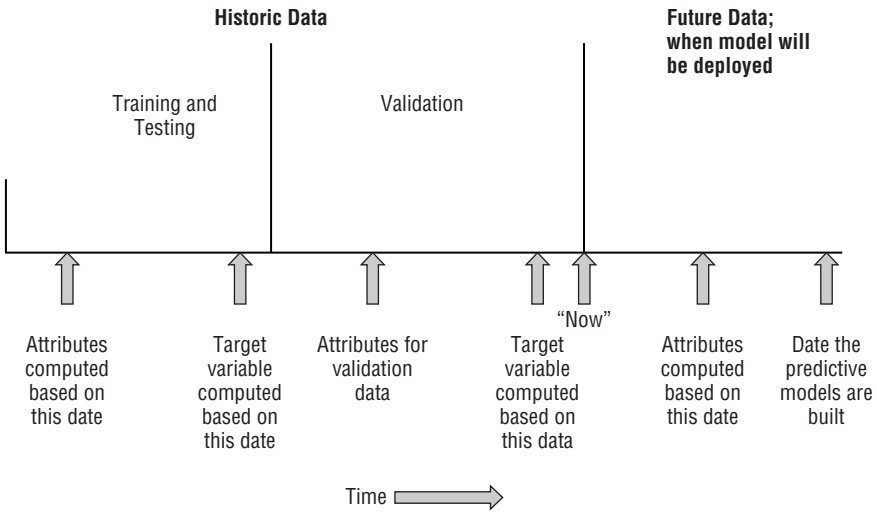


Figure 4-19: Sampling for temporal data

The key principle at work is to sample in a way that helps measure the accuracy of predictive models in a manner consistent with how the models will be deployed.

Stratified Sampling

Stratified sampling balances counts based on one or more variables. The purpose is to remove the bias that may result from a sample having more records of one value of a variable of interest. For example, you may want to stratify a customer behavior data set to ensure males or females are represented equally, or the income quantiles of the customers so each quantile is represented equally. If historic data is representative of the expected distributions of variables that will be included in models, stratification of inputs is unnecessary. For sample sizes that are large enough, random sampling is unlikely to produce biased samples where key variables are distributed in unnatural ways.

The most common reason predictive modelers stratify samples is related to the target variable rather than input variables. The reasoning usually follows this line of thinking: Suppose a modeling group is building a classification model to predict whether or not customers will respond to a mailing. Furthermore, suppose the response rate of the historic data to be used in building the model is 5 percent. The modeling team built a model and assessed it, finding that it was

95 percent accurate. Upon further examination, they found that the reason that model was 95 percent accurate was that it predicted every customer would not respond to the mailing. So the good news is that the model has high accuracy (95 percent). The bad news is that the model tells the business not to mail to anyone because this is the best way to maximize accuracy.

As a result of this analysis, the modeling team decided the reason the models predicted every customer would not respond to the mailing was that the sample was biased. Because most of the records by far were non-responders, the algorithms focused on the non-responders, learning those patterns better. The solution was to break up this bias by stratifying the sample so that the modeling population is 50 percent 1s and 50 percent 0s.

This approach is perfectly fine statistically and can work well. But was it necessary? The answer is usually "no," and in addition, stratifying can lead to models that are worse than those built from the natural population. Often, these models that appear to be predicting every customer to be a non-responder are actually working very well; it is only the interpretation of the model that causes problems for the modeler.

Consider the example again with the 5 percent historic response rate. The model assessment stated that all of the customers were predicted as non-responders, but then the modelers decided to look at a ROC curve as well. The ROC curve for two models is shown in Figure 4-20. Examining the ROC curve tells a different story than the classification accuracy: Nothing appears wrong in the ROC curve at all.

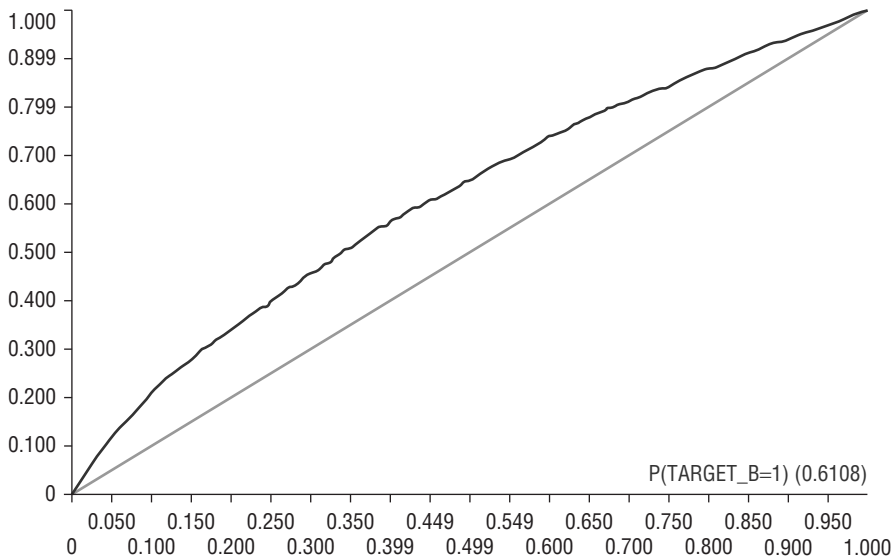


Figure 4-20: ROC curve for non-stratified sample

The question therefore is this: "How can a ROC curve look fine, but the classification accuracy of a model be so bad?"

First, classification algorithms do not generate classification decisions. In binary classification problems, if the two class values of the target variable are 0 and 1, the classification model does not predict a 0 or a 1. The model predicts the likelihood that the customer belongs to the class 0 or the class 1. The actual distribution of the predicted probabilities is shown in Figure 4-21. The peak of the histogram is near the value 0.05, which is expected since the percentage of target variables equal to 1 is 5 percent (0.05). There is clearly a range of predictive values, which is why the ROC curve looked good. The model is not degenerate, predicting 0s for all records.

Note that the largest value of the predicted probabilities is only 0.35, less than the value 0.5. This is important because every predictive modeling software package thresholds the classifier predictions by 0.5: If the predicted probability exceeds 0.5, the record is given the predicted label equal to 1; otherwise, it is given a value 0. Because no record exceeds the 0.5 threshold, all of the records are given a label of 0.

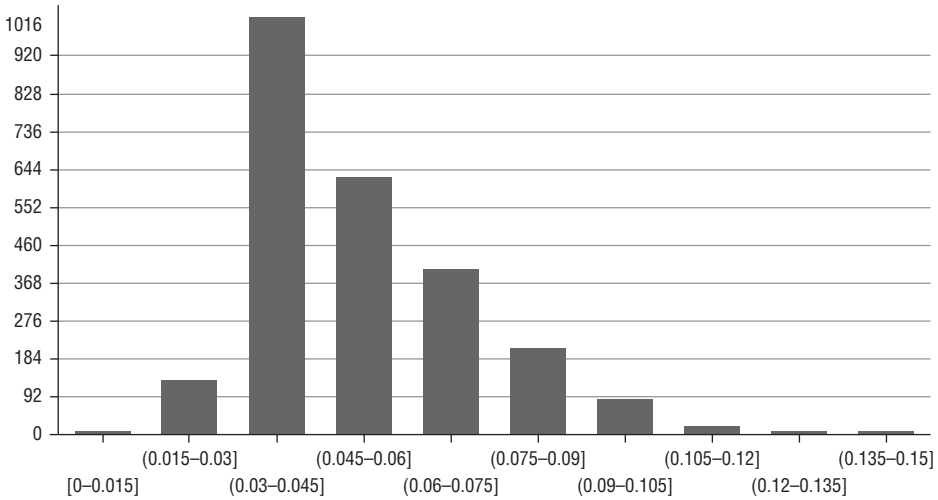


Figure 4-21: Distribution of predicted probabilities

This is exactly what is happening in the confusion matrix of Table 4-21: All records are given a predicted label of 0 because no record exceeded the 0.5 threshold. But if a threshold of 0.05 is used instead (the prior probability),

the confusion matrix looks much different, as shown in Table 4-22. This confusion matrix looks far more interesting and is evidence that the model is working.

Table 4-21: Confusion Matrix from 5 Percent Target Rate, Threshold 0.5

TARGET	1	0	TOTAL
1	0	2,399	2,399
0	0	45,307	45,307
Total	0	47,706	47,706

Table 4-22: Confusion Matrix from 5 Percent Target Rate, Threshold 0.05

TARGET	1	0	TOTAL
1	1,312	1,087	2,399
0	18,115	27,192	45,307
Total	19,427	28,279	47,706

If you stratified the sample, you would get a confusion matrix very similar to the one shown in Table 4-22. What are the dangers (if any) of just stratifying the sample based on the target variable? First, with only 2,399 1s, stratifying by removing records with target variable = 0 would remove 43,000 records, leaving 4798 total records to form decision boundaries from (instead of 47,706). Removing records when data sizes are already small can lead to less precise decision boundaries and more false alarms.

If instead you replicate records to achieve equal sized populations of the target variable, you will be building models from 95KB records, nearly half of which are replicates of the 2,399 1s (they would be replicated, on average, 19 times each in the data).

Example: Why Normalization Matters for K-Means Clustering

K-Means clustering is an unsupervised learning method that is susceptible to bias when the input data is non-normal or at least non-uniform. An example of the kinds of problems you can encounter is described here.

Let's begin by using the KDD Cup data from 1998, but only four of the variables: AGE, RFA_2F, LASTGIFT, and MAXRAMNT. All missing values

have been removed to simplify the analysis, and all AGE values less than 19 have been removed. Furthermore, only 3,684 examples are used. Table 4-23 shows the summary statistics for the four variables, and Figure 4-22 shows the box plots.

Table 4-23: Summary Statistics for Four Inputs to K-Means Model

VARIABLE	MINIMUM	MAXIMUM	MEAN	STD. DEVIATION	ROW COUNT
AGE	20	98	62.33	15.75	3,684
MAXRAMNT	5	1000	18.29	21.03	3,684
LASTGIFT	0	250	15.21	11.23	3,684
RFA_2F	1	4	2.20	1.15	3,684

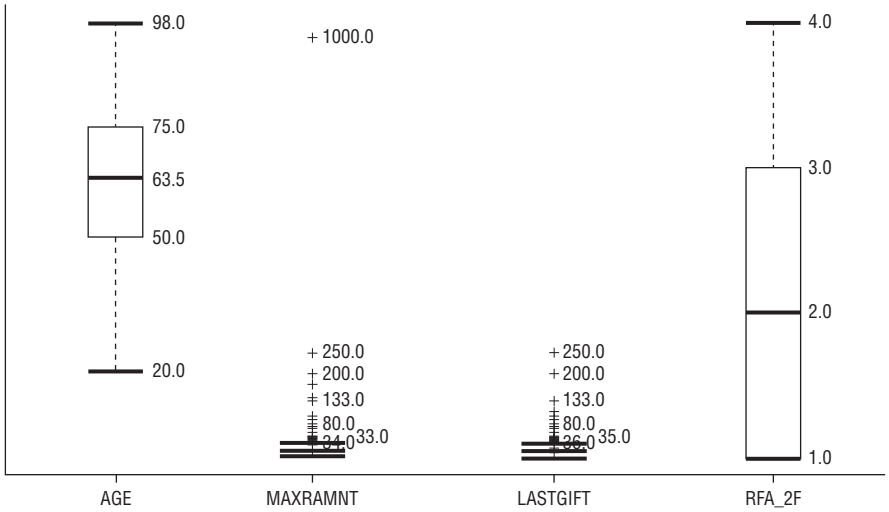


Figure 4-22: Box plots for four inputs to cluster model

A few characteristics of the data are worth noting. First, LASTGIFT and MAXRAMNT have severe positive skew. Second, LASTGIFT and MAXRAMNT have magnitudes that are much larger than RFA_2F and, to a lesser degree, AGE. However, AGE and RFA_2F are distributed without significant skew; even without showing the value of skew, you can see that the mean values from Table 4-23 are nearly the same as the median values shown in the box plots of Figure 4-22.

Five clusters were created from the data, with characteristics summarized in Table 4-24. The table has been sorted by the mean value of LASTGIFT and MAXRAMNT to show more clearly how the clustering algorithm determined cluster membership. Cluster_4 is a small cluster, having only 19 records, but with large values of LASTGIFT and MAXRAMNT; these are the outliers in the data

as shown in the box plot. In fact, the mean values for these two variables are 8 to 13 times larger than the mean values in cluster_1. Cluster_2 is also a relatively small cluster with relatively large values of LASTGIFT and MAXRAMNT.

Table 4-24: Cluster Description by Mean Values of Inputs

CLUSTER	COUNT	MEAN AGE	MEAN RFA_2F	MEAN LASTGIFT	MEAN MAXRAMNT
cluster_1	1,044	62.00	2.37	12.03	14.4
cluster_3	1,200	79.10	2.37	12.19	14.6
cluster_0	1,096	43.42	2.03	15.14	17.1
cluster_2	325	64.77	1.64	31.68	38.8
cluster_4	19	69.63	1.84	102.89	184.2

The spread of mean values for these two variables is clearly much larger than the spread for AGE and RFA_2F, indicating that these values are determining cluster membership at the expense of the other two fields.

Following the steps outlined in this chapter, taking the log transform of LASTGIFT and MAXRAMNT will reduce their positive skew. In addition, normalizing all of the variables so they have the same scale will reduce the bias introduced by the magnitude of the variables so the clustering algorithm focuses on the relative values of the variables.

Table 4-25 shows the summary statistics after normalization, and Figure 4-23 the box plot. First, note that the minimum and maximum values of each value are now 0 and 1 respectively, due to the min-max normalization. The mean values of three of the variables are between 0.4 and 0.6, roughly in the center of the distributions though not exactly so. MAXRAMNT_log10 has a mean value of 0.2, indicating that the log transform did not remove all of the skew from the distribution. If you don't like this lingering skew, you may want to remove outliers to remove this skew by removing records with MAXRAMNT greater than 1.5 times the IQR (33 for this data). You can see the remaining positive outliers more clearly in the box plot.

Table 4-25: Summary Statistics of Four Variables after Normalization

VARIABLE	MINIMUM	MAXIMUM	MEAN	STANDARD DEVIATION	ROW COUNT
AGE	0	1	0.543	0.202	3,684
RFA_2F	0	1	0.401	0.382	3,684
MAXRAMNT_log10	0	1	0.201	0.093	3,684
LASTGIFT_log10	0	1	0.476	0.100	3,684

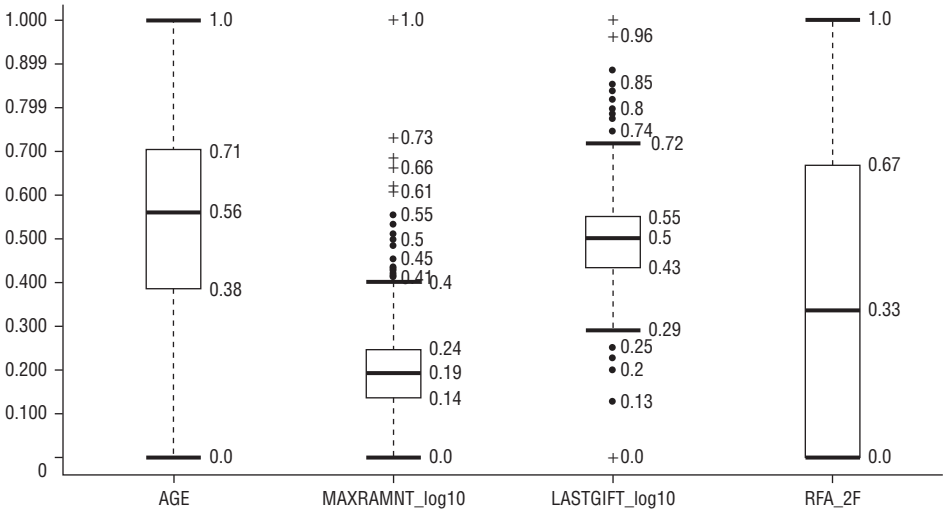


Figure 4-23: Box plots of normalized cluster inputs

After building the five clusters, the summary statistics are shown in Table 4-26. Now the ratio of the maximum to minimum mean value of LASTGIFT and MAXRAMNT is closer to 2 rather than the 8 to 13 in the un-normalized data. RFA_2F now takes a larger role in determining the clusters.

Table 4-26: Cluster Summaries for Normalized Inputs

CLUSTER	COUNT	MEAN AGE	MEAN RFA_2F	MEAN LASTGIFT	MEAN MAXRAMNT
cluster_1	490	0.670	1	0.036	0.0093
cluster_3	490	0.683	0.666	0.042	0.0081
cluster_0	467	0.314	0.827	0.044	0.0094
cluster_2	1,149	0.698	0.133	0.073	0.016
cluster_4	1,088	0.356	0.113	0.075	0.016

The change in the characteristics of the clusters can be seen even more clearly if the clusters are summarized by the variables in their original units, shown in Table 4-27.

Table 4-27: Summary of Clusters Built on Normalized Data with Inputs in Natural Units

CLUSTER	COUNT	MEAN AGE	MEAN RFA_2F	MEAN LASTGIFT	MEAN MAXRAMNT
cluster_1	490	72.30	4.00	9.06	14.27
cluster_3	490	73.27	3.00	10.44	13.10
cluster_0	467	44.48	3.48	11.06	14.39
cluster_2	1,149	74.45	1.40	18.23	21.28
cluster_4	1,088	47.77	1.34	18.72	20.95

Which cluster model is better: the one built on original variables or the one built from normalized variables? That is not an easy question to answer. The normalized model has less bias from LASTGIFT and MAXRAMNT. Clusters 2 and 4 tell an interesting story: One cluster is younger, big-dollar donors (cluster_4), and another cluster is older, big-dollar donors (cluster_2). Clusters 0, 1, and 3 all have similar LASTGIFT and MAXRAMNT values, but are differentiated by AGE and RFA_2F.

Summary

Data preparation clearly takes considerable thought and effort to ensure the data is presented to the algorithms in a way they can be used effectively. Predictive modelers need to understand how the algorithms interpret the data so they can prepare the data appropriately for the algorithm. The extensive discussion in this Chapter about how to correct for skewed distributions is not needed for decision trees, for example, but can be necessary for linear regression.

Do not consider Data Preparation a process that concludes after the first pass. This stage is often revisited once problems or deficiencies are discovered while building models. Feature creating, in particular, is iterative as you discover which kinds of features work well for the data.

Overfitting the data is perhaps the biggest reason predictive models fail when deployed. You should always take care to construct the sampling strategy well so that overfitting can be identified and models adjusted appropriately.

CHAPTER

5

Itemsets and Association Rules

This chapter describes how to build and use association rules. Association rules have not always been in the mainstream of predictive analytics and aren't always included in predictive analytics software packages even to this day. The name given to association rules in modeling software is not standardized; the algorithms will sometimes be identified as *association rules*, sometimes as *item set mining* or *frequent item set mining*, sometimes even as *market basket analysis* after a common application of association rules.

The purpose of association rules is to enumerate interesting interactions of variables. While this sounds straightforward, the combinatorial explosion prevents one from finding all of the combinations using a brute-force approach. The algorithms used to build association rules, such as the most popular algorithms Apriori, Eclat, and FP-growth, use intelligent search to limit the full range of rules that could be found by eliminating branches of the search that no longer apply. The primary difference between these algorithms is the efficiency of the algorithms on a particular data set.

Suppose a grocery store chain wants to understand its customers' purchase behavior by examining which items are purchased together in a single visit. Ignore for the time being multiple visits in the same day. There is no target variable in this model because the decision makers are not interested in specific products purchased, but *any* product that is purchased and the relationships with the purchase of any product with the co-occurring purchases of any other product.

Customers purchase a wide variety of items together. For example, the model shows that a high percentage of customers purchase bread and milk in the same transaction, so if a customer purchases bread they are highly likely to purchase milk, and *vice versa*. Milk, however, is purchased so frequently that many products exhibit the same kind of behavior: Whether a customer purchases cereal, chicken, or Limburger cheese, there is still a high likelihood that milk is in the same basket.

However, the converse is not true: If one purchases milk, only rarely is Limburger cheese in that basket, in large part because very few customers purchase Limburger cheese. Customers who purchases high-end, specialty crackers, however, purchase Limburger cheese much more frequently than the rate of purchasing Limburger cheese alone, which is low for this store. In addition, the purchasers of Limburger cheese spent more on average than the average customer, a statistic that was computed after the model was computed and wasn't a part of the association rules model. As a result of the analysis, the decision-makers kept the high-end cracker on the shelf even though its sales were not strong because of the strong association with Limburger cheese and the type of customers who purchase Limburger cheese.

The most famous association rule is the fabled "beer and diaper" association, a story that is apparently true but has been retold in a variety of forms over the decades. According to the story, a drug store was analyzing their data and discovered that between 5 p.m. and 7 p.m. on Friday there was an unusually high association between beer purchases and diaper purchases. Aside from the amusing aspect of the story, it is a particularly good example of an algorithm finding unexpected patterns in the data that no analysts would have hypothesized as a potential.

It is precisely these kinds of interactions—beer and diapers, or specific crackers with high-valued cheese—that association rules are particularly adept at finding. They can find not only infrequent but unusual patterns, but also the most frequent patterns in the data.

Terminology

To help with the terminology of association rules, let's consider a simple supermarket example, called *market basket analysis*.

Consider the simple data set in Table 5-1. Each record represents a transaction. Only 12 of the 50 transactions are shown: the transactions with bread or milk purchased. A purchase is indicated by the number 1 in the cell and no purchase of the product by an empty cell. The other products purchased are not shown in the table. All the remaining transactions, 13 through 50, did not include milk or bread in checkout.

Table 5-1: Sample Supermarket Data

TRANSACTION	BREAD	MILK
1	1	
2	1	
3	1	
4	1	
5	1	1
6	1	1
7	1	1
8	1	1
9	1	1
10	1	1
11		1
12		1
. . .		
50		

In each transaction, two products are listed: bread and milk. If either was purchased, the column representing the project in the transaction is labeled as 1 if it was purchased and is empty if it wasn't.

Condition

A *condition*, or *conditional statement*, is a logical construct that evaluates to a true or false result. Conditions in association rules are built from nominal (categorical) input variables, unlike decision trees that can build conditions for both nominal and continuous variables. The following are examples of conditions:

- Product = "Beer_brand_x"
- Product_Beer_brand_x = 1
- SKU = BU4C3D5R2Z6
- Age in range 45 to 65
- Bread = 1
- Milk = ""

The first three bullets are different ways to represent the same idea: The first is Product as a multi-valued categorical variable, the second as an exploded

version of the multi-valued categorical variable so that each product is a 1/0 dummy variable, and the third is the actual SKU for that product. The next bullet is a condition for the variable AGE. Note that continuous variables can be used if they are binned prior to building the rules. The final two conditions refer to conditions related to products from Table 5-1.

Left-Hand-Side, Antecedent(s)

Antecedents are the "If" part of the rule. An association rule can be expressed by "if-then" or symbolically as left-hand-side (LHS) and right-hand-side (RHS) parts. An example rule containing milk and bread can be expressed in any of the four ways:

If (milk = 1) then (bread = 1)

If milk then bread

milk = 1 $\rightarrow$ bread = 1

milk $\rightarrow$ bread

The second and fourth examples assume implicitly that each of the two products were purchased. The antecedent in the rule is "milk = 1." There can be any number of antecedents in the rule, where the combination of rules is always an *and* function, such as:

If milk and peanut butter and jam then bread

Right-Hand-Side, Consequent, Output, Conclusion

The right-hand-side of the rule is the "then" part of the rule, bread in the examples for the LHS rules. As with the LHS rules, there can be multiple conditions on the RHS, although most implementations of association rules limit the conclusion to a single condition.

Every rule must have a consequent. However, a rule does not have to have any antecedents. When no antecedents are in the rule, the rule is merely the consequent itself, and match the records where the consequences is true.

Rule (Item Set)

The *rule*, or *item set*, is the full expression of the antecedents and consequent together, such as the rule "if milk then bread." In rules with no antecedents, the rule is merely the consequent alone.

Support

Support, also called *rule support,* is defined as the number of times a rule occurs in the data divided by the number of transactions in the data. If the data is arranged so that each record is one transaction, the number of transactions is the same as the number of records, and the calculation is the number of times the rule is true divided by the number of records.

Consider two rules built from Table 5-1:

bread → milk

milk → bread

The support for both rules in Table 5-1 is 12 percent:

Rule 1: If bread then milk

$$Support = \frac{\#\ transactions\ with\ bread = 1\ and\ milk = 1}{total\ \#\ transactions}$$

$$Support = \frac{6}{50} = 12\%$$

Rule 2: If milk then bread

$$Support = \frac{\#\ transactions\ with\ milk = 1\ and\ bread = 1}{total\ \#\ transactions}$$

$$Support = \frac{6}{50} = 12\%$$

While most often support is represented as a percentage of transactions, sometimes it is expressed as the actual number of transactions matching the rule.

Antecedent Support

Sometimes the antecedent support is calculated and reported separately. Antecedent support is the percentage of transactions when the antecedent is true, regardless of whether the consequent is true.

For the data in Table 5-1, the antecedent support for bread is 20 percent (10 purchases in 50 transactions), and the antecedent support for milk is 16 percent (8 purchases in 50 transactions). If there were more products listed in Table 5-1, and the antecedents were "bread and milk," the antecedent support of this two-condition antecedent would be 12 percent (6 occurrences of both milk and bread divided by 50 transactions).

Confidence, Accuracy

The *confidence* or *accuracy* of a rule is the percentage of transactions when the antecedents are true *and* the consequent is true divided by the number of rules where the antecedents are true, regardless of the value of the consequent—or, in other words, the support divided by the antecedent support.

For the rule "bread → milk," the support is 12 percent and the antecedent support is 20 percent, so the confidence is 12 ÷ 20, or 60 percent.

Lift

The *lift* of a rule is a measure of how many times more likely the consequent will occur when the antecedent is true compared to how often the consequent occurs on it's own. Expressed differently, it is the confidence of the rule divided by the baseline confidence of the consequent alone. The baseline confidence is the percentage of records with the consequent true.

To summarize, Table 5-2 lists several measures of association for two conditions shown in Table 5-1: bread and milk purchased at checkout. The first rule, where Bread is the antecedent and Milk is the conseqeuent, can be written "if Bread then Milk." It matches 12 percent of the records, Milk on its own matches 16 percent of the records, and the rule has a confidence of 60 percent, meaning that if someone purchased Bread (the antecedent), that person purchased Milk in 60 percent of transactions. The lift, 3.75, indicates that if the person purchased bread, the person is 3.75 times more likely to purchase Milk as well compared to all transactions.

Interestingly, the lift is the same for the reverse of the rule, "if Milk then Bread," though the rule confidence is higher (75 percent compared with 60 percent). Even though the rule confidence is higher, the lift is the same because the baseline confidence is also higher.

Table 5-2: Key Measures of Association Rules for Simple Market Basket Data

CONSEQUENT	ANTECEDENT	RULE SUPPORT %	BASELINE CONFIDENCE %	RULE CONFIDENCE %	LIFT
Milk	Bread	12	16	60	3.75
Bread	Milk	12	20	75	3.75

Note that while the support for both of the rules is the same, the confidence and lift are different; the direction of the rule matters.

Parameter Settings

The primary purposes for parameter settings in association rules are to reduce the number of rules identified by the algorithms and therefore speed up the process of finding the rules. This is a concern because of the combinatory explosion that occurs when the item set length increases. If there are 50 products to consider, there are 1,225 two-way combinations, 19,600 three-way combinations, and 230,300 four-way combinations. Clearly, with the vast number of rules generated from four and more antecedents , there is a need to reduce the set of rules that should be examined by the modeler.

Common parameter settings include:

- Minimum support
- Minimum confidence
- Maximum number of antecedents or itemset length (# antecedents + 1 for the consequent)
- Maximum number of rules

There is no theory to tell the modeler what the minimum or maximum values set by these parameters should be. If minimum support is set too high, no rules will be reported. If they are set too low, no rules are filtered and there may still be large numbers of rules. Sometimes the modeler will have intuition about the values of support. Sometimes a minimum confidence will be required in the business objectives.

One effective practice is to build rules with only one antecedent, then two antecedents, and continue to increase antecedents only after adjusting other parameters, such as minimum support or confidence, based on what is observed with a few antecedents.

How the Data Is Organized

Predictive modeling data consists of rows and columns, where rows describe a unit of analysis and columns a description of that unit of analysis. For customer analytics, a row may be a customer, and the columns, the set of descriptors of that customer.

Standard Predictive Modeling Data Format

Association rule data can be built from data where each record is a transaction and each column a product. For problems other than market basket analysis,

the same principles apply. For an invoice fraud detection problem, each row can represent an invoice and the columns describe the characteristics of the invoice. Association rules can be built from all of the categorical variables that describe the invoice.

In this layout, the columns used to build the association rules are either antecedents or consequents; each record has a unique record ID (a transaction in Table 5-1). The modeler must define which columns represent antecedents and which column is the consequent.

The problem with this format is that the width of the data can be extraordinarily large. Consider an example with a supermarket building association rules to identify products purchased at checkout (market basket analysis). If there are 20,000 SKUs to be tracked and evaluated individually, the data has to have 20,000 columns in the data. This is a challenge for many predictive analytics software packages. In addition, the representation of the data will be sparse. If, on average, only 5 SKUs are in any basket, 5 of the 20,000 columns in each record (transaction) will have a 1, indicating it is included in the transaction, and 19,995 will be 0.

Transactional Format

In transactional format, there are only a few columns but many more rows. Each row represents a single item to be associated with others, often a product name or product code. The only columns are Transaction ID and Item. (One may also have the item value or other ancillary pieces of information to aid in interpretation, such as customer ID and store ID, but these are not necessary for building the association rules.)

In this format, the association rules find relationships between rows having the same transaction ID rather than columns with values that co-occur in the same record. If ten products were purchased at checkout, that transaction would be represented by ten rows of data. Table 5-3 contains the same information as Table 5-1, but with each record representing a product in the shopping cart rather than containing all the products in the shopping cart. Transaction ID 5, for example, is represented in two records in Table 5-3 but only one record in Table 5-1.

Table 5-3: Sample Supermarket Data in Transactional Format

TRANSACTION	PRODUCT PURCHASED
1	Bread
2	Bread

3	Bread
4	Bread
5	Bread
5	Milk
6	Bread
6	Milk
7	Bread
7	Milk
8	Bread
8	Milk
9	Bread
9	Milk
10	Bread
10	Milk
11	Milk
12	Milk
. . .	
50	

The advantage of the transactional format is that the width of the data no longer becomes a problem. In the example with 20,000 SKUs, rather than having 20,000 columns in the data, you have only one column of SKUs, though many more rows. In addition to transactional format being narrower, it is also a more efficient representation of the data. In the format with one record per transaction, a cell exists for every SKU in a row of the data, meaning that the vast majority of cell values will be 0, indicating the SKU was not a part of the transaction. If the average basket size of a transaction is 5 SKUs, there are only five records per transaction in the transactional format.

Consider a data set with 1,000,000 transactions, 20,000 SKUs, and the same average of 5 SKUs per transaction. In the standard predictive analytics format, you have 1,000,000 rows and 20,000 columns, or 20 billion cells in the data containing SKUs with 1 or 0 values indicating they were included in the transaction. Of the 20 billion cells, only 1 billion are populated.

By way of contrast, the transactional format will have 1,000,000 transactions times 5 SKUs per transaction only, or 5,000,000 records and two columns.

Measures of Interesting Rules

Now that dozens, hundreds, or even thousands of rules have been identified in the data, what next? How does one identify which rules are interesting or good? This depends on the purpose of the model.

For some applications, support is the key: You are trying to discover which rules occur most often. For other applications, high confidence is key: You are trying to find rules whose consequent occurs more often in subsets of the data defined by the antecedents. Or perhaps you would like to identify the highest lift rules. In still other applications, confidence is important, but only if the support is high enough. Interpretation of these rules is subjective, depending on what the analyst and decision-maker believes is most interesting.

Consider association rules built on the KDD Cup 1998 data on a subset of the data including only donors (TARGET_D > 0). The data was set up in the standard predictive modeling format where each record was a donor. Several of the continuous variables were binned into quintiles, including LASTGIFT, AVGGIFT, NGIFTALL, and TARGET_D. The consequent was set as the top quintile of giving: TARGET_D between $20 and $200. The rules were limited to a maximum of two antecedents, a minimum support of 5 percent, and a minimum confidence of 1 percent. More than 1,400 rules were found using the Apriori algorithm.

After sorting by confidence, the top rules are shown in Table 5-4. As is typical when sorting by confidence, many of the rules are just above the minimum support threshold, although the top rule is considerably above this threshold. Note, too, that the fifth and sixth ranked rules both have support of 5.492 percent and confidence of 87.218, strong evidence that these two rules match the identical records.

Table 5-4: List of Association Rules Sorted by Confidence

CONSEQUENT	ANTECEDENT	SUPPORT %	CONFIDENCE %	LIFT
TARGET_D = [20, 200]	RFA_2A = G and AVGGIFT = [15, 450]	11.5	88.9	2.82
TARGET_D = [20, 200]	RFA_2 = L1G and AVGGIFT = [15, 450]	6.5	88.2	2.80
TARGET_D = [20, 200]	RFA_3 = A1G and LASTGIFT = [20, 450]	5.5	87.7	2.78
TARGET_D = [20, 200]	RFA_4 = A1G and LASTGIFT = [20, 450]	5.4	87.5	2.77

TARGET_D = [20, 200]	RFA_3 = A1G and RFA_2 = L1G	5.45	87.2	2.77
TARGET_D = [20, 200]	RFA_3 = A1G and RFA_2F = 1.0	5.5	87.2	2.77
TARGET_D = [20, 200]	RFA_5 = A1G and LASTGIFT = [20, 450]	5.1	87.1	2.76
TARGET_D = [20, 200]	RFA_4 = A1G and RFA_3 = A1G	5.6	87.0	2.76
TARGET_D = [20, 200]	RFA_2 = L1G and LASTGIFT = [20, 450]	7.6	87.0	2.76
TARGET_D = [20, 200]	RFA_4 = A1G and RFA_2 = L1G	5.4	87.0	2.76
TARGET_D = [20, 200]	RFA_4 = A1G and RFA_2F = 1.0	5.4	87.0	2.76
TARGET_D = [20, 200]	RFA_3 = A1G	5.7	87.0	2.76
TARGET_D = [20, 200]	RFA_3 = A1G and RFA_2A = G	5.7	87.0	2.76
TARGET_D = [20, 200]	RFA_2A = G and LASTGIFT = [20, 450]	13.2	86.9	2.76
TARGET_D = [20, 200]	AVGGIFT = [15, 450] and LASTGIFT = [20, 450]	18.2	86.8	2.75
TARGET_D = [20, 200]	RFA_4 = A1G	5.6	86.7	2.75
TARGET_D = [20, 200]	RFA_4 = A1G and RFA_2A = G	5.6	86.7	2.75
TARGET_D = [20, 200]	RFA_5 = A1G and RFA_3 = A1G	5.1	86.7	2.75

A second list is shown in Table 5-5, this time sorting the rules by support. The top rule matches nearly 60 percent of the population, although it has a lift below 1.0, meaning that when PEPSTRFL is true, the likelihood that a donor is in the top giving group is smaller than average. One interesting rule is the fifth, with the antecedent LASTGIFT in the range 20 to 450. This matches more than 27 percent of the population and still has confidence nearly as high as the top confidence rules. Just below this rule are four rules that apparently match the same records as evidenced by their identical support and confidence values: These are redundant interactions.

Table 5-5: List of Association Rules Sorted by Support

CONSEQUENT	ANTECEDENT	SUPPORT %	CONFIDENCE %	LIFT
TARGET_D = [20, 200]	PEPSTRFL	59.6	21.1	0.67
TARGET_D = [20, 200]	RFA_2A = F	42.5	42.9	1.36
TARGET_D = [20, 200]	RFA_2F = 1.0	37.0	53.3	1.69
TARGET_D = [20, 200]	RFA_2A = E	28.9	4.1	0.13
TARGET_D = [20, 200]	LASTGIFT = [20, 450]	27.6	79.6	2.52
TARGET_D = [20, 200]	RFA_2 = L1F	24.2	52.3	1.66
TARGET_D = [20, 200]	RFA_2 = L1F and RFA_2F = 1.0	24.2	52.3	1.66
TARGET_D = [20, 200]	RFA_2 = L1F and RFA_2A = F	24.2	52.3	1.66
TARGET_D = [20, 200]	RFA_2F = 1.0 and RFA_2A = F	24.2	52.3	1.66
TARGET_D = [20, 200]	AVGGIFT = [15, 450]	24.0	75.5	2.40
TARGET_D = [20, 200]	NGIFTALL_ bin_3 [7, 11]	23.0	30.9	0.98
TARGET_D = [20, 200]	LASTGIFT_bin_4 [15, 19]	22.8	31.0	0.98
TARGET_D = [20, 200]	RFA_2A = E and PEPSTRFL	22.5	3.8	0.12
TARGET_D = [20, 200]	LASTGIFT_bin_4 [15, 19] and RFA_2A = F	22.2	30.3	0.96

Deploying Association Rules

In many applications, association rules are built solely to understand the data better. However, there are other applications where deployment of the rules is desirable. For example, you can build rules for cross-sell or up-sell of retail products. You may find a rule that identifies that if a customer buys a particular pair of black pants, he also tends to buy a black belt. The model can be used to

identify customers who have purchased the black pants, but do not have the black belt in their basket: These customers are good candidates for additional treatment or incentives to encourage the purchase that many others have made.

When deploying the rules, they are implemented in the order specified by the analyst: by confidence, support, antecedent support, lift, or some other criterion of interest. However, for applications such as product recommendations and couponing, you should check not only if the antecedent products are in the basket, but also if the consequent product is in the basket. Some products are purchased only once (the consequent) and should not be encouraged to be purchased again. Other products can be purchased in greater number; they can be recommended even if already purchased.

Redundant rules never fire. In Table 5-5, if the antecedent RFA_2 = L1F is in the record, the consequent, TARGET_D = [20, 200], is expected to be true. However, the next three rules in the list will never fire because no matches remain after the RFA_2 = L1F rule fires.

Variable Selection

One variable selection strategy is to identify redundant variables and keep one of them for modeling, removing all the others. This is often done by examining the correlations of candidate input variables. However, this works for continuous variables only.

Association rules can be used to compare all categories in much the same way. After building the association rules, antecedents with identical support and confidence are identified. Only the variables in antecedents from one of the redundant rules are kept; the other variables are discarded. This works most easily with simple rules (two antecedents), but can be extended to more.

Interaction Variable Creation

Some algorithms are "main effect" models and are not able to identify interactions directly. Linear and logistic regression, k-nearest neighbor, and K Means clusters are all examples. However, interactions are often useful and sometimes critical to building accurate predictive models.

Creating interactions is most often either a manual process, where each interaction is hypothesized and then built one at a time by the analyst, or an exhaustive process whereby algorithms include every interaction. If one begins with 50 input variables, there are 1,225 two-way interactions either to test manually, or all are included in the model. Association rules can be used to find the best

interactions by searching exhaustively through all possible combinations of interactions and listing them by their association to the target variable.

The three-step process for finding interactions using association rules is as follows:

1. Set up the association rules to have one consequent: the predictive model target variable.
2. Set the parameters to include only two or three antecedents depending on whether one is trying to find two-way or three-way interactions. If the software limits by the itemset length rather than antecedents, choose an itemset length of 3 for two-way interactions or 4 for three-way interactions.
3. After building the rules, sort them by confidence. Include the interactions with the highest confidence if they match different records (don't select two interactions with identical confidence and support).

Once the top rules have been identified, these definitions can be used to build new columns as new dummy variables that indicate this combination.

However, the three-step process could result in far too many rules for practical use. If too many interactions remain after applying Step 3, the approaches for reducing described in the section "Variable Selection Prior to Modeling," in Chapter 4. can be used, including applying statistical tests and computing correlations. Including only rules that domain experts recognize as interesting interactions can also help remove rules.

Problems with Association Rules

There are still many challenges to building, understanding, and deploying association rules.

Redundant Rules

You have already seen examples of redundant rules in previous sections. If the software supports the removal of these rules easily through a rule editor, the redundant rules can be deselected. Otherwise, the analyst must remove the variables before building the association rules. Removing redundant rules does not affect how the association rules are deployed, only the efficiency of the rule deployment. However, the time savings can be significant.

Too Many Rules

Sometimes so many rules are found that navigating the rules to find what is interesting becomes very difficult. In these cases, removing redundant

rules, increasing thresholds, or reducing the number of input variables (if possible) can help focus the analysis. If the number of antecedents or the item set length is large, the size can quickly exceed what is expedient to compute.

Too Few Rules

If the minimum support or confidence thresholds are too high, no rules may be found. Lowering these thresholds can help, although knowing specifically what threshold is low enough is not always obvious.

Building Classification Rules from Association Rules

When the consequent of the association rules is limited to a single variable, association rules appear very similar to classification models. They have inputs and an output. They generate rules, very much like decision trees. One can measure the accuracy of the rule much like classification accuracy or lift.

However, there are also important differences, such as:

- Association rules can have multiple consequents, one consequent condition per rule.
- Association rules, in some implementations, can even have multiple conditions as a part of the consequent.
- Classification finds the *best* way to associate inputs with the output. Association rules find *all* associations between the inputs and the outputs.
- Most classification algorithms operate on both continuous and categorical data. Association rules operate only on categorical data.
- Most classification algorithms guard against overfit, whether directly or indirectly. Association rules don't consider overfit or underfit, although one can increase support to decrease the likelihood of finding happenstance rules.

Association rules can be converted into a viable, supervised learning classification algorithm by applying a few additional constraints. Some commercial tools support building classification rules, greatly simplifying the task of the analyst to produce them.

Two steps are often applied when building classification rules:

- Allow only one consequent, the target variable. Some implementations of association rules do not allow this limitation, and therefore rules will have to be filtered after they are built to eliminate all consequents except for the target variable.

- Include rules that, after applying the rules in sequence (such as by confidence), match a sufficient number of records to produce a statistically significant prediction. Significance can be measured by tests such as the chi-square test.

Table 5-6 shows classification rules for a fraud detection application. The consequent is the fraud variable, a 1/0 outcome, and the rules in the table measure the confidence that the transaction was *not* fraud. Top rules where classification rules were applied to a new data set are included in the table from the thousands of rules found.

The first rule, 278, will match the same records when applying the rule as it did on the training data. However, once this rule matches, those 11,809 records have already been classified. The second and third rules (2255 and 1693) have slightly smaller confidence in selecting non-fraudulent transactions because they matched transactions already classified by the higher-accuracy Rule 278 that preceded them. This is even more evident with Rules 2258 and 1337 where only 16 and 20 transactions, respectively, match. On the training data, not shown in the table, these matched more than 4,000 transactions each.

Table 5-6: Applying Rules in Sequence

RULE ID	# RECORDS HIT BY RULE IN SEQUENCE	CONFIDENCE (FROM MODEL) %	CONFIDENCE APPLYING RULE IN SEQUENCE %
278	11,809	99.53	99.53
2255	7,215	99.50	99.21
1693	1,927	99.49	98.91
2258	16	99.49	93.75
1337	20	99.48	95.00
993	2,727	99.48	98.90
977	2	99.47	100.0
2134	2,516	99.46	98.77
1783	4,670	99.46	99.07
984	6	99.46	100.0

Table 5-6 requires pruning to eliminate rules that, when applied in the sequence, no longer provide a stable prediction of the target variable.

Summary

Association rules provide a fast and flexible way to identify patterns in categorical variables. Their speed allows them to be applied to large data sets. Their flexibility allows them to be applied in a variety of ways, including building supervised or unsupervised models and finding good interaction effects for use with other modeling algorithms.

CHAPTER

6

Descriptive Modeling

Descriptive modeling algorithms are also called unsupervised learning methods, meaning that the algorithms try to find relationships between inputs rather than to a specific target variable. It is an appropriate technique for modeling problems when a target variable cannot be quantified or when the purpose of modeling is only to find relationships between inputs.

Consider a company seeking to design marketing programs that are effective for their customers. A single program for all customers may be effective, but usually a company will identify customer *segments* or subgroups with similar characteristics to each other, but different from the other segments. These segments can form the basis of separate, more specific marketing campaigns based on attributes such as age, sex, income, socio-economic status, geographic region, and more.

Unsupervised learning methods discover the best way to segment the data; the algorithms find which attributes drive the formation of the segments, how large these segments are naturally in the data, and what the statistical characteristics of these segments are. For example, a company may discover from its data that 10 percent of its customers are female homeowners in their thirties living in the northeast and any of several levels of education. This segment says nothing about any target variables, such as customer lifetime value (CLV), unless the target variable is also treated as an input to the model.

Or consider another example. A company would like to understand the risk associated with making a loan to an applicant so they can apply additional scrutiny to potentially high-risk loan applicants. The company collected years of data including measures of which customers successfully paid their loans in full and which customers defaulted on their loan, so on the surface, this appears to be a good candidate for a supervised learning approach to predictive modeling: predict the likelihood a customer will default on a loan given the known attributes of that customer at the time the decision is made to issue the loan or deny it. The company has created a label with value 1 for those customers who defaulted, and a label with value 0 for those who did not default.

However, there can be problems with this approach. First, it can take years to prosecute a customer for default on a loan. The more time that elapses between the decision to issue the loan and the default, the more difficult it is to use the default action as a target variable because the economic circumstances of loan applicants changes with time; five years after a loan is issued, circumstances that led to an individual's default may no longer exist. Second, a customer may not legally default, but may still represent high risk. The loan may be written off rather than being labeled a "default." Or the loan may have been delinquent for many months and a final settlement was reached with the customer to avoid legal battles. These loans may be labeled "not default," which is technically true, but the label doesn't accurately reflect the true risk of the loan.

Because of these kinds of complications with supervised learning techniques applied to identifying risk, some companies opt for building an unsupervised learning model to identify unusual or anomalous customers that may be associated with higher risk. The thinking goes like this: The vast majority of customers will not default, and their attributes should look "normal." However, some customers will have unusual financial characteristics that should lead to increased scrutiny of their application. These anomalous customers could be flagged for additional screening prior to loan approval.

The unsupervised modeling algorithms described here are the ones most commonly implemented in predictive software: K-Means clustering, Kohonen, Self-Organizing Maps (SOM), and Principal Component Analysis (PCA).

Data Preparation Issues with Descriptive Modeling

All three of these algorithms have several requirements for data preparation. Details of how to prepare data for clustering are provided in Chapter 4. First, the inputs must be numeric. If one would like a categorical variable to be included in the model, it must be converted to a number. Most commonly, this is done through exploding the categorical variable, with N levels or values, into N dummy variables.

Second, all the data must be populated; there can be no missing values. Any missing values need to be imputed with a numeric value, or records with any

missing values should be removed (listwise deletion). Third, usually, practitioners try to transform inputs so that they are more nearly normally distributed. This includes removing severe outliers and reducing skew (positive and negative).

This chapter explains key ideas for descriptive modeling. The algorithms included in the chapter are the three that are found most commonly in the leading predictive analytics software packages and are used most often when analyzing large data.

Principal Component Analysis

Principal Component Analysis (PCA) is an old and well-known algorithm, and one that is available in nearly every predictive analytics software package. However, it isn't always described in predictive analytics and data mining textbooks, especially those books that present algorithms from a machine learning perspective.

PCA is often understood as a dimensionality reduction method: a technique to reduce the number of inputs to predictive models by combining original variables that are correlated with each other into a smaller number of independent, information-rich variables. But PCA can also be used to describe interesting characteristics of data, identifying variables that are correlated with each other.

The PCA Algorithm

We begin with two variables from the nasadata data set shown in the scatterplot in Figure 6-1, Band1 and Band2. The narrow band of values in the cigar-shaped cloud of data indicates that these two variables are strongly correlated with each other. In fact, their correlation coefficient is +0.855—very high.

From a predictive modeling perspective, if you include both of these variables as inputs to the models, are you getting two *ideas*? Or, mathematically speaking, how many *independent* ideas are represented as inputs? Independence is an assumption in many modeling algorithms, not least of which is linear regression. When two variables are correlated, they have shared variance, or co-variance. Coefficients in a linear regression model therefore do not convey the full story in the predictive model of how much influence a variable really has in predicting the outcome.

The principal components (PCs) in a PCA model identify the direction where the spread of the data is maximized. In two dimensions, this corresponds to the image in Figure 6-2: The first principal component (PC1) is the direction where the variance is largest, the arrow pointing to the upper right, and by extension, extending also downward and to the left. The second PC is by definition of the algorithm orthogonal to the first, where orthogonal means it is perpendicular but in multidimensional space. Because these variables are orthogonal, their correlation is exactly 0, or in other words, they are mathematically *independent*

from one another. There is no shared variance, or stated equivalently: One cannot predict one projection from the other. This is ideal for linear regression models, for example, because the regression algorithm assumes independence of the input variables. It can also be beneficial for other numeric algorithms such as neural networks, k-nearest neighbor, and support vector machines.

With more than two inputs, the process repeats with each subsequent component still orthogonal to all the previous PCs, but representing the direction of the largest variance remaining in the data. This is purely a conceptual characterization; the PCA algorithm is not iterative, but finds all of the PCs simultaneously through matrix multiplication (matrix decomposition) and is computationally fast even for a large number of inputs. The actual PCs are linear projections of the original inputs. PCA continues until there are no more directions left in the data. If there are N inputs to the PCA model, this occurs after N PCs are computed.

The projection of these two variables, Band1 and Band2, onto the PC space, PC1 and PC2, is shown in Figure 6-3. The axes were scaled in the scatterplot to the same minimum and maximum values, revealing how much more spread there is in PC1 compared to PC2. The correlation between the two variables is zero and can be inferred by a simple test: Can one pick a value of PC1 and predict from that the value of PC2? The answer is no: For any value of PC1, PC2 can be any number between –5 and 5.

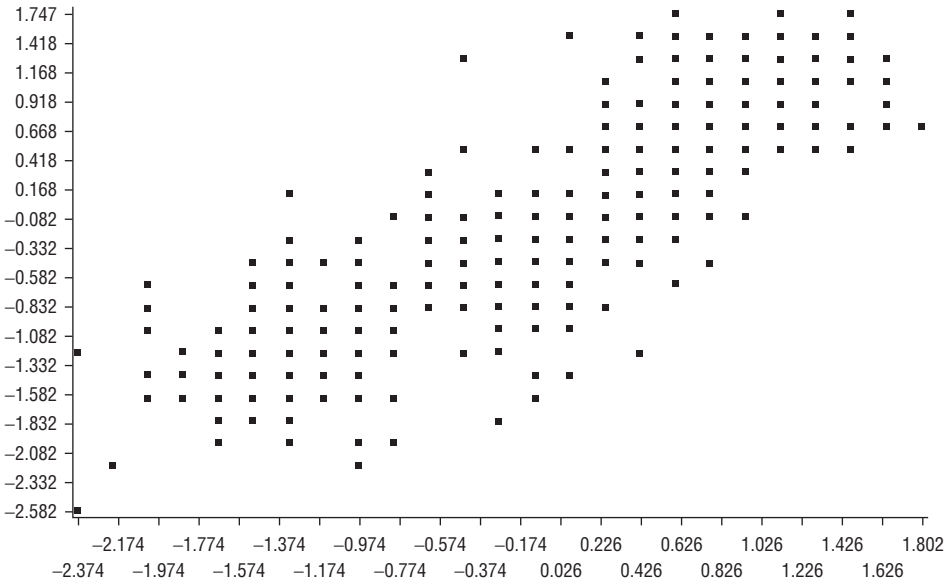


Figure 6-1: Nasadata Band1 vs. Band2

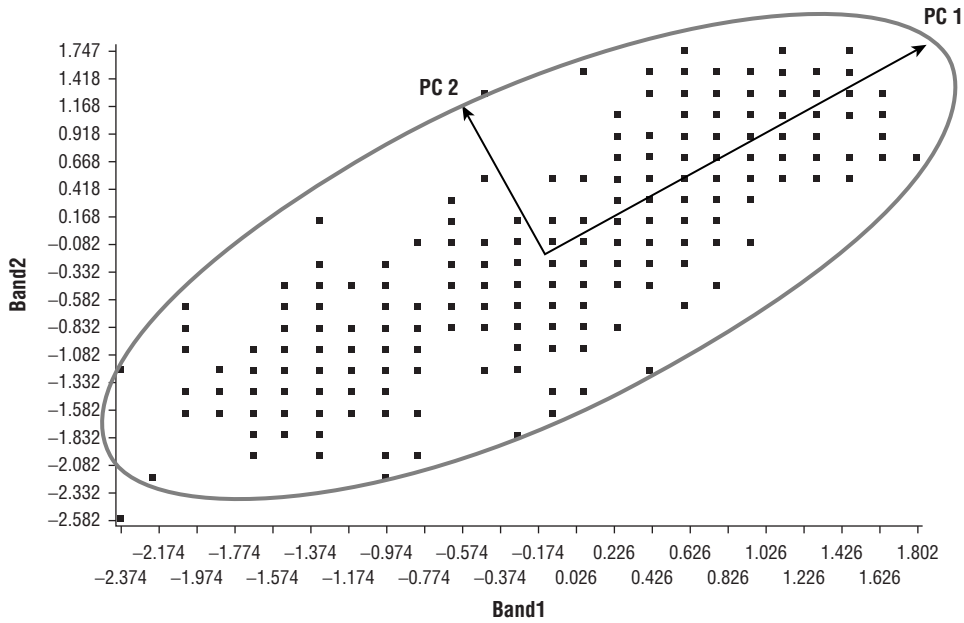


Figure 6-2: Nasadata Band1 vs. Band2 with principle component directions

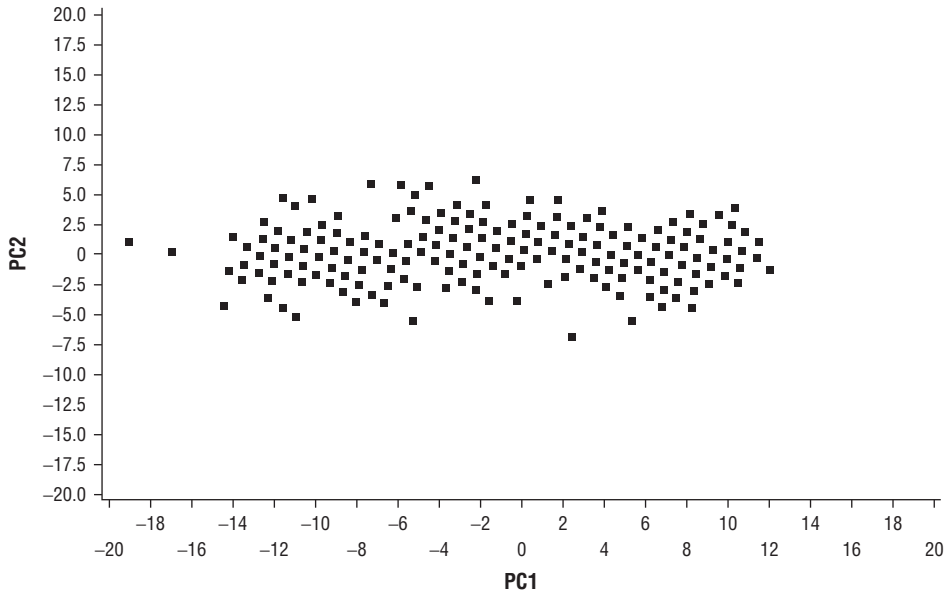


Figure 6-3: Scatterplot of Principal Component 2 vs. Principal Component 1

The matrix decomposition of the PCA algorithm converts the original data into two other matrices, a diagonal matrix containing *eigenvalues*, and a second matrix containing *eigenvectors*. The precise definition of these matrices isn't the concern of this book, but these two terms will be used in some predictive analytics software. There are as many eigenvalues as there are original inputs used in computing the PCs. Likewise, there are as many PCs as there are inputs to the model. The eigenvalues represent the proportion of variance explained by each PC. For the simple example, a PCA model with only Band1 and Band2 as inputs, the eigenvalues are 54.66 and 4.19, as shown in Table 6-1. The percent variance explained by the first PC is 92.9 percent [54.66 ÷ (54.66+4.19)], meaning that nearly all of the variance in the scatterplot shown in Figure 6-1 can be explained by a single component, PC1.

Table 6-1: Eigenvalues and percent Variance Explained for PCA Computed from Band1 and Band2

PCA—BAND1 & BAND2	EIGENVALUE	% VARIANCE EXPLAINED	CUMULATIVE % VARIANCE EXPLAINED
PC1	54.66	92.9	92.9
PC2	4.19	7.1	100

Table 6-2: Eigenvectors for PCA Computed from Band1 and Band2

EIGENVECTORS	BAND1	BAND2
PC1	0.756	0.655
PC2	0.655	–0.756

The eigenvectors, shown in Table 6-2, represent the contributions of each input variable onto each PC: PC1 = 0.756 × Band1 + 0.655 × Band2, and PC2 = 0.655 × Band1 – 0.756 × Band2. These values are sometimes called *loadings* of the variables on the PCs, although this term is more often used in the related field of Factor Analysis. The loadings of the variables onto the first PC are nearly equal, understandable given that the two variables are linearly correlated. In the second component, they have opposite signs of one another, although once again similar magnitudes, indicating that the direction of the second component is perpendicular to the first.

In two dimensions, there isn't much need for PCA. However, in larger dimensional space, the value can be considerable. The nasadata data set has 12 candidate inputs. The PCA model inputs were normalized using z-scores so they are all on the same scale. When PCA is extended to all of these dimensions, you get the eigenvalues shown in Table 6-3.

Table 6-3: Twelve PCs for Nasadata

PCA	EIGENVALUE	% VARIANCE EXPLAINED	CUMULATIVE % VARIANCE EXPLAINED
PC1	8.622	71.8	71.8
PC2	2.064	17.2	89.0
PC3	0.840	7.0	96.0
PC4	0.113	0.9	97.0
PC5	0.106	0.9	97.9
PC6	0.065	0.5	98.4
PC7	0.056	0.5	98.9
PC8	0.050	0.4	99.3
PC9	0.036	0.3	99.6
PC10	0.022	0.2	99.8
PC11	0.014	0.1	99.9
PC12	0.012	0.1	100.0

The first three PCs explain 96 percent of the variance in the data; this data has a high degree of cross-correlation in the inputs. Even though there are 12 candidate inputs for use in predictive models, there are really only perhaps three true dimensions to the data; one can use the top three PCs without losing much of the information (variance) in the data.

The Cumulative % Variance Explained column is often plotted in a *scree plot*, like the one shown in Figure 6-4. Sometimes practitioners identify a *knee* in the plot to determine when one has reached a point of diminishing returns in adding PCs to the model. This is an inexact science to be sure, but is used often nevertheless. In this figure, the knee appears at the third PC, which is consistent with the percent variance explained method of selecting PCs.

Applying PCA to New Data

Once the principal components have been computed, the model itself is the set of eigevector; these are saved by the software and applied to new data. The modeler typically selects how many PCs to keep and the software computes the projections for the appropriate eigenvectors. These new variables, the PCs, are added to the data flow as additional columns in the data. For example, if one decides to keep three PCs from the nasadata, three new created PCs from the

twelve inputs (Band1, Band2, . . . , Band12) will be added. If one computed the variance of each of these three PCs on the training data used to create the PCA model, one will see that the variance matches the eigenvalues exactly; even in this sample shown in Table 6-4, it is clear that PC1 has larger magnitudes than PC2 and PC3. If the input data was z-scored or scaled to have zero mean, the PCs will also have zero mean.

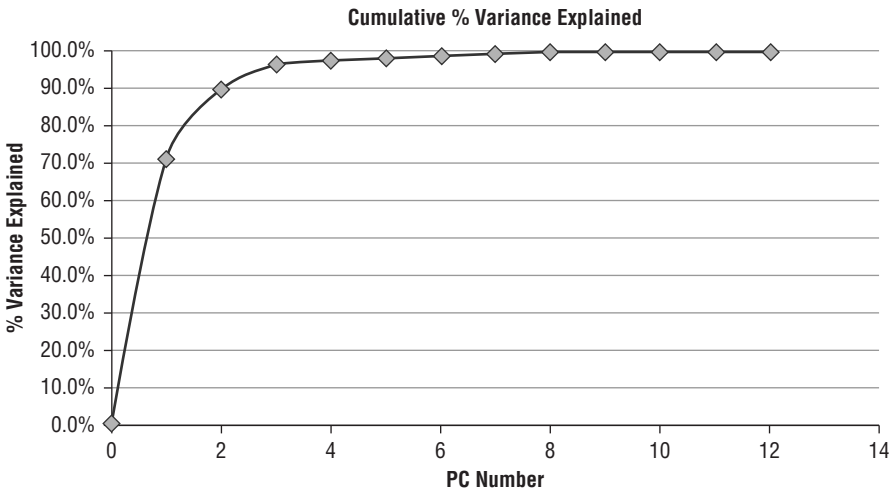


Figure 6-4: Scree plot

Table 6-4: Top Three Principal Component Values for Ten Data Points

CLASS	PC1	PC2	PC3
alfalfa	–2.678	–0.342	–0.485
wheat	1.121	2.642	1.195
wheat	0.920	2.442	0.972
rye	3.064	0.888	-0.556
alfalfa	–2.451	–0.262	–0.840
oats	1.726	0.404	–0.491
rye	4.602	0.504	–1.146
soy	0.878	–1.234	1.286
oats	0.781	0.278	–0.568
soy	0.060	–0.869	1.399

PCA for Data Interpretation

So far, PCA has been described as a way to reduce the number of inputs for predictive models. Another important use of PCA models is for data interpretation and variable selection.

Interpretation of PCA models is subjective. Most often, practitioners look at the loadings for each PC and pick the top few values. These variables are then deemed to be the ones that are most important in the formation of that PC. The cutoff for which variables are the top contributors is not always obvious and usually not determined in a precise way.

For example, in Table 6-5, the top six PCs are shown along with the loadings (eigenvector values) for each PC. The highest loading variables on each PC are shown in boldface. In the first PC, all of the inputs, Band1 through Band12, contribute to the PC, although Band4 and Band5 contribute slightly more than the others. One might label PC1 as the "main effect" component because of the broad-based contribution from all inputs. PC2, however, has as its top loaders Band1 and Band10. So PC2 is the Band1+Band10 component. PC3 is dominated by Band6, Band11, and Band12, and therefore is the Band6+Band11+Band12 component. PC4 is the Band3+Band6+Band10 component, and so forth.

Table 6-5: Six Principal Components of Nasadata

NASADATA	PC1	PC2	PC3	PC4	PC5	PC6
Band1	−0.188	**0.559**	−0.088	0.249	0.090	**−0.694**
Band2	−0.273	0.372	0.018	0.264	**−0.662**	**0.389**
Band3	−0.311	0.201	0.038	**0.417**	**0.645**	**0.442**
Band4	−0.330	0.072	0.023	−0.003	−0.127	−0.240
Band5	−0.329	0.082	0.133	−0.110	−0.034	0.217
Band6	−0.283	0.138	0.519	**−0.506**	−0.065	−0.015
Band7	−0.313	−0.118	0.320	−0.194	0.265	−0.121
Band8	−0.307	−0.292	0.022	−0.009	−0.005	−0.055
Band9	−0.291	−0.346	−0.093	0.175	−0.128	−0.091

Continues

Table 6-5 (*continued*)

NASADATA	PC1	PC2	PC3	PC4	PC5	PC6
Band10	–0.221	–**0.504**	0.150	**0.456**	–0.143	–0.191
Band11	0.289	0.067	**0.545**	0.320	0.015	–0.031
Band12	0.295	0.003	**0.522**	0.218	–0.090	–0.045

The magnitude of eigenvectors determines the influence of the variable on the PC, but the sign indicates the direction of influence. In PC2 from Table 6-4, Band1 and Band10 are the top loaders, but they do so in the opposite direction as indicated by the opposite signs. Interestingly, they are only mildly negatively correlated on all the data (–0.22), but in the first component PC1, their loadings are in the same direction (both negative). The PCA model is saying that as indicated by PC2, there are records where the two variables are strongly but inversely correlated with each other.

Sometimes, rather than using the PCs themselves as the inputs to predictive models, you can use the top loading variable for each of the components as the representative of that component. From Table 6-4, instead of using the top three PCs as inputs to a predictive model, you could use the top magnitude representatives from these PCs: Band4 from PC1, Band1 from PC2, and Band11 from PC2. In fact, in several applications of PCA analysis for dimensionality reductions, this approach of using the representative variable has actually improved predictive accuracy over using the actual principal components.

Additional Considerations before Using PCA

PCA, however, has some issues to keep in mind before using it for variable selection with predictive modeling. First, just because the dimensionality and a few principal components can describe the vast majority of the variance in the data doesn't mean that classification or regression accuracy will be improved. Consider Figure 6-5. The data is bi-modal as evidenced by the two ellipses; one ellipse for the dark gray class and a second for the light gray class. The upper arrow shows the direction of the first PC computed from the light gray data only and the lower arrow shows the direction of the first PC computed from the

dark gray data only. The first PC when computed from all the data, both dark gray and light gray, is in the same direction as these two.

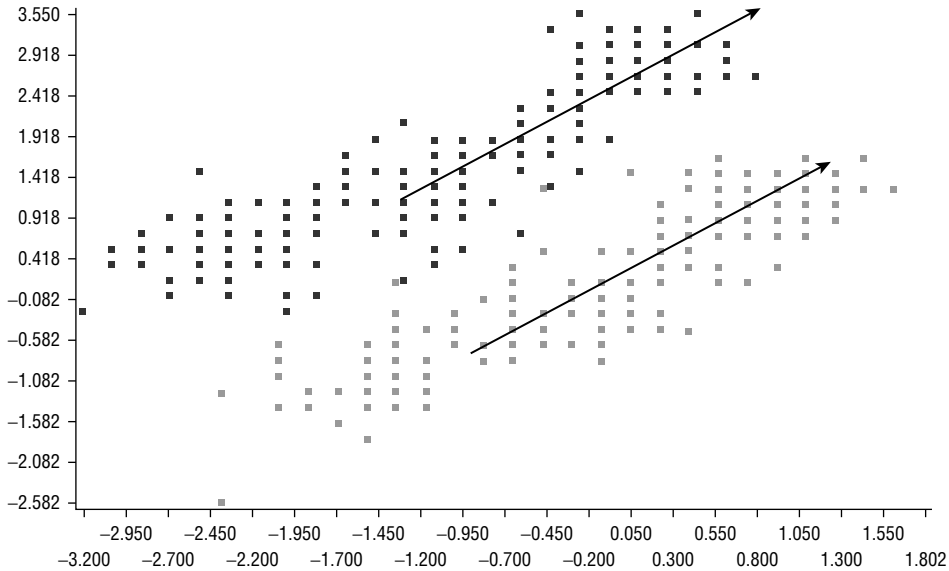


Figure 6-5: Principal components for bi-modal distribution

After performing PCA, the histogram of the first PC in Figure 6-6 shows overlap between the two classes on this first PC; just because a PC describes large amounts of variance in the data doesn't mean that it will discriminate between target classes.

Second, even if some of the components are good predictors, one doesn't necessarily know which ones are the best predictors from the variance explained. It may be that PC6 is the best predictor even though it explains relatively little variance overall in the data. Third, PCA is a *linear* method that creates linear projections of the inputs onto each of the PCs. If the predictive information in the inputs is nonlinearly related to the target variable, or if the key interactions between variables are nonlinear, the PCA transformation itself can destroy any predictive information the inputs had, making predictive models less accurate than models built from the original inputs.

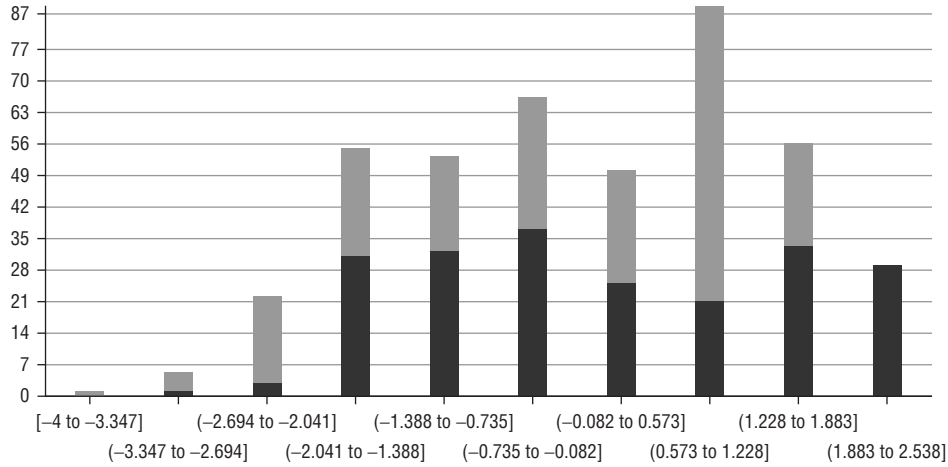


Figure 6-6: Histogram of first principal component of bi-modal data

Fourth, while PCA can reduce dimensionality, such as reducing the nasadata from twelve variables down to the top three PCs, the reduction affects only the models, not the original data needed to build and score the PCs. Whenever you apply the PCA model, you still need all twelve of the original inputs to create the three PCs. From a deployment perspective, this is not a simplification of data size at all.

PCA works best when the data is numeric and distributed either Normally or Uniformly rather than having multiple modes, clumps, large outliers, or skewed distributions. Moreover, PCA works best when the data has primarily linear relationships between inputs and with the target variable.

The Effect of Variable Magnitude on PCA Models

Computing PCs unbiased by input variable magnitude requires normalization of the data. At a minimum, scaling all the inputs to have comparable maximum magnitude. This problem manifests itself in two ways. First, larger magnitude input variables will load higher in the earlier PCs. Table 6-6 contains the minimum and maximum values for the twelve nasadata input variables in their natural units, that is, as they occur in the data before any normalization or transformation. Note that the largest magnitudes for the minimum and maximum values in the table are for Band11 and Band12. Then, in Table 6-7, you can see that in this PCA model, Band11 and Band12 have the largest magnitude and also dominate the first PC. The highest loading variables on each factor are shown

in boldface. After normalizing the data (using z-scoring), the first PC is distributed throughout all of the bands in the PCA model summarized in Table 6-5.

Table 6-6: Summary Statistics for Nasadata Input Variables

NASADATA	MINIMUM	MAXIMUM	RANGE	MAX MAGNITUDE
Band1	–13.6	10.4	24	13.6
Band2	–13.1	8.9	22	13.1
Band3	–8.6	7.4	16	8.6
Band4	–10.9	7.1	18	10.9
Band5	–18.2	14.8	33	18.2
Band6	–15.8	9.2	25	15.8
Band7	–14.5	8.5	23	14.5
Band8	–26.7	21.3	48	26.7
Band9	–24.8	22.2	47	24.8
Band10	–24.5	18.5	43	24.5
Band11	–68.1	37.9	106	68.1
Band12	–43.8	24.2	68	43.8

Table 6-7: Eigenvectors of First Six PCs from Nasadata with Natural Units

NASADATA	PC1	PC2	PC3	PC4	PC5	PC6
Band1	–0.063	–0.266	**0.423**	0.054	**–0.500**	**–0.586**
Band2	–0.090	–0.123	0.372	–0.029	**–0.419**	**0.547**
Band3	–0.078	–0.031	0.228	0.053	–0.080	–0.214
Band4	–0.097	0.006	0.205	–0.006	–0.060	0.062
Band5	–0.169	0.036	**0.446**	–0.078	0.177	0.285
Band6	–0.084	0.070	**0.444**	–0.234	0.352	0.039
Band7	–0.097	0.127	0.225	–0.074	0.219	–0.169
Band8	–0.304	**0.398**	0.144	–0.042	0.322	–0.283
Band9	–0.316	**0.417**	–0.051	0.103	0.283	–0.294

Continues

Table 6-7 (*continued*)

NASADATA	PC1	PC2	PC3	PC4	PC5	PC6
Band10	–0.198	**0.602**	–0.108	0.105	–0.351	–0.165
Band11	**0.702**	0.341	0.328	**0.519**	0.058	0.041
Band12	**0.451**	0.283	0.019	**–0.797**	–0.226	–0.041

Second, the magnitudes of the eigenvalues are also affected by the magnitudes of the inputs. One rule-of-thumb many modelers use to select the significant principal components is to keep those whose eigenvalues are greater than 1. In Table 6-8, ten of the twelve fit this criterion even though the first four components already describe 99 percent of the variance. After normalizing the data, only two PCs had eigenvalues greater than 1 (see Table 6-3).

Note, however, that simple scaling affects the eigenvalues but *not* the percent of the variance explained. If the z-score normalized inputs are scaled upward by a factor of 7 by simple multiplication, all of the eigenvalues end up being greater than 1. However, the *relative* contribution of the PCs does not change with a simple scaling of the inputs; the percent variance explained is the same for both the z-scored inputs and the z-scored inputs that were scaled upward by the factor of 7.

Table 6-8: Principal Component Eigenvalues for Nasadata with Natural Units

NASADATA	EIGENVALUE	% VARIANCE EXPLAINED	CUMULATIVE % VARIANCE EXPLAINED
PC1	1317.3	83.0	83.0
PC2	169.5	10.7	93.6
PC3	75.34	4.7	98.4
PC4	9.412	0.6	99.0
PC5	5.093	0.3	99.3
PC6	2.471	0.2	99.4
PC7	2.123	0.1	99.6
PC8	1.929	0.1	99.7
PC9	1.672	0.1	99.8
PC10	1.302	0.1	99.9

NASADATA	EIGENVALUE	% VARIANCE EXPLAINED	CUMULATIVE % VARIANCE EXPLAINED
PC11	0.967	0.1	99.9
PC12	0.845	0.1	100.0

Clustering Algorithms

Cluster models find groups of data points that are relatively close to one another. Inherent in the definition of the word "close" is the concept of distance between data points.

Consider a simple scatterplot from the KDD Cup 1998 data in Figure 6-7. It is obvious that point P_1 and point P_2 are closer to one another than points P_1 and P_3. The most common way this is measured in clustering algorithms is by the Euclidean distance, the distance we all learned in high-school algebra: "The shortest distance between two points is a straight line." Distance plays a prominent role in both clustering algorithms described in this chapter: K-Means and Kohonen Self-Organizing Maps.

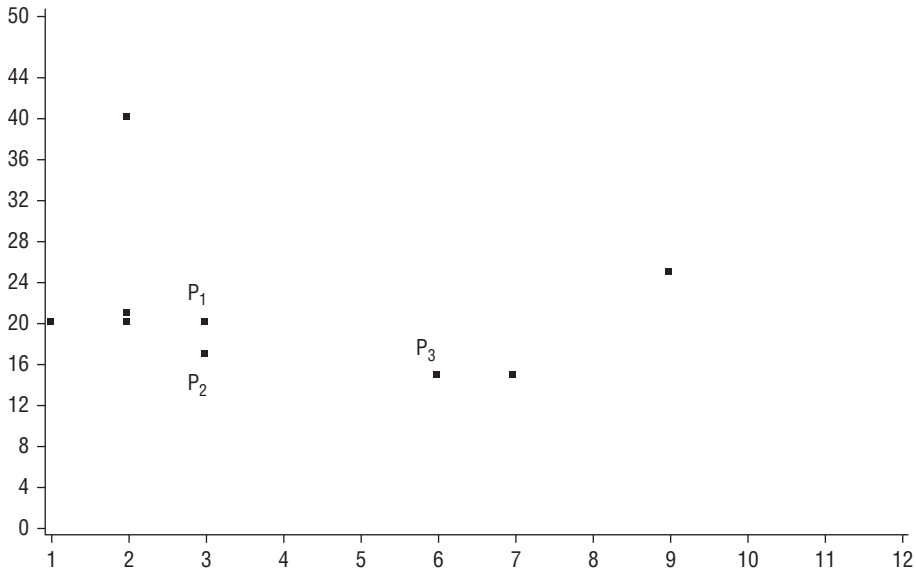


Figure 6-7: Distances between points in Scatterplot

The K-Means Algorithm

K-Means belongs to the class of clustering algorithms called *hard partitioning* algorithms because every data point falls into one partition (cluster) and one only. It is a simple algorithm logically and computationally, but is amazingly effective in creating data groups. It is also an iterative algorithm; it isn't guaranteed to converge, but nearly always does so. A practitioner will typically only have to specify the inputs to the model and how many clusters the algorithm should find; the algorithm takes care of the rest. The following steps describe only what the algorithm does, with the corresponding scatterplots illustrating the steps in Figure 6-8:

Initialization: Select the number of clusters to find. More sophisticated implementations of the algorithm will attempt a range in the number of clusters specified by the user and retain the single number of clusters that is considered best. Either way, the user must have an idea of the number of clusters that exist in the data or that are desired to be discovered in the data.

Step 1: Assign one cluster center per cluster. Most software implementations identify a random data point in the training data for each cluster and assign this point as a cluster center.

Step 2: Compute the distance between each cluster center and every data point in the training data.

Step 3: Assign a label to each data point indicating its nearest cluster center.

Step 4: Compute the mean value for every input in each cluster based. This is essentially like performing a SQL "group by" for each cluster and computing the mean for each input. This is the new cluster center.

Step 5: Repeat Steps 2 through 4 until cluster membership does not change.

Measuring Cluster Goodness of Fit

How can one determine if a cluster model is a good one? This question goes to the heart of why unsupervised learning is generally a more difficult problem than supervised learning. With unsupervised models, there is no counterpart to metrics commonly used in supervised learning, such as average error or classification accuracy. The metrics most commonly used in unsupervised learning are related to two areas: distances and interpretation. The latter is discussed in more detail in Chapter 7.

Ideally, a good cluster model will have relatively small *intra-cluster* distances and relatively large *inter-cluster* distances. Or stated another way, the clusters will be compact and far from each other. Compactness is usually computed using the same distance measures as were used in building the clusters. In the case of Figure 6-9, the data shows distinct clusters easy to identify visually.

The clustering algorithm, however, doesn't know how many clusters there are, but does the best it can to find the number of clusters specified by the user.

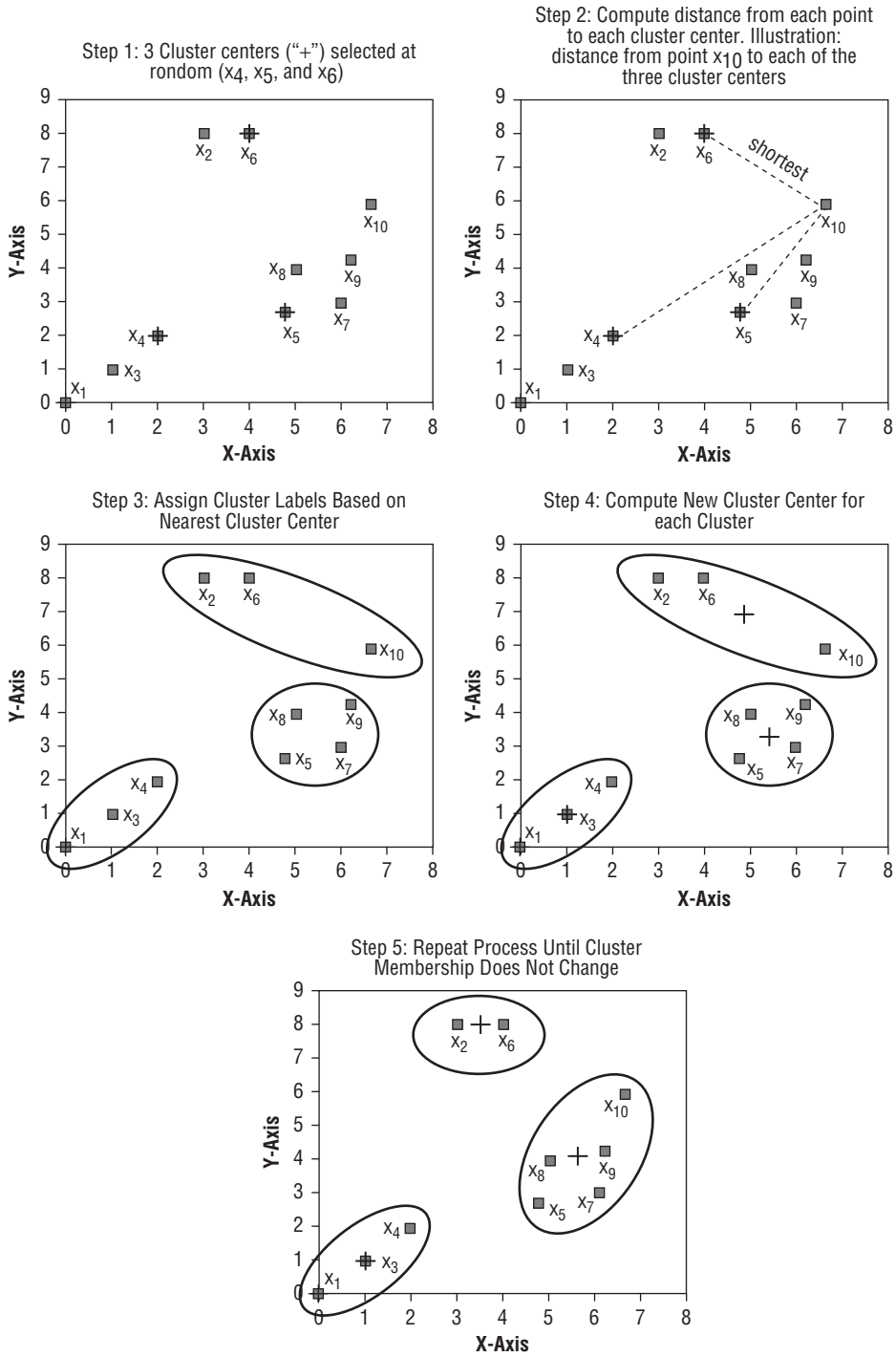


Figure 6-8: K-Means algorithm steps

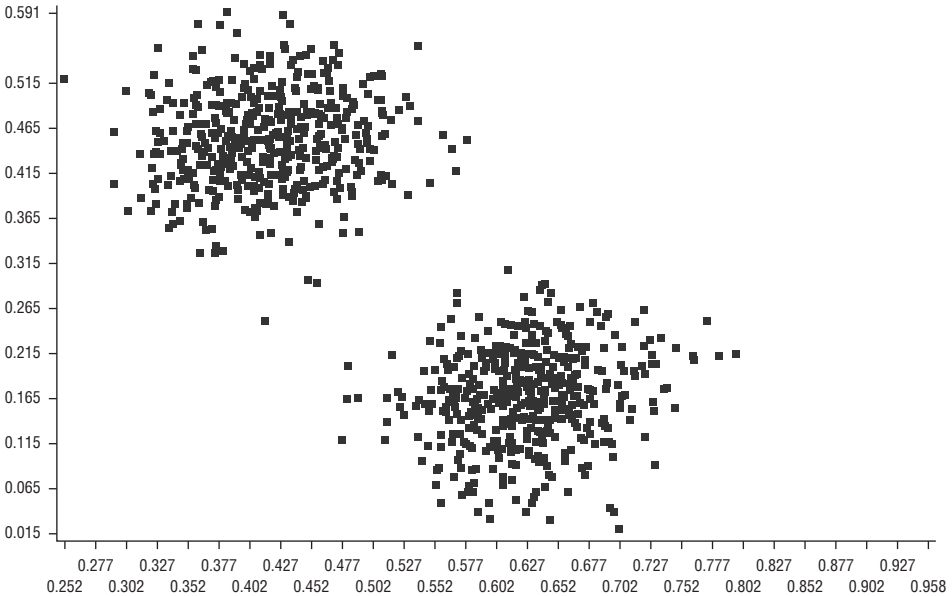


Figure 6-9: Scatterplot of data with two obvious groups

Sometimes the data doesn't support clean clustering of the data with the number of clusters specified by the user. Consider the data in Figure 6-10. A two-cluster model was created and the labels for the clusters is shown in Figure 6-11. Clearly, the cluster boundaries are imposed and not natural.

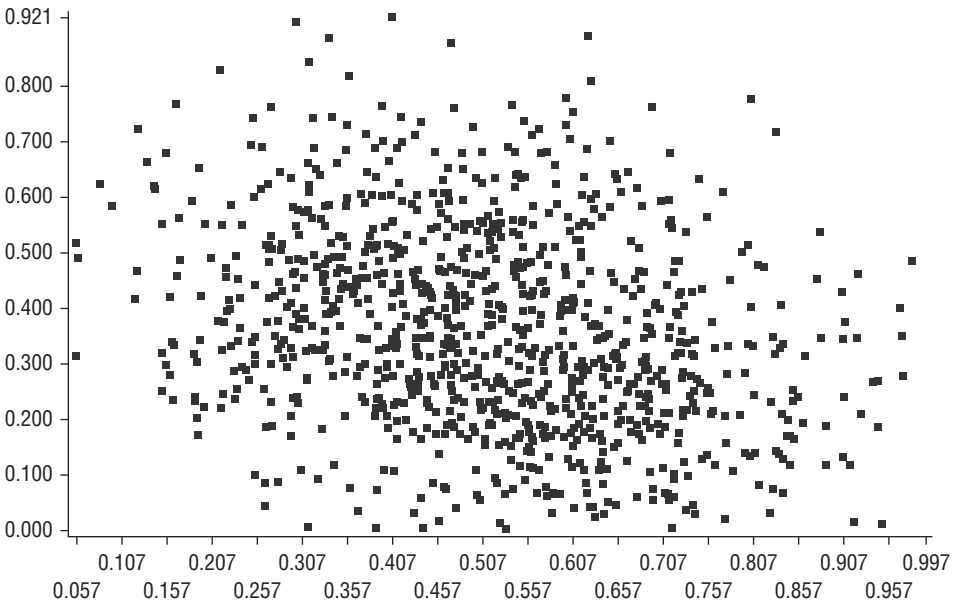


Figure 6-10: Scatterplot of data with one group

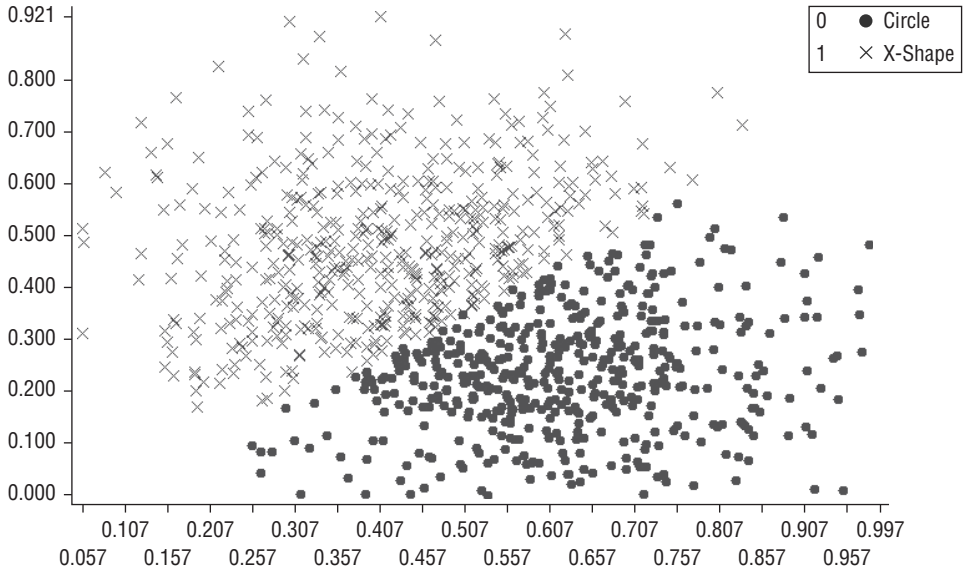


Figure 6-11: Cluster labels for forced two-cluster model

One of the most commonly used measures of cluster "goodness" is the Sum of Squared Error (SSE) metric, which is the sum of the differences in distance between each data point in a cluster and the cluster center (the mean). The SSE value for the 1,000 data points in Figure 6-11 is 40.9, whereas SSE for the 1,000 data points in Figure 6-9, clearly better suited for a K-Means clustering model, is 16.5, less than half the value.

Some software implementations also provide a measure of the distance between the clusters, usually a measure of how many standard deviations apart the cluster centers are, typically an average of the standard deviations of the inputs making up the clusters. In the case of Figure 6-9, the clusters are approximately 5 standard deviations apart, whereas for Figure 6-11, it is only 1.6. With values less than 2, you know there is considerable overlap not of the clusters themselves, but of the *spherical shape* the cluster model assumes. Figure 6-12 shows the 1 standard deviation ring around the cluster centers for the two clusters. Note too that, as is always the case with quadratic error functions (squared error), the decision boundary between clusters is linear.

Selecting Inputs

The examples shown so far are simple, with only two inputs to make visualization easier. However, the reality is that most often, clustering models are created from dozens to hundreds of candidate inputs. Ideally, you only include inputs that are helpful in creating clean clusters and no more.

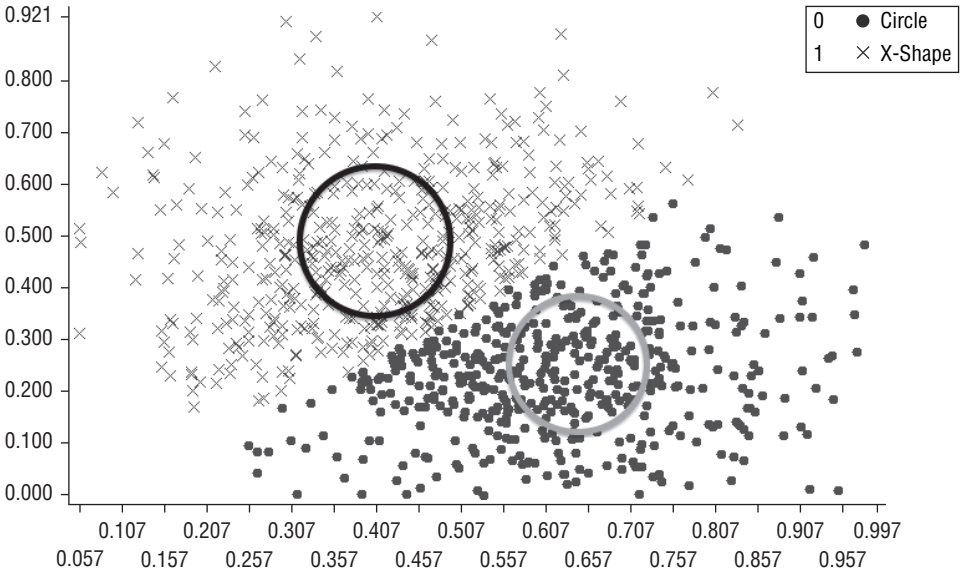


Figure 6-12: One standard deviation ring for each of two clusters

The description of PCA models emphasized their ability to find groups of variables with high correlation. K-Means cluster models are similar in several respects. First, both algorithms are based on variance and squared error metrics. Second, from a practical standpoint, both algorithms identify correlated variables. A good K-Means model identifies two or more variables whose correlation is high enough that the data clumps together. Examine Figure 6-9 (the good clustering model) again. When two variables are correlated, knowing one variable provides a strong indication of the value of the second variable. In this figure, when the x-axis value is 0.4, you have a strong indication that the y-axis is more than 0.4.

The difference between correlation analysis and K-Means models is that the K-Means cluster label creates a *subset* of data points that are correlated. In Figure 6-9, if the cluster in the upper left of the plot was instead centered on an x-value of 0.625 rather than 0.4, it would still be easy to cluster, but the conditioned correlation, the correlation between the x-axis and y-axis values in that cluster, is very high even though the global correlation between the x-axis and y-axis would not be high. In this sense, K-Means clustering models are multidimensional correlation detectors.

K-Means does not provide a mechanism for variable selection in the algorithm itself; the user must make the selection before building the model. Therefore, care must be taken to select good variables. But there is no simple way to find what these variables are. Chapter 7 describes some cluster interpretation methods to uncover which variables are strong influencers in creating the clusters.

Variable selection can be done through a combination of domain knowledge and assessment of the influence of variables in the clusters.

The *curse of dimensionality* problem has already been described, and it is certainly an issue with K-Means models; the more inputs included in the model, the greater the number of data points that are needed to fill the space. This is a key reason why most practitioners limit the number of inputs to dozens at most.

Data Preparation for K-Means

Careful attention to data preparation steps should be taken to build good K-Means models. Many of the data preparation steps follow the best practices described in Chapter 4. Others are specific to this kind of algorithm. If data preparation steps are not done properly, the clusters may be biased inadvertently toward just a few of the complete set input variables or may not be well formed.

Data Distributions

Data preparation for K-Means models is the same as has been described previously for numeric algorithms. The key corrective actions include:

1. Fill in missing data or remove records with missing values.
2. Remove or mitigate outliers through data transformations.
3. Reduce skew.
4. Explode categorical variables into dummy variables. Include categorical variables in this way only if necessary.
5. Scale all inputs so they are on the same scale through min-max normalization or z-scores.

These are all standard steps in building K-Means models. However, sometimes there are advantages to breaking the rules. First, as was described with PCA models, magnitudes can affect the models significantly, biasing the formation of clusters. Sometimes, however, this may be exactly what you want; if for some reason, you want some of the variables to drive cluster formation, one way to encourage this is to have their magnitudes larger than the other variables. We've done this on many occasions.

Second, categorical variables are problematic in computing distances. A variable encoded as 0 or 1 has only two distance values compared to another record: the minimum distance (0) or the maximum distance (1). Because of this curse of extremes, categorical variables, especially when there are large numbers of values for the categorical variable, can have more influence in the formation of clusters than any of the continuous variables. To mitigate this effect, first you can either ensure all variables are scaled to the range 0 to 1, or if the influence of

categorical variables is to be reduced further, you can z-score the numeric data so the range of their values is nearly always –3 to 3, a much larger magnitude and range of values than the dummy variables. A second way to reduce the influence is to recode the dummy variables from 0 and 1 to new values, such as 0.2 and 0.8.

Sometimes, to avoid the inherent conflict between clustering continuous and categorical data, one can bin the continuous data and create dummy variables from those bins so *all* the inputs are dummy variables. There are circumstances where this approach can be helpful, but because K-Means is more naturally a numeric algorithm, it should be treated as a secondary option or last resort.

Irrelevant Variables

Irrelevant variables, variables that do not contribute in any way to the formation of the clusters, create a noise floor in the distance calculation for every cluster. This may not have an effect in the actual cluster formation, but certainly will affect the assessment of how good the clusters are. If you are using the sum of square distances to measure cluster compactness, the more irrelevant variables that are included in the model, the larger the sum of square distance will be and therefore the worse the cluster will appear to be. These issues are addressed in Chapter 7.

In Chapter 4, two classes of input variables were identified that should be removed before beginning analysis: redundant variables and irrelevant variables. Clustering algorithms are particularly sensitive to both of these kinds of variables and therefore need to be considered prior to modeling.

When Not to Correct for Distribution Problems

K-Means clustering, as an algorithm that uses the Euclidean Distance, builds spherical cluster shapes. More precisely, K-Means builds *hyper-spherical* clusters because clustering is done in multidimensional space. Finding spherical clusters is more natural if the data itself is spherical, which translates to Normal distributions. Most classical treatments of K-Means clustering recommends that data be transformed so that it is normally distributed or at least nearly so. At a minimum, I recommend that the data be uniformly distributed.

How much deviation from Normal or Uniform is too much? There is no rule to tell us how much is too much. Modelers should examine clusters to make sure the most important variables to the formation of clusters are not selected primarily because they have strange distributions or severe outliers.

Selecting the Number of Clusters

Determining the number of clusters, the value of "k," is an inexact science. The selection of k is required to build the model and as has already been described, some software implementations try several values of k and make a judgment to determine the best number of clusters from the models that were created.

There are general guidelines you can use to determine the value of k. First and foremost, k is often determined by business considerations. If the marketing department is interested in building customer segments for the purpose of creating customized marketing literature for a new up-sell campaign, they may determine that four and only four customer segments are needed. A four-cluster model would therefore be created, identifying the best way to segment the customers into four groups. When cluster interpretation is highly important, the number of clusters in the model is usually kept small, perhaps 10 or fewer.

On the other hand, clustering can be used to identify anomalies in the data as well. Outliers and anomalies can be identified in two different ways in clustering. First, if one builds relatively small numbers of clusters, the cluster centers will represent the big picture: Where are the centers of the main group in the data? However, because every record *must* be included in one and only one cluster, some data points, the outliers, will be relatively far from the cluster center to which they have been assigned. This kind of multidimensional outlier detection is straightforward: Flag records more than 3 or more standard deviations from the cluster center.

A second way to identify multidimensional outliers, especially when one wants to find patterns of anomalies, is to build lots of clusters, perhaps hundreds. Some of the clusters will have one or two records in them; this merely indicates a true outlier that is unlike anything else in the data. The more interesting clusters may be those that have 0.1 to 1 percent of the total number of records. In a data set with 10,000 records, these will be the clusters with 10 to 100 records. What is particularly interesting about these outliers is that they are unusual, but unusual in a group; a pattern of anomalous behavior is represented in the cluster. In fraud detection problems, for example, these may be interesting cases to investigate, perhaps indicating a new trend of fraud not previously identified.

These methods for determining the number of clusters are based on business concerns. There is no optimal number of clusters one can identify numerically. However, there are empirical ways to identify how many clusters occur naturally in the data compared to other numbers of clusters using SSE. Consider Table 6-9, which shows SSE for eight different K-Means clustering models, ranging from 3 to 250 clusters.

Table 6-9: Sum of Squared Errors vs. Number of Clusters

NUMBER OF CLUSTERS	SSE	% REDUCTION	SSE * # CLUSTERS
3	80,224	0.0	240,673
9	49,641	–38.1	446,768
15	35,251	–56.1	528,764
31	17,722	–77.9	549,368
51	10,881	–86.4	554,934
71	7,678	–90.4	545,122
101	5,691	–92.9	574,782
250	2,694	–96.6	673,384

Measuring SSE is sometimes used to select the number of clusters in much the same way the Scree Plot is used for PCA: You can find a knee in the plot to determine the best number of clusters. The data in Table 6-9 is plotted in Figure 6-13, and the knee appears to be somewhere between 70 to 100 clusters, once the reduction in SSE is greater than 90 percent.

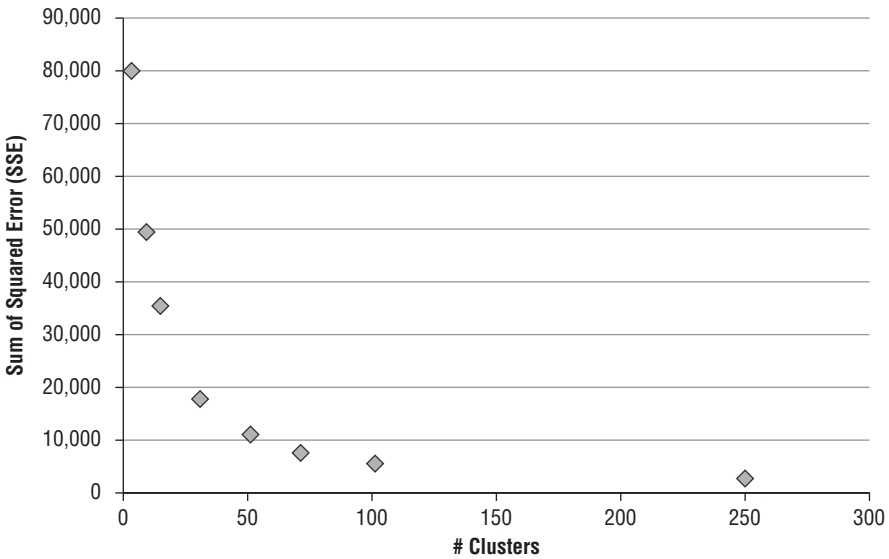


Figure 6-13: SSE vs. # clusters to find the "knee"

As the number of clusters increases, the reduction in SSE is expected; the more clusters in the model, the fewer data points on average will be in the clusters and therefore the more compact the clusters will be. One way to compensate for the natural decrease in the size of SSE is to multiply it by the number of clusters. When one does this, this new measure flattens out by 31 clusters, although it increases again at 250. Again, this is not a theoretical result, just another empirical way to try to identify the best number of clusters.

The Kinds of Clusters K-Means Finds

K-Means clustering typically finds spherical clusters when it uses Euclidean distance to measure distances between data points. Euclidean distance produces linear boundaries between clusters when there is overlap between them. Figure 6-14 shows seven clusters found from only Band1 and Band9 of the nasadata data set. Each color represents a cluster label. Note that the transition between the cluster regions is always linear and that each region is a contiguous group; no cluster region is broken up by another, inside another, or has a serpentine shape.

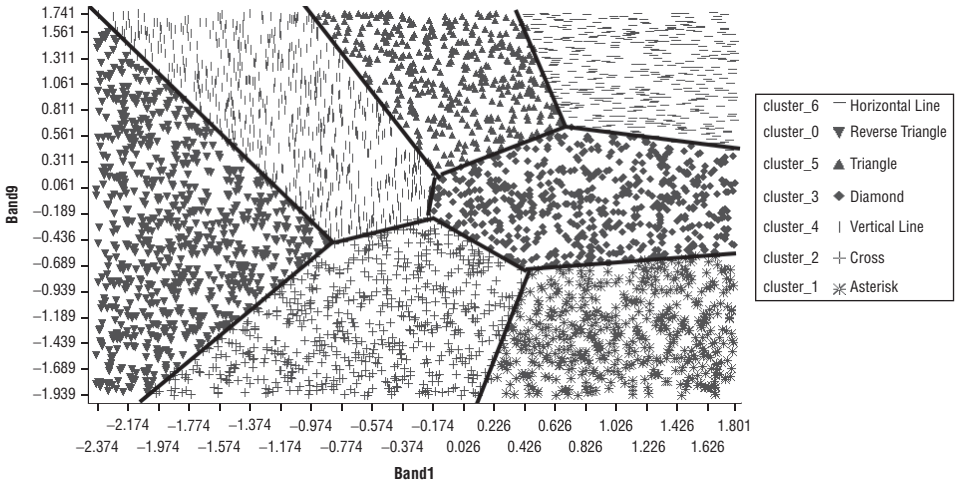


Figure 6-14: Cluster regions from Euclidean distance

On the other hand, data that is not spherical is difficult for K-Means, with the algorithm imposing clusters that don't make sense visually. Consider a function called the *sombrero function*. The part of the sombrero function with a target variable equal to 1 is shown in Figure 6-15.

The K-Means clustering algorithm cannot find two clusters for this data set that make sense. Figure 6-16 shows models for 2, 5, and 8 clusters. The 2-cluster model just splits the data down the middle—not very helpful. The 5-cluster model begins to divide the inner data clump from the outer ring, but still keeps portions of the inner circle mixed with the outer ring; it appears to segment the data by wedges. The 8-cluster model is finally able to split the inner circle from the outer ring, although to do so, it needs two clusters to represent the inner circle and six clusters to represent the outer ring. A post-processing step would be needed to group the clusters if the true underlying data structure is to be learned from the data.

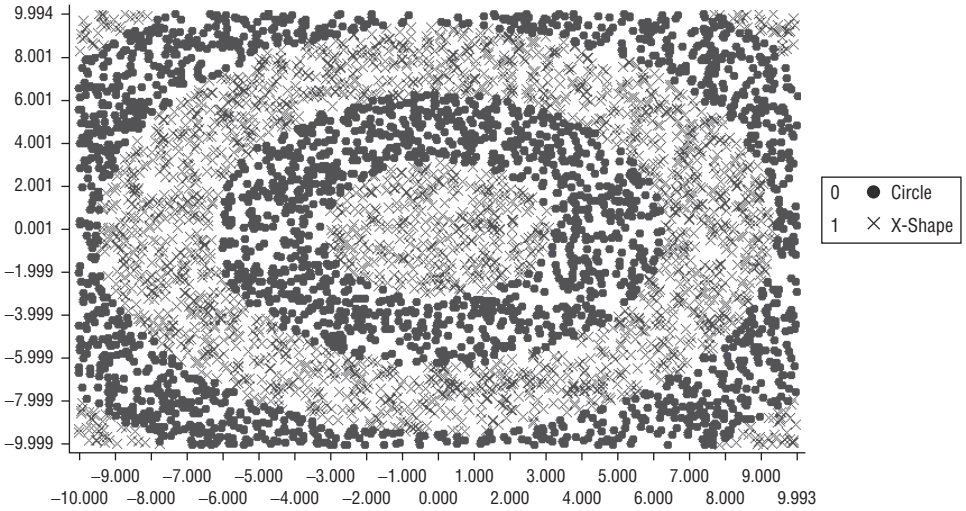


Figure 6-15: Sombrero function

This is not to say that K-Means clustering cannot be used for data that isn't spherical. Even though the clusters found in Figure 6-16 are not spherical and do not form spherical groups, they nevertheless form interesting groups that still provide insight into the structure of the data.

Consider again the sombrero data. After applying the cluster models shown in Figure 6-16 to randomly generated data, the cluster regions are clearer, as can be seen in Figure 6-17. The 2-cluster regions in Figure 6-17 reveal nothing new, but the 5- and 8-cluster models reveal interesting groupings. The 5-cluster model has five equal-sized regions. However, the 8-cluster model separates the center region of the sombrero function into its own cluster, and divides the outer ring into seven equal-sized wedges. These groupings, while not natural groupings, are nevertheless interesting groupings. As always with K-Means clusters using Euclidean distance, the boundary between pairs of clusters are linear.

Other Distance Metrics

Euclidean distance is by far the most commonly used distance metric used in building the K-Means clustering model. Another metric that could be used and is sometimes available is the Mahalanobis distance, which normalizes the distance by the covariance matrix, thus eliminating the need to scale the inputs to a common range. Another way to describe the Mahalanobis distance is that it transforms the inputs so that in a scatter plot of every pair of variables, the data is centered on 0 in both directions, have roughly the same range, and will be shaped like a circle. Using the Mahalanobis eliminates the need to correct for variable scales in the data.

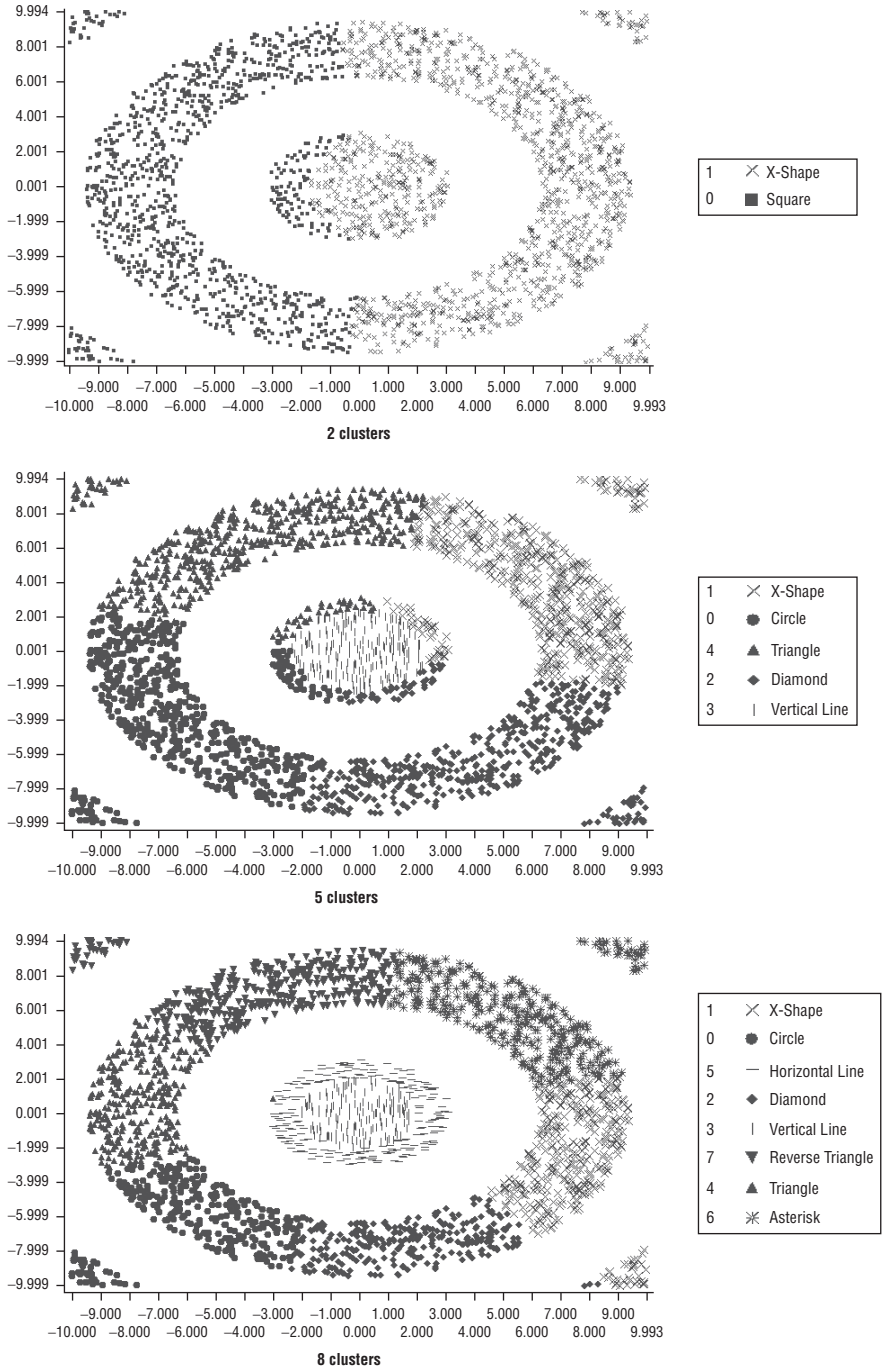


Figure 6-16: 2-, 5-, and 8-cluster models for the sombrero function

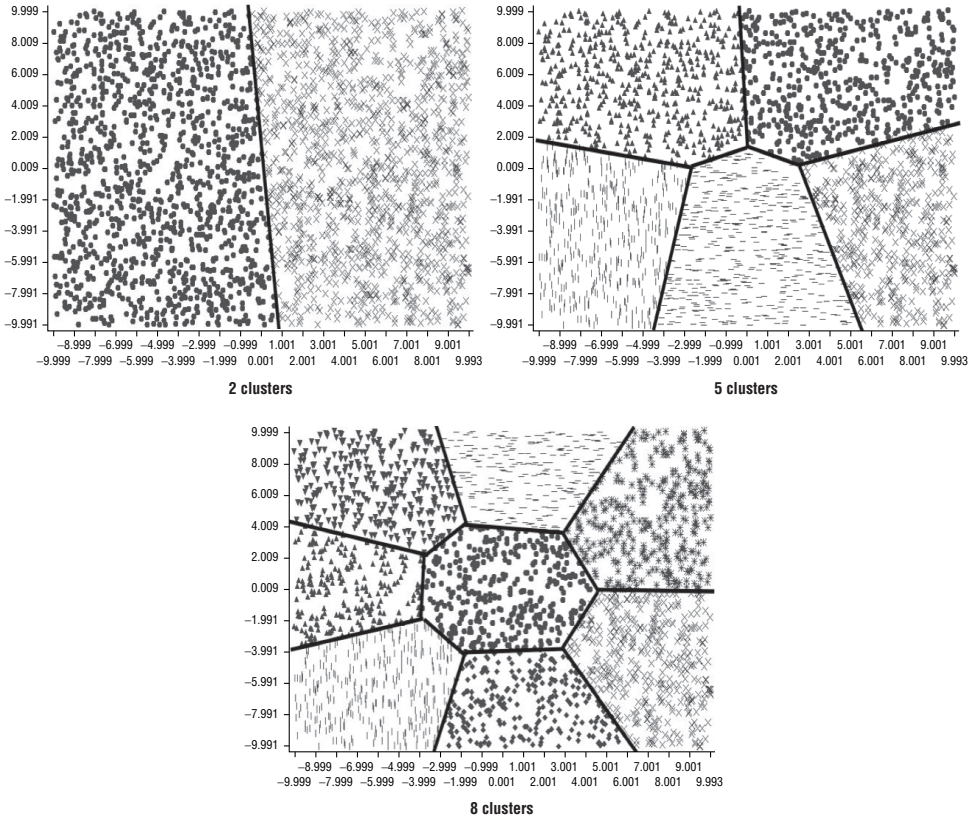


Figure 6-17: Application of 2-, 5-, and 8-cluster models to random data

Another distance metric sometimes included in software is the Manhattan distance, also known as the "city block" distance. Mathematically, this metric measures the absolute value of the difference between points rather than the square of the difference. It is rarely used in practice, but could be an interesting alternative for some applications.

Some software also includes the Minkowski distance, which isn't a single distance metric but rather is a generalization of Euclidean distance, parameterized by r. When r is equal to 2, the Minkowski distance is exactly the Euclidean distance which squares the distances between data points. When r is equal to 1, the Minkowski distance is the Manhattan distance. However you aren't limited to setting r equal to 1 or 2. The larger the value of r, the more the Minkowski distance becomes simply the value of the maximum absolute difference between

data points in any single dimension. This is called the L-infinity norm or the Chebyshev distance and can be used to build clusters that minimize the worst-case distance.

Kohonen SOMs

The Kohonen Self-Organizing Map (SOM) algorithm is the most well-known unsupervised learning neural network algorithm, and perhaps the second in popularity only to the multilayered perceptron. They are known by a variety of names, such as SOMs, Kohonen Maps, Kohonen Networks, Feature Maps, and more. Predictive analytics software places Kohonen Maps in a variety of categories: Some put them in the grouping of clustering algorithms, others in the neural network algorithms, and some even put them in a category unto themselves.

The connection of Kohonen Maps to neural networks is due to a few algorithmic similarities with other neural networks. First, the algorithm updates weights during training from a single record at a time. Second, the visual representation of the network looks like other kinds of neural networks; it has nodes and connections between nodes. Third, conceptually, a Kohonen Map is considered to be a *learning algorithm*. It can be updated without retraining completely by starting from the weights of a current model rather than from new, randomly generated weights.

Kohonen Maps are often described as projections of inputs onto a two-dimensional space, like the one shown in Figure 6-18, a 4×4 lattice of the inputs. The terms lattice and map are used interchangeable with Kohonen networks. The number of nodes in the SOM corresponds to the number of clusters the map is representing in the model. For the 4×4 map, there are 16 nodes, which therefore provide a 16-cluster representation of the data. If the number of inputs is small, the SOM can identify multi-modal patterns in the data by exploding the original inputs into additional dimensions. On the other hand, if the number of inputs is large, much greater than 16 for the 4×4 map, the SOM creates a multidimensional scaling of the inputs down to two dimensions.

Figure 6-19 shows a 6×1 map. Note that while the map is still two-dimensional, the connections occur in only one dimension. In general, while a 6×1 map and a 3×2 map have the same number of nodes (6), they will produce different models because of the nature of the connections. The largest neighborhood distance from one end to the other of the 6×1 map is 5 units away (from the left end to the right end), whereas in a 3×2 map, the largest neighborhood distance is only 3: up one and over 2 from the lower-left corner.

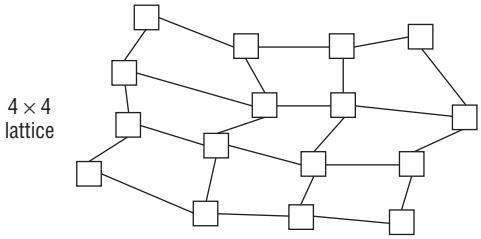


Figure 6-18: Kohonen Self-Organizing Map, 4 × 4

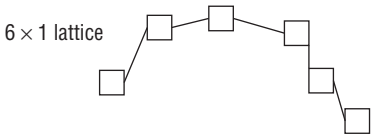


Figure 6-19: Kohonen Self-Organizing Map, 6 × 1 Map

The Kohonen SOM algorithm doesn't have to be represented as a two-dimensional map; the algorithm supports any representation, including 1, 2, 3, or more dimensions in the map. Most often, undoubtedly because paper and screens are two-dimensional, the map is also shown in two dimensions.

The maps in Figures 6-18 and 6-19 are rectangular: The nodes are connected in a rectangular grid so each node has at most four other nodes connected to it. Some implementations of the algorithm choose a hexagonal grid, where six nodes are connected to each node.

The Kohonen SOM Algorithm

SOMs are iterative algorithms like other neural networks, operating on one record at a time and then updating weights based on that single record. The specific algorithm steps are as follows:

Initialization Step 1: Select the SOM architecture, typically the length and width of the map.

Initialization Step 2: Scramble the order of the records in the training set so that the order of presentation of the records is random.

Initialization Step 3: Initialize all weights in the map to random values.

Step 1: Pass one record through the network, computing the difference between the inputs in the record and the weight values. The distance is the Euclidean distance between the record and the weights.

Step 2: Identify the winning node in the SOM (the node closest to the data point). Adjust the weights of that node proportionally to the distance between the weights and the record.

Step 3: Identify the neighbors to the winning node. Adjust the weights of the neighboring nodes proportionally to the distance to the weights of the record, but less than for the winning node.

Step 4: Repeat Steps 1–3 for all the records in the training set. The size of the neighborhood influenced by each record shrinks as the algorithm proceeds until eventually only the winning node is adjusted.

Step 5: Continue training until a stop condition applies, such as a time or a maximum number of passes through the data.

Some implementations of SOMs update the weights of the neighboring nodes by a fixed amount depending on how many units away the node is from the winning node, essentially based on the Manhattan distance to the winning node. Other implementations update the weights of the neighboring nodes proportional to the distance from the winning node; the closer the neighbor is to the winning node, the more it is influenced to update in the direction of the record.

The neighborhood idea is key to the success of the algorithm. Without any neighborhood influence, the SOM would be essentially the same algorithm as the K-Means algorithm. By allowing for communication between node locations, the SOM can often position the centers of the nodes in better locations that reflect the relative density of data in each region of the input space.

Sometimes Kohonen Maps will create node memberships that alternate between large number of records in the node and nearly no records in a neighboring node: dead nodes. This can continue to the point where the map essentially uses only half the nodes specified because the others have 0 or nearly 0 records. For example, using the 4×4 map of Figure 6-18, Figure 6-20 indicates significant populations of records in a node by a darker box, and nearly no records in a node by a white box.

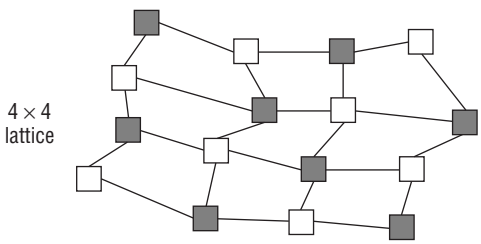


Figure 6-20: Kohonen Map with dead nodes

While this effect is not an indication of a model that hasn't converged properly, it can be disconcerting to the modeler. If one truly would like to have 16 clusters in the map, the size of the map may have to be increased, even doubled. Algorithmically, there is a fix that can be found in software on rare occasions, sometimes called a "conscience" mechanism, which penalizes a node for getting too large and therefore making nearby nodes more likely to pick up additional records.

Kohonen Map Parameters

The Kohonen Map has a few parameters that can be modified to affect the model it trains. Typical parameter settings include:

- # Nodes in X Direction
- # Nodes in Y Direction
- Max # Iterations
- Learning Rate
- Learning Decay Rate
- Random Seed for Weight Initialization
- Type of Neighborhood (rectangular or hexagonal)
- Neighborhood Influence (# hops)

At a minimum, implementations in software will allow selection of the number of nodes in the model (x and y directions, or sometimes listed as rows and columns) and maximum iterations (or maximum time).

Visualizing Kohonen Maps

Kohonen Maps are often described as aids for data visualization. If there are *N* inputs to the Kohonen Maps, and the map itself is expressed in two dimensions, such as a 3×3 map, then the model can be interpreted as projecting the *N* dimensional space onto a two-dimensional plane.

For example, consider a 3×3 Kohonen Map trained from the nasadata data set using all twelve inputs, Band1 through Band12. A 3×3 map contains therefore nine clusters that can be arranged in a two-dimensional grid. Figure 6-21 shows this 3×3 grid with the mean values for four different inputs to the model overlaid on top of the grid: Band1, Band5, Band10, and Band12. Along with the mean values, a color code has been applied to each node: green for larger positive mean value and red for larger negative mean values. Because all inputs to this model have been z-scored, the mean in each node is a measure of the number of standard deviations from the input's mean.

The grid labels are 1, 2, and 3 in each direction (x and y). The order of these labels matters because of the connectedness of the Kohonen Map. This map is a rectangular grid.

Often, the most interesting aspect of these visualizations is the gradient of the mean values. Each of the four variables has relatively smooth and monotonic slopes of mean values through the map. Band1 becomes larger and more

positive for *smaller* values of Kohonen Y, whereas Band12 is more positive for *larger* values of Kohonen Y.

A second way to look at the maps is to compare the relative values of the inputs in the same node. Kohonen Map coordinate (1,1) is positive for Band1, Band5, and Band10 but negative for Band12. Coordinate (3,1) is positive for Band1 and Band12, negative for Band10, but neutral for Band5.

These four maps also indicate that each of the four inputs is used in the formation of the node locations. If one of the inputs has a mean value close to 0 in all nine nodes, the input therefore has no influence in the formation of the nodes and can be safely discarded from the model.

Another method of visualizing the Kohonen nodes is to plot the 3×3 grid with the relative distance between nodes represented by the length of the line connecting the nodes. Some more sophisticated implementations of the Kohonen Map algorithm will identify when a map has "wrapped on top of itself," as shown in Figure 6-22. In this situation, nodes in the Kohonen Map have connections that cross each other, indicating that initial nodes that were originally neighboring each other have moved past one another during training. Some practitioners consider these maps to be inferior models and will discard them. A new random initialization of weights is sometimes all that is needed for the map to be rebuilt and converge properly.

BAND 1		Kohonen X		
		1	2	3
Kohonen Y	1	0.840	0.922	1.102
	2	–0.322	–0.146	0.370
	3	–1.285	–1.463	–0.275

Band 5		Kohonen X		
		1	2	3
Kohonen Y	1	1.215	0.869	0.079
	2	0.424	–0.025	–0.453
	3	–0.366	–0.908	–1.508

Band 10		Kohonen X		
		1	2	3
Kohonen Y	1	0.746	0.033	–1.309
	2	0.992	–0.213	–1.396
	3	0.992	0.063	–1.233

BAND 12		Kohonen X		
		1	2	3
Kohonen Y	1	–1.513	–0.125	0.810
	2	–0.299	0.112	0.637
	3	0.609	0.885	0.685

Figure 6-21: Overlaying mean values of variables on top of the Kohonen Map

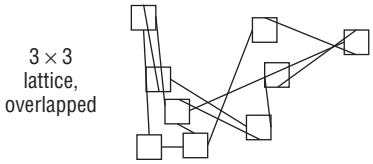


Figure 6-22: 3 × 3 Kohonen Map overlapped on itself

Similarities with K-Means

Kohonen Maps have many similarities to K-Means clustering models, many of which are described in Table 6-10. Data preparation steps for Kohonen Maps are very similar to those done for K-Means, including missing data handling, use of numeric data, and the need for input magnitude normalization.

The similarities break down, however, with parameter updating. The training algorithm for Kohonen Maps proceeds record-by-record, but more importantly, it isn't "winner take all"; nearby nodes to the node closest to a record are also adjusted. In K-Means clustering, the statistics of each cluster are defined solely on cluster members (the records nearest to its cluster center); nearby clusters are not affected.

Table 6-10: Comparison of Kohonen SOMs and K-Means

CHARACTERISTIC	KOHONEN MAP	K-MEANS
# Nodes/Clusters	Number of nodes pre-determined by length and width of map	Number of clusters pre-determined.
# Inputs in Model	Number of inputs fixed at beginning of training, no variable selection	Number of inputs fixed at beginning of training, no variable selection
# Model Parameters	Nodes defined by weights, one per input	Clusters defined by mean value, one per input
Model Initialization	Random initialization of all weights	Random selection records to be cluster centers
Cluster Membership and Distance Metric	Node membership defined by minimum Euclidean distance from record to node weights	Cluster membership defined by minimum Euclidean distance from record to cluster mean

Parameter Updating	Weights changed for node closest to one record at a time and repeated for all records; nearby nodes also modified	Mean values for each cluster updated based on all data at once, independent of other clusters
Positioning of Clusters/ Nodes	Inter-Node distances are direct result of training—indicate data distances	Inter-cluster distances are determined indirectly

How similar Kohonen Maps appear compared to K-Means models varies widely; sometimes for the same number of clusters and nodes, the models are nearly identical, while in other situations they bear little resemblance to one another. Consider Kohonen Maps as a separate but related algorithm to K-Means.

A key metric that measures how well K-Means models fit the data is the SSE measure, which computes the average distance between the records in a cluster and the center of a cluster. Kohonen Maps could take advantage of this same metric to help in determining the best number of nodes to include in the map. In Kohonen Map terminology, SSE is computed by calculating the distance between the records in a Kohonen node and the weights of that node. However, this measure is rarely if ever computed.

Summary

Every predictive modeler should become adept at building descriptive models, whether the models are deployed or used primarily to understand the data better. Most predictive modeling software includes Principal Component Analytics and K-Means algorithms; analysts should learn these algorithms to include in their repertoire. Some tools also contain Kohonen Self-Organizing Maps, which are built and assessed similarly to K-Means. Pay particular attention to the input data to ensure the clusters are formed well; data preparation for these three algorithms is very important.

CHAPTER

7

Interpreting Descriptive Models

Once a descriptive model has been built, the next task of predictive modelers is to interpret the meaning of the model. Most software provides summaries of the clusters in tabular or graphic format. The typical summaries provide helpful information to interpret the meaning of the clusters, most typically the cluster center, but fall short of explaining why the clustering algorithms formed the clusters.

This chapter describes approaches you can take to gain insights from clustering models. We focus on three aspects of interpretation: describing clusters, describing key differences between clusters, and translating variable values from normalized values back to their original units. The example in the chapter is based on a K-Means model, but the techniques apply to Kohonen SOMs just as well.

Standard Cluster Model Interpretation

Model summaries for K-Means clusters always contain counts of records in the training data that fell into each cluster and the mean value of every variable for each cluster. Sometimes the standard deviation of each variable in each cluster is also included. For Kohonen SOMs, sometimes only the count of records that fell into each node is included in the reports.

Even if the software does not provide summaries, generating summary statistics is straightforward: First group by cluster ID, and then for each cluster compute the statistics of interest, most often counts of records in each cluster, the minimum, maximum, mean, and standard deviation for each variable in the cluster. The mean value for each variable in each cluster is, by definition, the cluster center. If you also include the standard deviation of each variable in each cluster, evaluating the spread of each variable furthers your understanding of the characteristics of each cluster. If the standard deviation is relatively small, the variable has a compact range in the cluster.

For example, consider a K-Means clustering model containing 11 input variables and their data types, listed in Table 7-1. The dummy variables, originally having values 0 and 1, have been scaled to values 0.3 and 0.7 to reduce their influence in the clustering model. This practice is not usually done in software, but can be helpful to reduce the bias that sometimes occurs with dummy variables. The ordinal, interval, and ratio variables have all been scaled to the range from 0 to 1.

Table 7-1: Variables Included in a 3-Cluster Model

VARIABLE	VARIABLE TYPE
D_RFA_2A	Dummy
E_RFA_2A	Dummy
F_RFA_2A	Dummy
RFA_2F	Ordinal
DOMAIN1	Dummy
DOMAIN2	Dummy
DOMAIN3	Dummy
FISTDATE	Interval
LASTDATE	Interval
LASTGIFT_log10	Ratio
NGIFTALL_log10	Ratio

After building a 3-cluster model with the K-Means algorithm, Table 7-2 shows the mean values of these 11 variables in the clusters: These are the cluster centers. The relative values of each of the cluster centers can be compared to the global, overall statistics. Clusters 1 and 2 have a higher than average number of gifts given (NGIFTALL), higher than average number of gifts given in the past year (RFA_2F), higher than average most recent gift given (LASTDATE), but lower than average size of the last gift (LASTGIFT). In other words, Clusters 1 and 2

contain donors who give more often but in lower amounts than average. Cluster 3 is the reverse: These donors give larger amounts but less frequently.

Table 7-2: Cluster Centers for K-Means 3-Cluster Model

VARIABLE	CLUSTER 1	CLUSTER 2	CLUSTER 3	OVERALL
# Records in Cluster	8,538	8,511	30,656	47,705
LASTDATE	0.319	0.304	0.179	0.226
FISTDATE	0.886	0.885	0.908	0.900
RFA_2F	0.711	0.716	0.074	0.303
D_RFA_2A	0.382	0.390	0.300	0.331
E_RFA_2A	0.499	0.500	0.331	0.391
F_RFA_2A	0.369	0.366	0.568	0.496
DOMAIN3	0.449	0.300	0.368	0.370
DOMAIN2	0.300	0.700	0.489	0.493
DOMAIN1	0.515	0.300	0.427	0.420
NGIFTALL_log10	0.384	0.385	0.233	0.287
LASTGIFT_log10	0.348	0.343	0.430	0.400

Many predictive analytics software packages provide reports in the form of histograms of each variable by cluster. For example, Figure 7-1 shows a histogram of RFA_2F. It is clear from this figure that Cluster 3 is different from the other two, and that RFA_2F contributes to this difference.

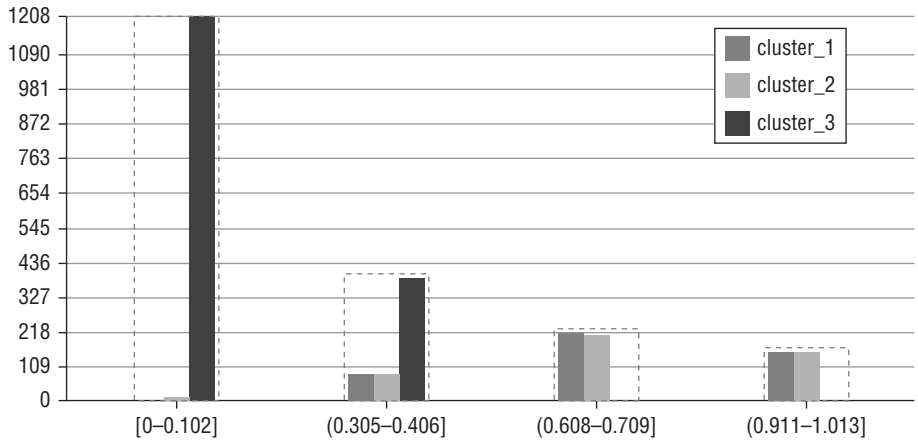


Figure 7-1: RFA_2F stacked histogram to interpret clusters

Problems with Interpretation Methods

There are significant problems with using the procedure described so far as a way to interpret cluster models. First, if you have normalized the data prior to clustering or building the models, all of the summaries are in the normalized units and may not be readily understood. Second, the summary statistics tell you what each cluster is like, but not how the clusters differ, and therefore why the clusters were formed as they were.

Normalized Data

Chapter 6 described the common reasons for normalizing data prior to clustering. Table 7-2 shows the mean values of the normalized data. For instance, RFA_2F has mean values of 0.711, 0.716, and 0.074 for Clusters 1, 2, and 3 respectively. RFA_2F had four values prior to normalization: 1, 2, 3 and 4. After normalizing to a range between 0 and 1, these four values become 0, 0.333, 0.666, and 1.0. Therefore, if the values of a normalized RFA_2F in Cluster 3 are close to 0, this corresponds to values that are predominantly 1 in the unnormalized units. On occasion in Cluster 3, RFA_2F values are equal to 0.333, which corresponds to the value 2 in unnormalized units.

Continuing with RFA_2F, you can see that Clusters 1 and 2 have values of RFA_2F primarily equal to 3 and 4, the more frequent donors. The same argument can be made for dummy variables such as DOMAIN1, DOMAIN2, and DOMAIN3. These have been scaled to values 0.3 and 0.7, and therefore Cluster 2, DOMAIN1, and DOMAIN3 are always equal to 0.3 in normalized units, which corresponds to the value 0 in unnormalized units; Cluster 2 has no members with DOMAIN3 or DOMAIN1 populated. Dummy variables and simple ordinal variables are therefore easy to convert from normalized units back to unnormalized units.

However, more complex ordinal variables, interval variables, and ratio variables become more problematic to interpret. This becomes even more complicated when additional transformations are applied prior to the scaling step; in this data, NGIFTALL and LASTGIFT are transformed by the log base 10 operation prior to scaling to the range 0 to 1. (LASTGIFT is the donation amount given the last time the donor gave.) From Table 7-2, we know that the mean LASTGIFT_log10 value in Cluster 3 is 0.430. So how much money on average was donated in the last gift for donors in Cluster 3? You cannot figure this out without knowing the scaling factor that transformed LASTGIFT_log10 to a number between 0 and 1. After doing the math to convert the normalized units to unnormalized units, the mean LASTGIFT_log10 value in Cluster 3 is $20.40. Interpreting clusters based on the normalized units is therefore problematic, requiring additional work to identify the actual values of the cluster centers.

Within-Cluster Descriptions

Describing the mean of a cluster tells us what that cluster looks like, but tells us nothing about why that cluster was formed. Consider Clusters 1 and 2 from Table 7-2. The mean values describe each of the clusters, but a closer examination shows that nearly all the variables have means that are similar to one another (we will see how significant these differences are after computing ANOVAs). The only three variables that, after visual inspection, contain differences are DOMAIN1, DOMAIN2, and DOMAIN3. The differences are key: If the purpose of the cluster model is to find distinct sub-populations in the data, it is critical to not only describe each cluster, but also to describe how they differ from one another.

Examining the differences between cluster characteristics provides additional insight into why the clusters were formed. However, determining how the clusters differ can be quite challenging from reports such as the one shown in Table 7-2. Good visualization of the clusters can help. But whether you use tables or graphs, identifying differences requires scanning *every* variable and *every* cluster. If there are 20 inputs and 10 clusters, there are 200 histograms or summary statistics to examine.

Identifying Key Variables in Forming Cluster Models

The question "how do the clusters differ from one another?" can be posed as a supervised learning problem. After a cluster model is built, each record is placed in one and only one cluster. Now each record has an additional column in the data: the cluster label column. If the cluster label is identified as the target variable and the same inputs used to build the cluster model are inputs to the supervised learning model, you can determine which inputs are the best predictors of the cluster label from the supervised model.

There are several supervised learning methods that can help differentiate the clusters, but the purpose of the model is interpretation not accuracy *per se*; we want to use methods that emphasize interpretation. Three techniques for cluster model interpretation are described here: ANOVA, hierarchical clustering, and decision trees. These come from different disciplines and can describe different characteristics of the cluster models. They should therefore be viewed as complementary rather than replacements for one another.

There is one additional issue to consider when building supervised models for cluster model interpretation. The single column containing the cluster label has as many levels as there are clusters. Table 7-2 shows the result of a model with only three clusters. However, there can be dozens or even hundreds of clusters in the model, and therefore there can be dozens or hundreds of levels in the cluster label column. A single model intended to differentiate dozens to

hundreds of variables can be problematic for supervised learning methods, particularly for the ANOVA approach.

One approach to overcoming this problem is to explode the multi-level target variable into dummy variables, one per cluster, and then build a separate model for differentiating each cluster from all the others. This adds to the complexity of the analysis, but can yield insights that are hidden in models that try to differentiate all levels simultaneously.

ANOVA

Running an ANOVA is a simple and effective way to determine which variables have the most significant differences in mean values between the clusters. The ANOVA grouping variable is set to the cluster label, and the inputs to the cluster model are used as the test columns. Fortunately, the typical data preparation needed prior to computing an ANOVA is the same preparation already done for clustering, so no new data preparation is typically needed. The ANOVA computes an *F*-statistic that measures the differences in mean values between the clusters and a *p*-value indicating the significance of the differences in mean values.

The ANOVA method works best when there are relatively few clusters. The more clusters in the model, the less likely that inputs will have significantly different values for all clusters; often, a few clusters will have values of a variable that differ from the other clusters but in most clusters there won't be much variation at all. This effect will suppress the *F*-statistic and could obscure the significant role of a variable in the cluster models if the mean value of a variable is the same for most of the clusters and only varies for a few.

For example, consider the 3-cluster model described in Table 7-2. Table 7-3 shows the *F*-statistic and *p*-value for each of the variables, sorted by significance. RFA_2F has the largest *F*-statistic by far, and therefore is considered the most important variable in forming the clusters. The least significant variables are the DOMAIN1, 2, and 3 dummy variables. One note on significance should be emphasized. The fact that the *p*-values for all of the variables is 0 or near 0 can be indicative of the size of the data set as well as the differences in mean values.

Table 7-3: ANOVA Interpretation of Cluster Variable Significance

VARIABLE	*F*-STATISTIC	*P*-VALUE
RFA_2F	111630.5	0
D_RFA_2A	21187.9	0
LASTGIFT_log10	10435.7	0
E_RFA_2A	8499.3	0

NGIFTALL_log10	7664.8	0
F_RFA_2A	6813.5	0
LASTDATE	3313.3	0
FISTDATE	1073.4	0
DOMAIN1	103.0	0
DOMAIN2	36.6	1.11E–16
DOMAIN3	21.7	3.84E–10

Further insights into the key variables driving cluster formation are then obtained through visualization of these variables through histograms and scatterplots. One example is the histogram of RFA_2F in Figure 7-1; it is easy to see why it was identified by the ANOVA as defining the clusters.

Consider two final notes on the ANOVA approach. First, if some of the variables did not have statistically significant mean values, they would be good candidates for pruning from the cluster model entirely. Second, the ANOVA approach can show the effect of different normalization and scaling approaches in preparing the data for clustering. When clusters were built *without* scaling on this data set, the top four variables as ranked by *F*-statistic were DOMAIN2, DOMAIN3, E_RFA_2A, and F_RFA_2A: all dummy variables. These dummy variables therefore drove the formation of the clusters when their natural 0 and 1 values were used. After they were scaled to 0.3 and 0.7, they fell in importance. Neither scaling method is right or wrong; they only provide different emphases for the K-Means and Kohonen SOMs to use in building the clusters.

Hierarchical Clustering

A second technique that you can use to interpret cluster models is *hierarchical clustering*. Why build a hierarchical cluster model to interpret K-Means clusters? Hierarchical clustering has the very appealing quality that the model shows all levels of groupings of the data, from each record belonging to its own group, to all records belonging to a single group. Usually, for predictive modeling, the disadvantage of using hierarchical clustering is the size of data; often, there are far too many records for hierarchical clusters to be efficient and practical.

However, if rather than beginning with individual records, we instead begin with the cluster centers found by K-Means as the input data (refer to Table 7-2), the number of records is small, and hierarchical clustering becomes very attractive. Consider a K-Means model with 21 clusters. After aggregating the data by cluster label and computing the mean value for each input, you have 21 records and 14 inputs. The hierarchical clustering model will then identify groups of

cluster centers that are most alike and group them together. The sequence of clusters found by hierarchical clustering shows clearly which clusters are most related to other clusters, and, by extension, which clusters are most dissimilar from other clusters.

Decision Trees

Decision trees are excellent choices for cluster interpretation because they are supervised models known for their ease of interpretation. (If you are unfamiliar with decision trees, Chapter 8 includes a full explanation.) In addition, decision trees scale well with large numbers of inputs or large numbers of levels in the target variable. If the cluster model contains 100 clusters, decision trees can find patterns for each of the clusters, provided there are enough records to find the patterns.

To use decision trees as an interpreter of clustering models, the same protocol is followed as was done for the ANOVA. The target variable is set to the cluster label column, and the same inputs used to build the clusters are selected to build the decision tree. The depth of the tree should be kept relatively shallow to maintain the interpretation of the tree, usually four to five levels deep at most, although there are reasons for building more complex trees if the complexity of the cluster model warrants it.

An additional benefit of building decision trees is that there is no need to be tied to the normalized data as one is with building the ANOVA or hierarchical clusters. Decision trees are not distance-based algorithms and therefore are unaffected by outliers and skewed distributions. Therefore, rather than using the normalized inputs, one can use the original, unnormalized inputs for every variable in the cluster model. Dummy variables can be kept with the 0 and 1 values, or even used in their categorical form. LASTGIFT and NGIFTALL do not have to be log transformed and scaled.

A third benefit to using decision trees is that they are inherently interaction detectors. The ANOVA approach is effective at finding single variable relationships to the clusters. The tree can find when two or more variables together help form the cluster. This is closer to what one expects from the cluster models: multiple variables co-occurring to form clusters. Because the original form of the variables can be used in building the decision trees, interpretation of the clusters will be even more transparent than the other approaches described so far, which relied on normalized data.

Consider a tree that is built to interpret the 3-cluster K-Means model. For the purpose of this discussion, the rules generated by the tree are considered, where the rules are the combination of decisions the tree makes to try to predict the cluster label. The rules are broken down into three groups, one for each cluster label. Table 7-4 lists seven rules found by the decision tree created to predict the three clusters.

Table 7-4: Decision Tree Rules to Predict Clusters

CLUSTER LABEL	RULE	NUMBER OF RECORDS MATCHING RULE	NUMBER OF RECORDS WITH CLUSTER LABEL IN RULE	TREE ACCURACY
Cluster 1, Rule 1	DOMAIN2 = 0 and RFA_2F > 2	6,807	6,807	100%
Cluster 1, Rule 2	DOMAIN2 = 0 and RFA_2F = 2 and E_RFA_2A = 1	1,262	1,262	100%
Cluster 2, Rule 1	DOMAIN2 = 1 and RFA_2F > 2	6,751	6,751	100%
Cluster 2, Rule 2	DOMAIN2 = 1 and RFA_2F = 2 and E_RFA_2A = 1	1,304	1,304	100%
Cluster 3, Rule 1	RFA_2F = 1 and E_RFA_2A = 0	21,415	21,415	100%
Cluster 3, Rule 2	RFA_2F = 1 and E_RFA_2A = 1	2,432	2,411	99%
Cluster 3, Rule 3	RFA_2F = 2 and E_RFA_2A = 0 and F_RFA_2A = 1	5,453	5,344	98%

The seven simple rules found by the tree are effective in finding the key drivers of the clusters. For example, Cluster 3 has two distinct groups in its definition. First, as you saw in Figure 7-1, when RFA_2F = 1 the donor belongs to Cluster 3. Second, even if RFA_2F = 2, as long as F_RFA_2A is 1, the donor still belongs to Cluster 3. This second interaction went undetected in the ANOVA and wasn't clearly identified by the mean values of Table 7-1.

Once the rules are found, visualization can make the rules clearer. Cluster 1 and Cluster 2 rules are based largely on RFA_2F and DOMAIN2, and Figure 7-2 shows a visual cross-tab indicating where the rules apply in the plot. In Figure 7-2, the x axis is RFA_2F, with the box for Cluster 3, Rule 1 above the value RFA_2F = 1 in unnormalized units. The y axis is DOMAIN2 with its two values 0 and 1.

It is not unusual that trees built to predict cluster membership have high accuracy, although the more clusters in the target variable, the more complex the tree must be to uncover all the patterns and rules. In this 3-cluster model, these seven simple rules match more than 95 percent of the records in the data, and therefore represent the vast majority of key patterns in the cluster model, as shown in Table 7-5.

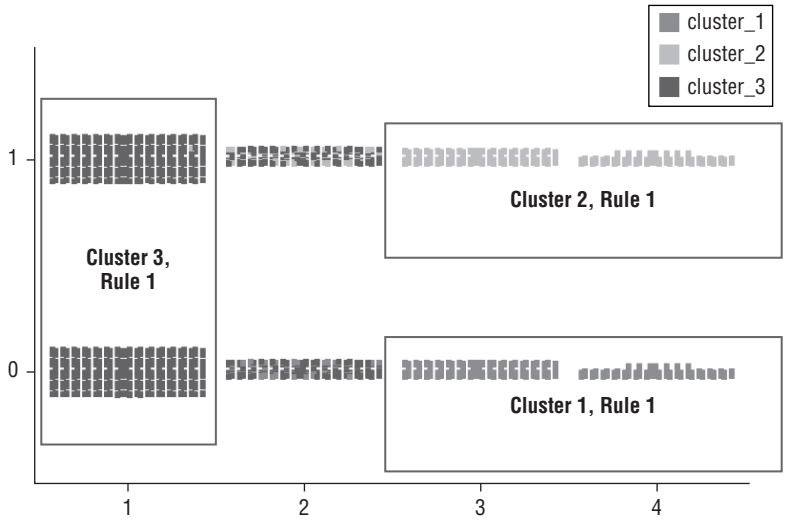


Figure 7-2: Visualization of decision tree rules

Table 7-5: Percentage of Records Matching Rule in Decision Tree

CLUSTER	# RULES	# RECORDS REPRESENTED BY KEY RULES	% OF CLUSTER
Cluster 1	2	8,069	94.5
Cluster 2	2	8,055	94.6
Cluster 3	3	29,300	95.6

Irrelevant Variables

The 3-cluster model used as an example in this chapter is a simple model with only 11 inputs. When cluster models are more complex, unlike this model, many variables may not show any relationship with the clusters when interpreted through any of these three methods.

What if some variables do not show up as significant in any interpretation model? These are good candidates for removal from the cluster model simply because they are not adding significantly to the formation of the clusters. If they truly have little influence in the clusters, their removal won't influence the cluster centers, particularly when the number of irrelevant variables is relatively small compared to the number of significant variables. A model rebuilt with fewer variables will deploy more quickly, however; this may be a consideration for some applications.

Cluster Prototypes

A prototype is an actual record in the modeling data that is a representative of the cluster rather than a statistical summary used to describe the cluster. The operation of identifying prototypes is the inverse of identifying outliers; prototypes find minimum distance records in the data as opposed to maximum distance records.

Cluster distances are computed between cluster centers and each record in the cluster. Some software will compute the distances and identify prototypes as part of the clustering algorithm, but most will not, leaving it to the modeler to identify.

Sometimes identifying records that typify the cluster can be helpful in gaining additional insight into the meaning of each cluster. The cluster center, the mean value of each input to the model, may not actually be a data point, and therefore is not truly representative of the records in the cluster. However, the records with the smallest distance to the cluster center are the most typical data points in the cluster (See Table 7-6).

Table 7-6: Cluster Prototypes

FIELD	CLUSTER 1, PROTOTYPE 1	CLUSTER 2, PROTOTYPE 1	CLUSTER 3, PROTOTYPE 1
Distance	0.375	0.301	0.313
LASTDATE	0.492	0.492	0.045
FISTDATE	0.895	0.895	0.915
RFA_2F	0.667	0.667	0
D_RFA_2A	0.3	0.3	0.3
E_RFA_2A	0.7	0.7	0.3
F_RFA_2A	0.3	0.3	0.7
DOMAIN3	0.3	0.3	0.3
DOMAIN2	0.3	0.7	0.3
DOMAIN1	0.7	0.3	0.3
NGIFTALL_log10	0.392	0.375	0.230
LASTGIFT_log10	0.347	0.347	0.434

But while these values are accurate and correct, they are not intuitive. Once the distance from the cluster center has been computed (in the normalized units) and the prototypes have been found, the original values of the key variables can be identified by joining the record by the record ID or by de-normalizing the data so that the variables associated with the minimum distance record can

be shown. This level of interpretation also helps decision-makers as they tie the clusters to what a record means, donors in the case of the KDD Cup 1998 data.

Table 7-7 shows the same table of prototypes with the variables transformed back to their original, pre-normalized units. Now you can see more clearly what values are typical of each cluster. Cluster 3 has more gifts (NGIFTALL) and smaller donation amounts (LASTGIFT) compared to Clusters 1 and 2. The socio-economic status of the donors (DOMAIN1 and DOMAIN2) is key in differentiating Cluster 1 from Cluster 2.

Table 7-7: Cluster Prototypes in Natural Units

FIELD	CLUSTER 1, PROTOTYPE 1	CLUSTER 2, PROTOTYPE 1	CLUSTER 3, PROTOTYPE 1
Distance	0.375	0.301	0.313
LASTDATE	9601	9601	9512
FISTDATE	9111	9109	9203
RFA_2F	3	3	1
D_RFA_2A	0	0	0
E_RFA_2A	1	1	0
F_RFA_2A	0	0	1
DOMAIN3	0	0	0
DOMAIN2	0	1	0
DOMAIN1	1	0	0
NGIFTALL	10	10	19
LASTGIFT	12	11	5

These tables show only one prototype for each cluster, but you can show more prototypes as well. The second prototype is the record that has the second smallest distance to the cluster mean, the third prototype the record that has the third smallest distance to the cluster mean, and so on.

Cluster Outliers

K-Means clustering and Kohonen SOMs require every record to fall into one and only one node. If there are outliers in the data, they still have to be assigned to a nearest cluster but at a large distance from the cluster center. An outlier in this context is multidimensional, based on all of the inputs in the data rather than any single variable. Sometimes multidimensional outliers are called *anomalies*.

Outliers are problematic for several reasons. First, outliers influence the cluster centers disproportionately, distorting the cluster centers. Second, operationally, these outliers are different from most of the examples in the cluster, and the

further they are from the cluster center, the less confidence we have that they can be treated the same way as other records in the cluster.

There is no precise definition of an outlier. However, if one begins with the assumption of a Normal distribution, a rule of thumb is to use three standard deviations from the mean as a metric because 99.8 percent of the data falls within plus or minus three standard deviations from the mean. Other commonly used thresholds include two standard deviations (95.6 percent of the data falls within this range) or even 90 percent of the data, calling data outliers that fall outside plus or minus 90 percent of the data.

Outliers are sometimes interesting in of themselves, particularly for risk analysis and fraud detection. The outliers are unlike anything else in the data and have been forced into becoming members of clusters they are not particularly similar to. They are, therefore, good candidates for more careful scrutiny.

Outliers in customer analytics may be ignored because they don't fit into an accepted segment, or they can be treated differently from the other customers in the segment, essentially forming a subcluster within the cluster. Identifying outliers and treating them as separate subgroups may be more efficient and pragmatically effective than adding clusters to the model specification and rebuilding.

As an example, consider Table 7-8, a table containing the cluster centers for the three clusters and the largest outlier from each cluster. The bold entries in Table 7-8 indicate values that are outliers in the cluster. Distances were computed in normalized units, but the values in the table are displayed in natural units to make the results easier to understand. Interestingly, LASTDATE has large values as outliers for each cluster even though it isn't a key variable.

Table 7-8: Cluster Outliers

FIELD	CLUSTER 1, CENTER	CLUSTER 1, OUTLIER	CLUSTER 2, CENTER	CLUSTER 2, OUTLIER	CLUSTER 3, CENTER	CLUSTER 3, OUTLIER
Distance	0	1.123	0	1.140	0	1.088
LASTDATE	9566	**9702**	9563	**9702**	9539	**9701**
FISTDATE	9069	8801	9064	**7401**	9173	**8609**
RFA_2F	3.1	1	3.1	1	1.2	1
D_RFA_2A	0.21	0	0.23	0	0.00	0
E_RFA_2A	0.50	1	0.50	1	0.08	1
F_RFA_2A	0.17	0	0.16	0	0.67	0
DOMAIN3	0.37	1	0.00	0	0.17	0
DOMAIN2	0.00	0	1.00	1	0.47	1
DOMAIN1	0.54	**0**	0.00	0	0.32	0
NGIFTALL	10.1	17	9.7	**67**	18.5	15
LASTGIFT	11.5	**0**	11.6	8	5.1	**0**

Adding a few additional clusters to the model by increasing K slightly or by adding a few more nodes in a dimension of an SOM may not capture these outliers into their own cluster, depending on how many additional clusters are added and the shape of the data. Eventually, if enough clusters are added, it is inevitable that the outliers will be characterized by their own clusters, although perhaps the outliers from a single cluster in a simpler model may require a dozen or more clusters to characterize them fully if the data points are dispersed in every direction from the center of the cluster.

Summary

Interpreting unsupervised learning models, clustering models in particular, is a challenge for predictive modelers because there are no clear and standard metrics like those used to assess supervised learning models. You should consider computing both within-cluster statistics to describe each cluster and between-cluster statistics to determine which variables are most important in separating the clusters. The former is found commonly in predictive analytics software, and even when the software does not compute within-cluster statistics, the analyst can compute them easily with simple aggregations of variables.

Software rarely computes between-cluster differences and reports the results; this chapter described several approaches you can use to identify the variables that define the clusters.

Remember to beware of outliers, variables with large magnitudes, and highly skewed variables that can bias statistics that summarize the clusters. Most of these problems should be identified and corrected during Data Preparation, though if any problems remain, they can still be uncovered from the cluster statistics.

CHAPTER

8

Predictive Modeling

This chapter explains key ideas behind algorithms used in predictive modeling. The algorithms included in the chapter are those that are found most commonly in the leading predictive analytics software packages. There are, of course, many more algorithms available than those included here, including some of my favorites.

Predictive modeling algorithms are supervised learning methods, meaning that the algorithms try to find relationships that associate inputs to one or more target variables. The target variable is key: Good predictive models need target variables that summarize the business objectives well.

Consider the data used for the 1998 KDD Cup Competition (`http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html`). The business objective is to identify lapsed donors that can be recovered with a future mailing. There are two potential target variables for this task: TARGET_B, the binary (flag) variable indicating whether or not a donor responded to the recovery mailing, or TARGET_D, the amount of the donation the donor gave if he or she responded to the recovery campaign. The algorithm you use depends in part on the target variable type: classification for categorical target variables and regression for continuous target variables.

Assume that the business objective states that a classification model should be built to predict the likelihood a lapsed donor will respond to a mailing. In this

case, you can build TARGET_B models with a classification algorithm. But which ones? What are the strengths and weaknesses of the different approaches? This chapter describes, primarily qualitatively, how the predictive modeling algorithms work and which settings you can change to improve predictive accuracy.

Predictive modeling algorithms can be placed into a few categories: Some algorithms are local estimators (decision trees, nearest neighbor), some are global estimators that do not localize (linear and logistic regression), and still others are functional estimators that apply globally but can localize their predictions (neural networks).

There are dozens of good predictive modeling algorithms, and dozens of variations of these algorithms. This chapter, however, describes only the algorithms most commonly used by practitioners and found in predictive analytics software. Moreover, only the options and features of the algorithms that have the most relevance for practitioners using software to build the models are included in the discussion.

Decision Trees

Decision trees are among the most popular predicting modeling techniques in the analytics industry. The 2009, 2010, 2011, and 2013 Rexer Analytics Data Miner surveys, the most comprehensive public surveys of the predictive analytics community, showed decision trees as the number 1 or number 2 algorithm used by practitioners, with the number of users ranging between 67 and 74 percent. Of the supervised learning algorithms, only Regression was used often by even more than one-third of the respondents.

Why such popularity? First, decision trees are touted as easy to understand. All trees can be read as a series of "if-then-else" rules that ultimately generate a predicted value, either a proportion of a categorical variable value likely to occur or an average value of the target variable. These are much more accessible to decision-makers than mathematical formulas. In addition, decision trees can be very easy to deploy in rule-based systems or in SQL.

A second reason that decision trees are popular is that they are easy to build. Decision tree learning algorithms are very efficient and scale well as the number of records or fields in the modeling data increase.

Third, most decision tree algorithms can handle both nominal and continuous inputs. Most algorithms either require all inputs to be numeric (linear regression, neural networks, k-nearest neighbor) or all inputs to be categorical (as Naïve Bayes).

Fourth, decision trees have a built-in variable selection. Coupled with their ability to scale with large numbers of inputs, they are excellent for data exploration with hundreds of variables of unknown power in predicting the target variable.

Fifth, decision trees are non-parametric, making no assumptions about distributions for inputs *or* the target variable. They can therefore be used in a wide variety of datasets without any transformations of inputs or outputs. Continuous variables do not have to conform to any particular distribution, can be multimodal, and can have outliers.

Sixth, many decision tree algorithms can handle missing data automatically rather than requiring the modeler to impute missing values prior to building the models. The method of imputation depends on the decision tree algorithm.

This chapter begins with an overview of the kinds of decision trees currently used by practitioners. It then describes how decision trees are built, including splitting criteria and pruning options. Last, it enumerates typical options available in software to customize decision trees.

The Decision Tree Landscape

Decision trees are relatively new to the predictive modeler's toolbox, coming from the Machine Learning world and entering into data mining software in the 1990s. The first tree algorithm still in use was the AID algorithm (Automatic Interaction Detection, 1963), later adding the chi-square test to improve variable selection in what became the CHAID algorithm (Chisquare Automatic Interaction Detection, 1980).

Meanwhile, in independent research, two other algorithms were developed. Ross Quinlan developed an algorithm called ID3 in the 1970s (documented in 1986), which used Information Gain as the splitting criterion. In 1993, he improved the algorithm with the development of C4.5 (1993), which used Gain Ratio (normalized Information Gain) as the splitting criterion. C4.5 was subsequently improved further in the algorithm Quinlan called C5.0, including improvements in misclassification costs, cross-validation, and boosting (ensembles). It also significantly improved the speed in building trees and built them with fewer splits while maintaining the same accuracy. The algorithm is sometimes referred to as just C5.

Another approach to building decision trees occurred concurrent with the development of ID3. The CART algorithm is described fully by Breiman, Friedman, Olshen, and Stone in the 1984 book *Classification and Regression Trees* (Chapman and Hall/CRC). CART uses the Gini Index as the primary splitting criterion and has been emulated in every major data mining and predictive analytics software package.

These are the three most common algorithms in use in predictive analytics software packages, although other tree algorithms are sometimes included,

such as the QUEST algorithm (Quick, Unbiased and Efficient Statistical Tree), which splits based on quadratic discriminant analysis, and other versions of trees that include fuzzy or probabilistic splits.

Decision tree terminology is very much like actual trees except that decision trees are upside down: Roots of decision trees are at the top, and leaves are at the bottom.

A *split* is a condition in the tree where data is divided into two or more subgroups. These are "if" conditions, such as "if petal width is <= 0.6, then go left, otherwise go right." The first split in a tree is called the *root split* or root node and is often considered the most important split in the tree. After the data is split into subgroups, trees formed on each of these subgroups are called a "branch" of the tree.

When the decision tree algorithm completes the building of a tree, terminal or leaf nodes are collectors of records that match the rule of the tree above the terminal node. Every record in the training dataset must ultimately end up in one and only one of the terminal nodes.

In Figure 8-1, you see a simple tree built using the classic Iris dataset, which has 150 records and 3 classes of 50 records each. Only two of the four candidate inputs in the Iris data are used for this tree: Petal length and petal width, sometimes abbreviated as pet_length and pet_width. The root node of the tree splits on petal width <= 0.6. When this condition is true, one goes down the left path of the tree and ends up in a terminal node with all 50 records belonging to the class setosa. On the right side of the root split, you encounter a second condition: petal width > 1.7. If this is also true, records end up in the terminal node with the class virginica. Otherwise, records end up in the terminal node with records that are largely belonging to the class versicolor.

The score for a record is the proportion of records for the most populated class in a terminal node. For example, the terminal node that contains all setosa Irises has a score of 100 related to the class setosa. Or put another way, any Iris that is found that has a petal width ≤ 0.6 inches has 100 percent likelihood of belonging to the setosa Iris species based on this data.

There is, therefore, one rule for every terminal node. For the tree in Figure 8-1, these are:

- If pet_width is ≤ 0.6, then the record belongs to class setosa with probability 100 percent.
- If pet_width > 0.6 and petal width > 1.7, then the record belongs to species virginica with probability 97.8.
- If pet_width is > 0.6 and ≤ 1.7, then the record belongs to species versicolor with probability 90.7 percent.

Trees are considered interpretable models because they can be read as rules. The interpretability, however, is severely reduced as the depth of the tree increases.

The depth is the number of conditions between the root split and the deepest terminal node, including the root split. In the tree shown in Figure 8-1, the depth is 2: first splitting on pet_width equal to 0.6 and then splitting on pet_width equal to 1.7.

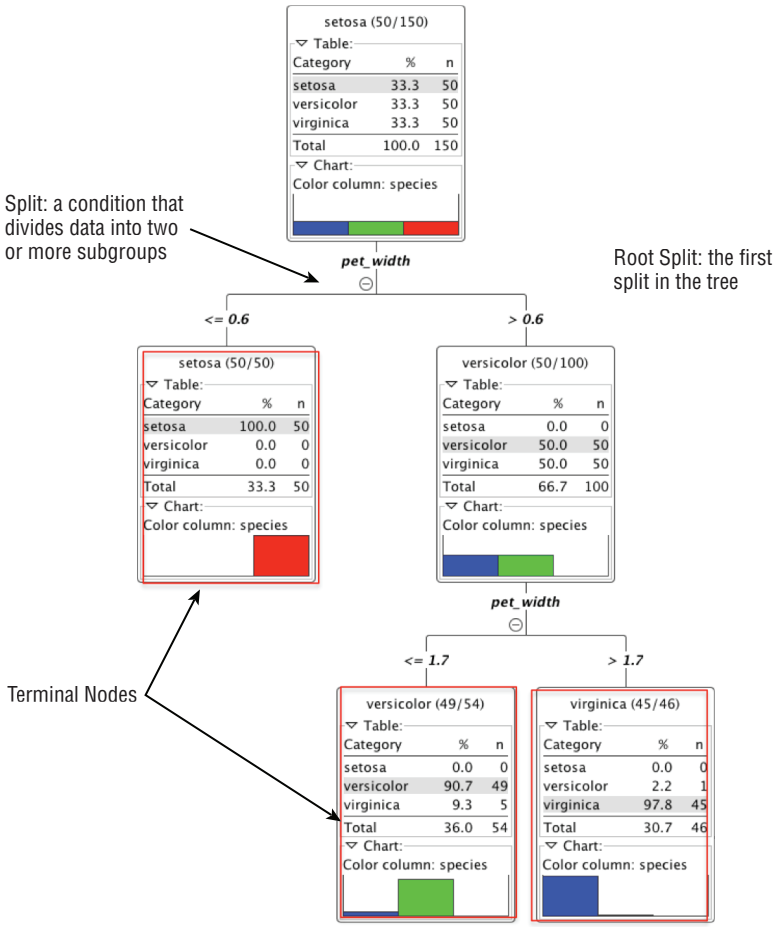


Figure 8-1: Simple decision tree built for the iris data

In a more complex tree (see Figure 8-2), every terminal has five conditions. The tree itself, even with a depth equal to only five, is too complex for the details of each node to be seen on this page. The terminal node at the far-bottom right can be expressed by this rule: If RFA_2F > 1 and CARDGIFT > 3 and LASTGIFT > 7 and AGE > 82 and RFA_2F > 2, then 46 of 100 donors do not respond. This rule that defines the records that fall into a terminal node, one of the 16 terminal nodes in the tree, has already lost considerable transparency and interpretability. What if there were 10 or 15 conditions in the rule? Decision trees are only interpretable if they are relatively simple. Figure 8-1 has a depth equal to two

and therefore is easy to see and understand. I find that trees become more difficult to interpret once the depth exceeds three. If the tree is many levels deep, you can still use the first two or three levels to provide a simpler explanation of the tree to others.

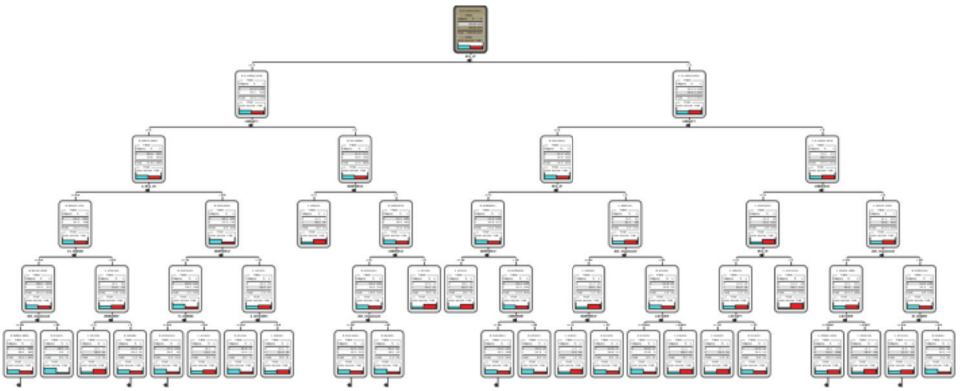


Figure 8-2: Moderately complex decision tree

Building Decision Trees

Decision trees belong to a class of *recursive partitioning algorithms* that is very simple to describe and implement. For each of the decision tree algorithms described previously, the algorithm steps are as follows:

1. For every candidate input variable, assess the *best* way to split the data into two or more subgroups. Select the best split and divide the data into the subgroups defined by the split.
2. Pick one of the subgroups, and repeat Step 1 (this is the recursive part of the algorithm). Repeat for every subgroup.
3. Continue splitting and splitting until all records after a split belong to the same target variable value or until another stop condition is applied. The stop condition may be as sophisticated as a statistical significance test, or as simple as a minimum record count.

The determination of *best* can be made in many ways, but regardless of the metric, they all provide a measure of purity of the class distribution. Every input variable is a candidate for every split, so the same variable could, in theory, be used for every split in a decision tree.

How do decision trees determine "purity"? Let's return to the Iris dataset and consider the variables petal width and petal length with three target variable classes, setosa (circle), versicolor (X-shape), and virginica (triangle), shown in Figure 8-3. It is obvious from the scatterplot that one good split will cleave out the setosa values by splitting on petal length anywhere between 2.25 and 2.75, or by splitting on petal width anywhere between the values 0.6 and 0.8.

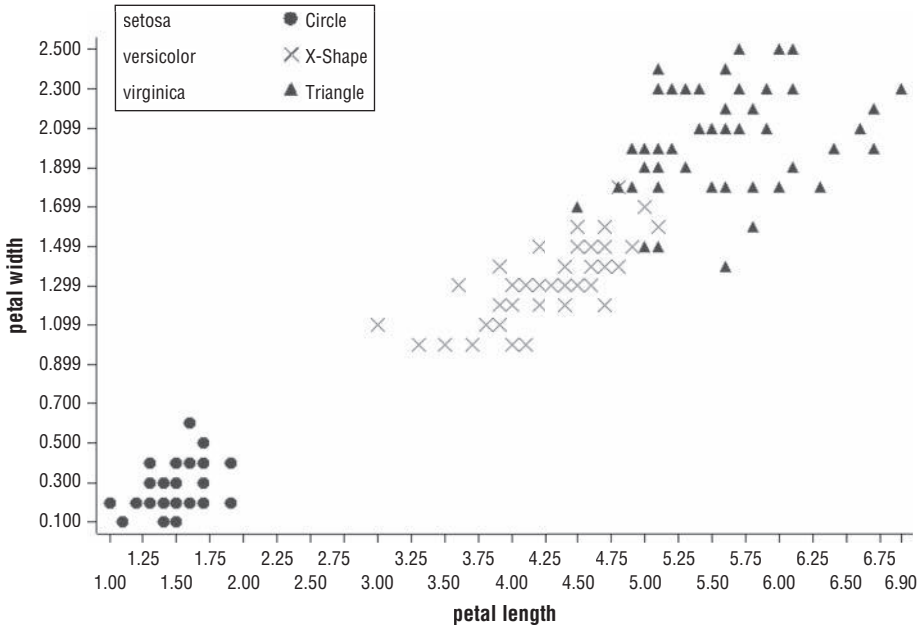


Figure 8-3: Iris data scatterplot of petal width versus petal length

Indeed, the CART algorithm first split could be on either petal length or petal width—both variables can identify perfectly the setosa class. Suppose petal length is selected. The second split selected by CART is at petal width > 1.75. The resulting regions from these two splits are shown in Figure 8-4. The shapes of the regions correspond to the shape of the target variable that is predominant.

However, what if the algorithm selected a different first split (equally as good). The decision regions created by the algorithm when the first split is petal width are shown in Figure 8-5. This tree has the same accuracy as the one visualized in Figure 8-4, but clearly defines different regions: In the first tree, large values of petal length and small values of petal width are estimated to be part of the class versicolor, whereas in the second tree with the same classification accuracy calls the same part of the decision space setosa.

Decision trees are nonlinear predictors, meaning the decision boundary between target variable classes is nonlinear. The extent of the nonlinearities depends on the number of splits in the tree because each split, on its own, is only a piecewise constant separation of the classes. As the tree becomes more complex, or in other words, as the tree depth increases, more piecewise constant separators are built into the decision boundary, providing the nonlinear separation. Figure 8-6 shows scatterplots of decision boundaries from a sequence of three decision trees. At the top, the first split creates a single constant decision boundary. At the middle, a second split, vertical, creates a simple stair step decision boundary. The third tree, at the bottom, the next stair step is formed by a second horizontal constant decision boundary, formed midway up the y axis. The decision boundary is now a nonlinear composition of piecewise constant boundaries.

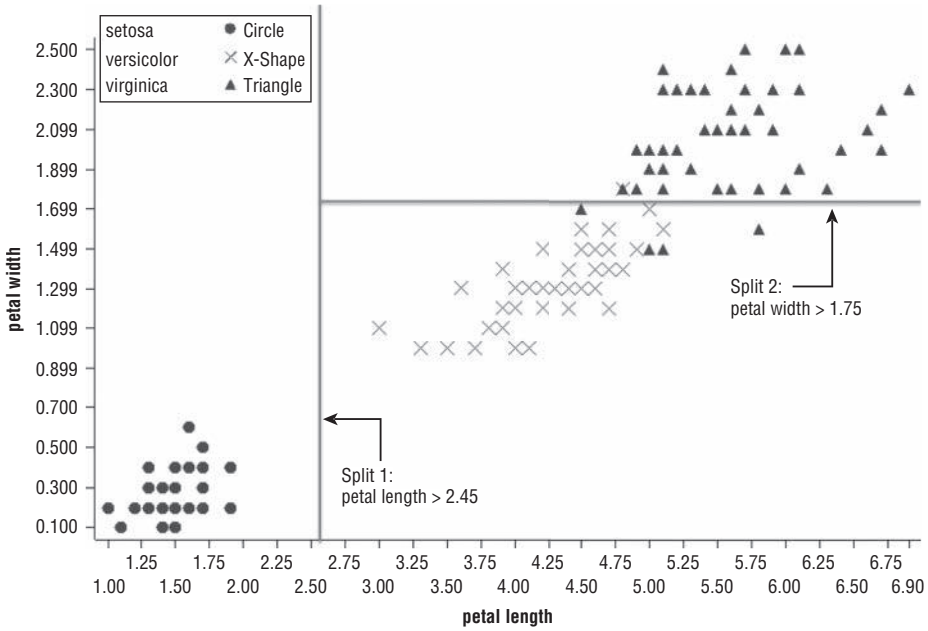


Figure 8-4: First two decision tree splits of iris data

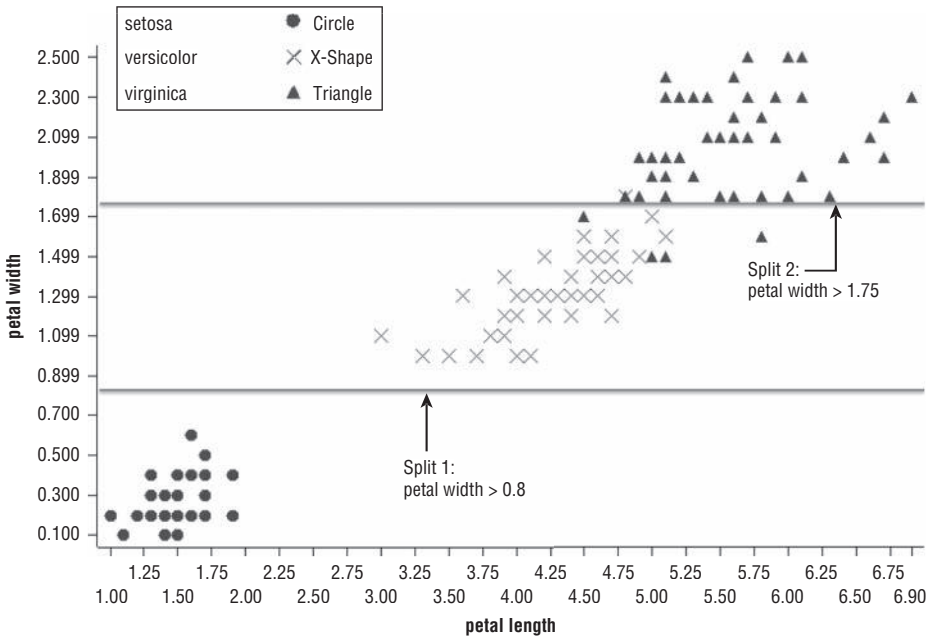


Figure 8-5: Alternative decision tree splits of iris data

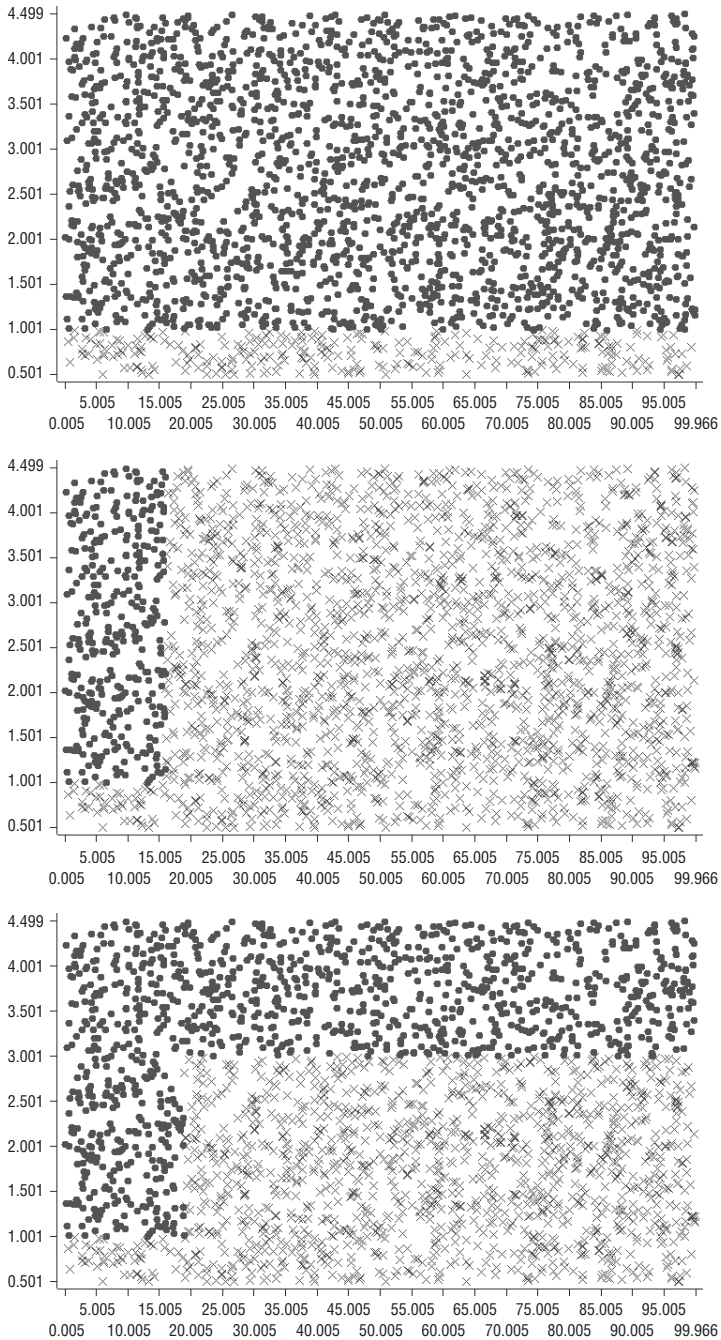


Figure 8-6: Nonlinear decision boundaries for trees

Decision Tree Splitting Metrics

The three most popular decision tree algorithms are CART, C5.0, and CHAID. Table 8-1 provides a brief comparison of these algorithms. Not all implementations

of the algorithms necessarily include all of the options in the table. For example, some implementations of the CART algorithm do not include the ability to change the priors.

The CART and C5.0 decision trees are similar in several regards. They both build trees to full-size, deliberately overfitting the tree and then they prune the tree back to the depth that trades off error with complexity in a suitable manner. Additionally, they both can use continuous and categorical input variables. CHAID, on the other hand, splits only if the split passes a statistically significant test (chi-square). The chi-square test is only applied to categorical variables, so any continuous inputs and output must be binned into categorical variables. Usually, the software bins the continuous variables automatically. You can bin the data yourself as well.

Decision Tree Knobs and Options

Options available in most software for decision trees include:

- **Maximum depth:** The number of levels deep a tree is allowed to go. This option limits the complexity of a tree even if the size of the data would allow for a more complex tree.
- **Minimum number of records in terminal nodes:** This option limits the smallest number of records allowed in a terminal node. If the winning split results in fewer than the minimum number of records in one of the terminal nodes, the split is not made and growth in this branch ceases. This option is a good idea to prevent terminal nodes with so few cases, if you don't have confidence that the split will behave in a stable way on new data. My usual preference is 30, but sometimes make this value as high as several hundred.
- **Minimum number of records in parent node:** Similar to the preceding option, but instead applies the threshold to the splitting node rather than the children of the node. Once there are fewer than the number of records specified by this option, no further split is allowed in this branch. This value is typically made larger than the minimum records in a terminal node, often twice as large. My usual preference is 50.
- **Bonferroni correction:** An option for CHAID only, this correction adjusts the chi-square statistic for multiple comparisons. As the number of levels in the input and target variable increases, and there are more possible combinations of input values to use in predicting the output values, the probability that you may find a good combination increases, and sometimes these combinations may just exist by chance. The Bonferroni correction penalizes the chi-square statistic proportional to the number of possible comparisons that could be made, reducing the likelihood a split will be found by chance. This option is usually turned on by default.

Table 8-1: Characteristics of Three Decision Tree Algorithms

TREE ALGORITHM	SPLITTING CRITERION	INPUT VARIABLES	TARGET VARIABLE	BINARY OR MULTI-WAY SPLITS	COMPLEXITY REGULARIZATION	HANDLING OF CLASS IMBALANCE
CART	Gini index, Two-ing	categorical or continuous	categorical or continuous	Binary	Pruning	Priors or Misclassification costs
C5.0	Gain Ratio, based on entropy	categorical or continuous	categorical	Binary (continuous variables) Multi-way (categorical variables)	Pruning	Misclassification costs
CHAID	chi-square test	categorical	categorical	Binary or Multi-way	Significance test	Misclassification costs

Reweighting Records: Priors

In predictive modeling algorithms, each record has equal weight and therefore contributes the same amount to the building of models and measuring accuracy. Consider, however, a classification problem with 95 percent of the records with the target class labeled as 0 and 5 percent labeled as 1. A modeling algorithm could conclude that the best way to achieve the maximum classification accuracy is to predict every record belongs to class 0, yielding 95 Percent Correct Classification! Decision trees are no exception to this issue, and one can become frustrated when building a decision tree yields no split at all because the great majority of records belong to a single target variable value.

Of course this is rarely what you actually want from a model. When you have an underrepresented target class, it is typical that the value occurring less frequently is far more important than the numbers alone indicate. For example, in fraud detection, the fraud rate may be 0.1 percent, but you can't dismiss fraudulent cases just because they are rare. In fact, it is often the rare events that are of most interest.

One way to communicate to the decision tree algorithm that the underrepresented class is important is to adjust the *prior probabilities*, usually called the *priors*. Most modeling algorithms, by considering every record to have equal weight, implicitly assign prior probabilities based on the training data proportions. In the case with the target class having 5 percent of the target class equal to 1, the prior probability of the target class equaling 1 is 5 percent.

However, what if the training data is either wrong (and you know this to be the case) or misleading? What if you really want the 0s and 1s to be considered *equally* important? The solution to these problems is to change the mathematics that compute impurity so that the records with target variable equal to 1 are weighted equally with those equal to 0. This mathematical up-weighting has the same effect as replicating 1s so that they are equal in number with 0s, but without the extra computation necessary to process these additional records. The tree algorithms that include an option of setting priors include CART and QUEST. For the practitioner, this means that one can build decision trees using the CART and QUEST algorithms when large imbalances exist in the data. In fact, this is a standard, recommended practice when using these algorithms for classification problems.

Reweighting Records: Misclassification Costs

Misclassification costs, as the name implies, provide a way to penalize classification errors asymmetrically. Without misclassification costs, each error made

by the classifier has equal weight and influence in model accuracy statistics. The number of occurrences of a particular value of the target variable has the greatest influence in the relative importance of each target variable value. For example, if 95 percent of the target variable has a value of 0 and only 5 percent has a value of 1, the learning algorithm treats the class value 0 as 19 times more important than the class value 1. This is precisely why a learning algorithm might conclude that the best way to maximize accuracy is to merely label every record as a 0, giving 95 Percent Correct Classification.

Priors provide one way to overcome this bias. Misclassification costs provide a second way to do this and are available as an option in most software implementations of decision trees. For binary classification, if you would like to overcome the 95 percent to 5 percent bias in the data, you could specify a misclassification cost of 19 when the algorithms label a record whose actual value is 1 with a model prediction of 0. With this cost in place, there is no advantage for the algorithm to label all records as 0 or 1; they have equal costs.

Typically, the interface for specifying misclassification costs shows something that appears like a confusion matrix with rows representing actual target variable values and columns the predicted values. Table 8-2 shows an example of a misclassification cost matrix with the cost of falsely dismissing target variable values equal to 1 set to 19. The diagonal doesn't contain any values because these represent correct classification decisions that have no cost associated with them.

Table 8-2: Misclassification Cost Matrix for TARGET_B

COST MATRIX	TARGET = 0	TARGET = 1
Target = 0	—	1
Target = 1	19	—

In the example with 5 percent of the population having the target variable equal to 1 and 95 percent equal to 0, let's assume you have 10,000 records so that 500 examples have 1 for the target, and 9,500 have 0. With misclassification costs set to 19 for false dismissals, if all 500 of these examples were misclassified as 0, the cost would be $19 \times 500 = 9500$, or equal to the cost of classifying all the 0s as 1s. In practice, you don't necessarily have to increase the cost all the way up to 19: Smaller values (even a third of the value, so 6 or 7) may suffice for purposes of building models that are not highly biased toward classifying 0s correctly.

The C5.0 algorithm will not create any splits at all for data with a large imbalance of the target variable, like the example already described with only 5 percent of the target equal to 1; there is not enough Information Gain with any split to justify building a tree. However, setting misclassification costs like the ones in

Table 8-2 communicate to the algorithm that the underrepresented class is more important than the sheer number of examples, and a tree will be built even without resampling the data.

In the case of binary classification, there is no mathematical difference between using priors and misclassification costs to make the relative influence of records with target variable value equal to 1 and those with target variable value equal to 0 the same. Misclassification costs can, however, have a very different effect when the target variable has many levels.

Consider the nasadata dataset, with seven target variable values (alfalfa, clover, corn, soy, rye, oats, and wheat). Priors adjust the relative occurrences of each, as if you replicated records to the point where each class value has the same number of records. Misclassification costs, on the other hand, provide *pairwise* weightings of misclassifications, and therefore there are 21 combinations of costs to set above the diagonal and 21 below, for a total of 42. These costs are not symmetric: If the example has an actual target value equal to alfalfa and the model predicts clover, one may want to set this cost differently than the converse. Table 8-3 shows a full misclassification cost matrix for the nasadata dataset. The actual target variable value is in the rows, and the predicted values in the columns. For example, misclassification costs for the target variable equal to alfalfa but a model predicting the target variable equal to clover are set to 10, whereas the converse is set to the default value of 1. Additionally, wheat misclassified as soy is penalized with a cost of 8.

Table 8-3: Misclassification Cost Matrix for Nasadata Crop Types

COST MATRIX	ALFALFA	CLOVER	CORN	SOY	RYE	OATS	WHEAT
Alfalfa	-	10	1	1	1	1	1
Clover	1	-	1	1	1	1	1
Corn	1	1	-	1	1	1	1
Soy	1	1	1	-	1	1	1
Rye	1	1	1	1	-	1	1
Oats	1	1	1	1	1	-	1
Wheat	1	1	1	8	1	1	-

Misclassification costs, therefore, are tremendously flexible. Figure 8-7 shows a simple decision tree using only the default misclassifications costs: All are equal

to 1. Band11 splits alfalfa and clover into Node 1 to the left, with approximately 50 percent of the cases belonging to each (55 alfalfa and 62 clover records).

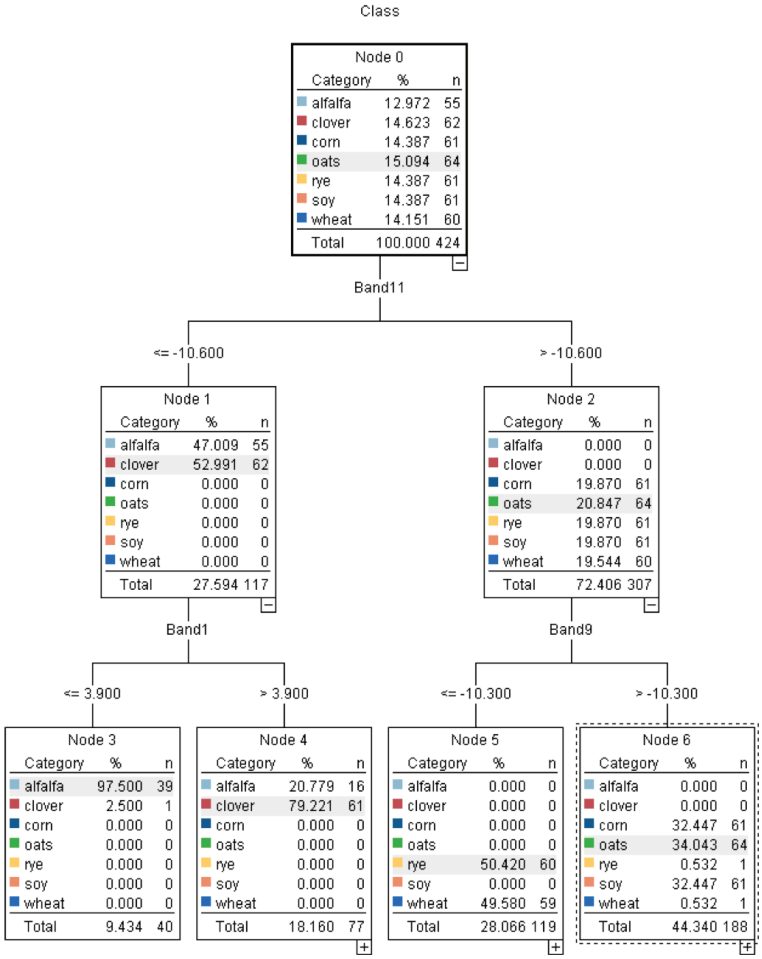


Figure 8-7: Nasadata decision tree with no misclassification costs

Misclassification costs help the classifier change the kinds of errors that are acceptable. A decision tree built with misclassification costs from Table 8-3 is shown in Figure 8-8. The first split no longer groups alfalfa and clover together—the misclassification costs made this split on Band11 too expensive—and splits on Band9 as the root split instead.

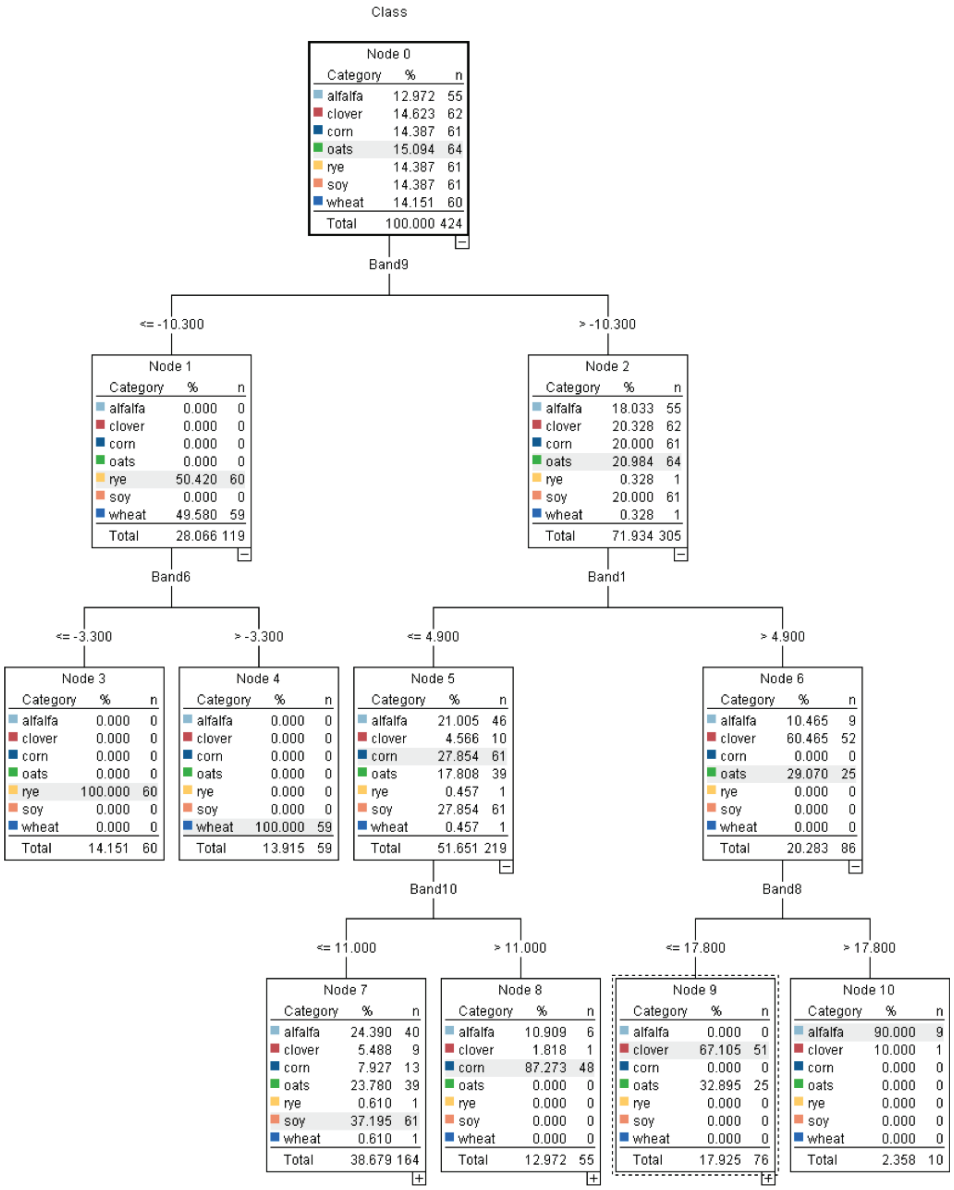


Figure 8-8: Nasadata decision tree with misclassification costs

However, with this flexibility comes difficulty. There are often no known or theoretical values to use for the costs, leaving the modeler with nothing more than a best guess to decide the values to use for misclassification costs.

Some software provides mechanisms to search over ranges of misclassification costs to help determine the optimum values. Nevertheless, misclassification costs can still be valuable for fine-tuning classifiers.

Other Practical Considerations for Decision Trees

Following are some practical considerations to make when using decision trees.

Decision trees are created through a greedy search (forward selection). Therefore, they can never "go back" to revisit a split in a tree based on information learned subsequent to that split. They can be fooled in the short run only to be suboptimal when the tree is completed. Removing the variable in the root split sometimes helps by forcing the tree to find another way to grow. Some software even provides a way to automate this kind of exploratory analysis.

Decision trees incorporate only one variable for each split. If no single variable does a good job of splitting on its own, the tree won't start off well and may never find good multivariate rules. Trees need attributes that provide some lift right away or they may be fooled. If the modeler knows multivariate features that could be useful, they should be included as candidates for the model. As an alternative, other techniques, such as association rules, could be used to find good interaction terms that decision trees may miss. You can then build new derived variables from these interactions and use them as inputs to the decision tree.

Trees are considered weak learners, or unstable models: Small changes in data can produce significant changes in how the tree looks and behaves. Sometimes the differences between the winning split and the first runner-up is very small, such that if you rebuild the tree on a different data partition, the winning/runner-up splits can be reversed. Examining competitors to the winning split and surrogate splits can be very helpful in understanding how valuable the winning splits are, and if other variables may do nearly as well. Trees are biased toward selecting categorical variables with large numbers of levels (high cardinality data). If you find a large number of levels in categorical variables, turn on cardinality penalties or consider binning these variables to have fewer levels. Trees can "run out of data" before finding good models. Because each split reduces how many records remain, subsequent splits are based on fewer and fewer records and therefore have less statistical power.

Single trees are often not as accurate as other algorithms in predictive accuracy on average; in particular, because of greedy, forward variable selection; the piecewise constant splits; and the problem of running out of data. Neural networks and support vector machines will often outperform trees. Consider building ensembles of trees to increase accuracy if model interpretation is not important.

Logistic Regression

Logistic regression is a linear classification technique for binary classification. The history of logistic regression is as follows:

Consider the KDD Cup 1998 data and the question, "Which lapsed donors to a charity can be recovered?" Dozens of giving patterns for each donor were collected and can be used to identify the patterns related to recovering and unrecovered lapsed donors. Two of the historic attributes that were collected were how many gifts a lapsed donor has given during his or her relationship with the charity (NGFITALL) and how much the lapsed donor gave for his or her last gift (LASTGIFT). Figure 8-9 shows how a logistic regression model predicts the likelihood that the donor can be recovered, 1 for recovered and 0 for not recovered. The TARGET_B outcome values predicted by the model (0 or 1) are separated by a line found by the logistic regression model: a *linear decision boundary*. If three inputs were included in the model, the decision boundary changes from a line to a plane that separates the predicted values equal to 0 from the predicted values equal to 1. With more than three inputs, the decision boundary becomes a hyper-plane.

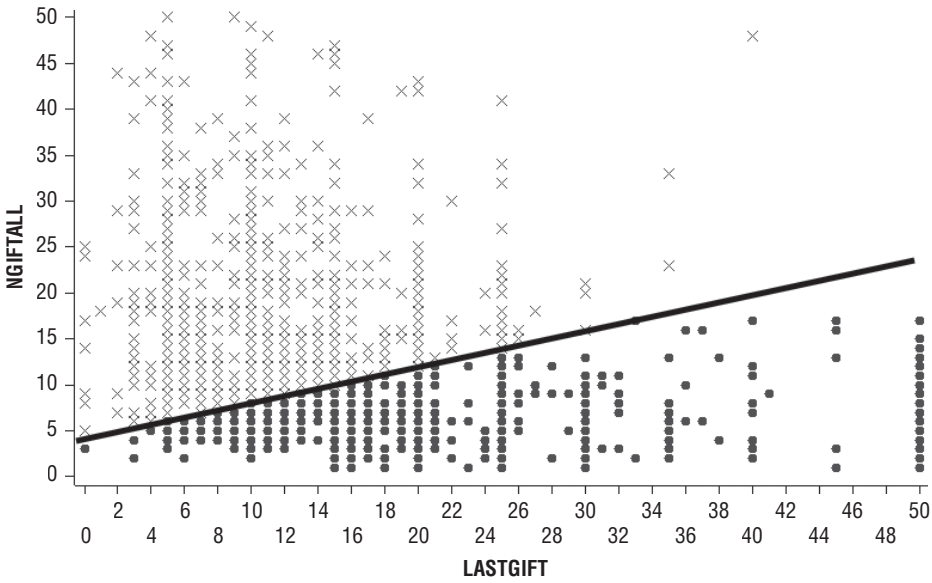


Figure 8-9: Linear decision boundary based on NGIFTALL and LASTGIFT

The core of a logistic regression model is the *odds ratio*: the ratio of the outcome probabilities.

$$odds\,ratio = \frac{P(1)}{1-P(1)} = \frac{P(1)}{P(0)}$$

Bear in mind that the odds ratio is not the same as the likelihood an event will occur. For example, if an event occurs in every 1 of 5 events (20 percent likelihood), the odds ratio is 0.2/(1 – 0.2) = 0.2/0.8 = 0.25. The odds ratio of the event *not* occurring is 0.8/0.2 = 4.0, or in other words, the odds of the event not occurring is 4 times that of it occurring.

Logistic regression does not build linear models of the odds ratio, but rather of the *log* of the odds ratio. Consider the data in Table 8-4. P(0) is the probability that the target variable has value equal to 0 given the input value specified, and P(1) is the comparable measure for a target variable equal to 1.

Table 8-4: Odds Ratio Values for a Logistic Regression Model

INPUT VALUE	P(0)	P(1)	ODDS RATIO	LOG ODDS RATIO
0	0.927	0.073	0.079	–1.103
5	0.935	0.065	0.07	–1.156
10	0.942	0.058	0.062	–1.208
15	0.948	0.052	0.055	–1.26
20	0.954	0.046	0.049	–1.313
25	0.959	0.041	0.043	–1.365
30	0.963	0.037	0.038	–1.417
35	0.967	0.033	0.034	–1.47

If you plot the input value versus the odds ratio (see Figure 8-10), you can see that the relationship between the input and the odds ratio is nonlinear (exponential). The same is true if you plot P(1) versus the input. This flattens out into a linear relationship when you compute the log of the odds ratio (see Figure 8-11).

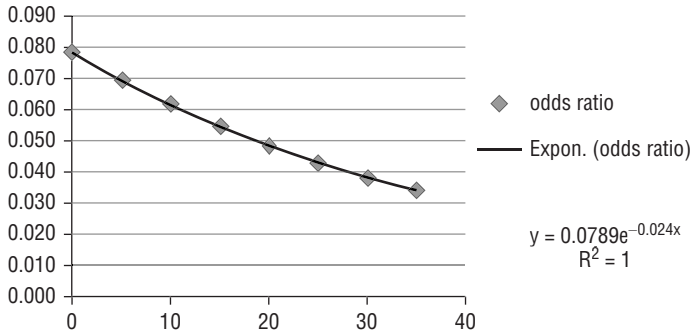


Figure 8-10: Odds ratio versus model input

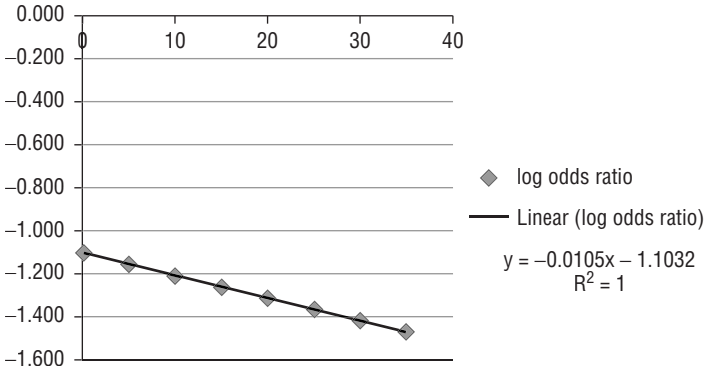


Figure 8-11: Log odds ratio versus model input

This linear relationship *is* the statistical model logistic regression in computing:

$$odds\,ratio = \frac{P(1)}{1-P(1)} = w_0 + w_1 \times x_1 + \ldots + w_n \times x_n$$

In the odds ratio equation, the values w_0, w_1, and so forth are the model coefficients or weights. The coefficient w_0 is the constant term in the model, sometimes called the bias term. The variables x_0, x_1, and so forth are the inputs to the model. For example, x_1 may be LASTGIFT and x_2 may be NGFITALL.

Without providing any derivation, the calculation of the probability that the outcome is equal to 1 is:

$$\Pr(target = 1) = \frac{1}{1 + e^{-(w_0 + w_1 \times x_1 + w_2 \times x_2 + \ldots + w_n \times x_n)}}$$

The probability formula is the function known as the *logistic curve* and takes on the shape shown in Figure 8-12. In contrast to the linear regression model whose predicted values are unbounded, the logistic curve is bounded by the range 0 to 1. The middle of the distribution, approximately in the x axis range –2 to 2 in the figure, is approximately linear. However, the edges have severe nonlinearities to smoothly scale the probabilities to the bounded range. When you change the input value from 0 to +2, the probability goes from 0.5 to 0.88, whereas when you change the input from +4 to +6, the probability changes only from 0.98 to 0.998.

How is the probability calculation, with values scaled nonlinearly so that it has a range between 0 and 1, and the linear decision boundary? Assume that the prior probability of the target variable, TARGET_B, is 50 percent. All of the data points that lie on the line that separates the predicted TARGET_B values equal to 0 from those equal to 1 have predicted probability values, P(1), equal to 0.5.

The further from the line a data point resides, the larger the value of P(1) or P(0). For a data point in the upper left of Figure 8-9, LASTGIFT = 5 and NGIFTALL = 70, the P(1) is 0.82: a high likelihood the donor can be recovered, and a data point far from the decision boundary. The further from the decision boundary, the smaller the relative increase in P(1) until it approaches 1.0.

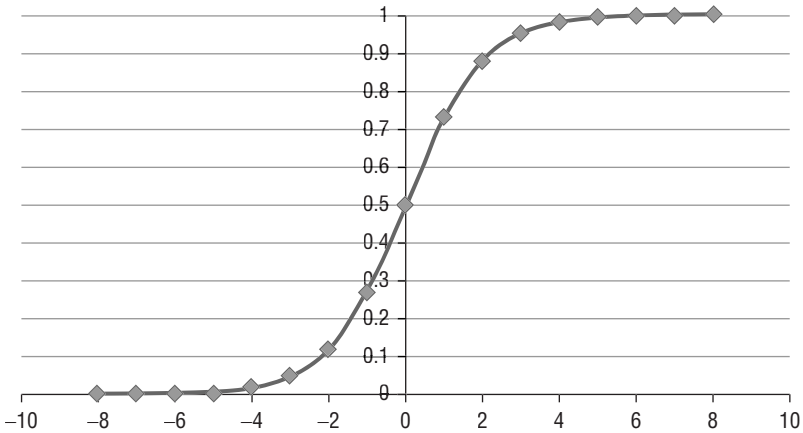


Figure 8-12: Logistic curve

Interpreting Logistic Regression Models

To interpret logistic regression models, you'll examine several numbers most software packages provide in their summary reports:

- **Coefficient:** Sometimes shown as the Greek symbol beta (ß). This is the parameter found by the logistic regression algorithm; one coefficient per input variable plus one constant (bias term).
- **Standard error of the coefficient (*SE*):** A measure of certainty of the coefficient found by logistic regression. Smaller values of standard error imply a smaller level of uncertainty of the coefficient value.
- **Confidence Interval (CI):** The range of values the coefficient is expected to fall between. The CI is computed as:

$$CI = \beta \pm SE$$

This is sometimes helpful when evaluating models. You can compute the Pr(1) not just for the coefficient with value ß, but also for the coefficient plus the *SE* and minus the *SE*, thereby computing the sensitivity of the logistic regression probabilities.

- **z statistic or Wald test:** z is calculated by the equation:

$$z = \frac{\beta}{SE}$$

  The larger the value of z, the more likely the term associated with the coefficient is significant in the model.

- **Pr(>|z|):** The p value associated with the z statistic. Often, analysts consider values less than 0.05 to indicate a significant predictor, although there is no theoretical reason for making this inference; it is historical precedence. The p value does not indicate how much accuracy the variable provides, *per se*, but rather how likely it is that the coefficient is different from 0. The more records one has in the training data, the smaller SE and the p value will be. Therefore, the more records you have, the more stable the coefficients become.

Some implementations of logistic regression include variable selection options, usually forward selection or backward elimination of variables. The p value is often the metric used to decide to include or exclude a variable, by including new variables only if the p value is less than some value (typically 0.05), or excluding a variable if its p value is greater than some value (often 0.1). These thresholds are usually configurable.

If variable selection options are not available, the practitioner can use the p values to decide which variables should be excluded. For example, if you use logistic regression to build a model to predict lapsed donors with target variable TARGET_B, and include inputs NGIFTALL, LASTGIFT, FISTDATE, RFA_2F, RFA_2A, and AGE, the following model was created, with coefficients and summary statistics shown in Table 8-5.

Table 8-5: Logistic Regression Model Report

VARIABLE	COEFF.	STD. ERR.	Z-SCORE	P>|Z|
NGIFTALL	0.0031	0.0024	1.2779	0.2013
LASTGIFT	0.0008	0.0014	0.5702	0.5685
FISTDATE	−0.0003	0.0000664	−3.9835	0.0000679
RFA_2F=2	0.2686	0.0405	6.636	3.22E−11
RFA_2F=3	0.3981	0.047	8.4638	0
RFA_2F=4	0.5814	0.0538	10.7987	0
RFA_2A=E	−0.1924	0.0535	−3.5937	0.0003
RFA_2A=F	−0.3558	0.0613	−5.8085	6.31E−09

VARIABLE	COEFF.	STD. ERR.	Z-SCORE	P>|Z|
RFA_2A=G	–0.5969	0.075	–7.9642	1.89E–15
AGE	–0.0014	0.0011	–1.3129	0.1892
Constant	–0.381	0.6351	–0.5999	0.5486

The worst predictor according to the *p* value is LASTGIFT, and therefore this variable is a good candidate for removal. This doesn't mean that LASTGIFT isn't a good predictor by itself, only that in combination with the other inputs, it is either relatively worse, or the information in LASTGIFT also exists in other variables rendering LASTGIFT unnecessary to include in the model. After removing LASTGIFT, the model is rebuilt, resulting in the model coefficient values and summary statistics shown in Table 8-6.

Table 8-6: Logistic Regression Model after Removing LASTGIFT

VARIABLE	COEFF.	STD. ERR.	Z-SCORE	P>|Z|
NGIFTALL	0.0031	0.0024	1.2772	0.2015
FISTDATE	–0.0003	0.0000664	–3.9834	0.0000679
RFA_2F=2	0.2671	0.0404	6.6136	3.75E–11
RFA_2F=3	0.396	0.0469	8.4465	0
RFA_2F=4	0.5787	0.0536	10.7924	0
RFA_2A=E	–0.1898	0.0534	–3.558	0.0004
RFA_2A=F	–0.349	0.0601	–5.8092	6.28E–09
RFA_2A=G	–0.5783	0.0673	–8.5932	0
AGE	–0.0014	0.0011	–1.3128	0.1893
Constant	–0.3743	0.635	–0.5894	0.5556

Now NGIFTALL and AGE are the only remaining variables with *p* values greater than 0.05. You can continue the process until all the variables have *p* values greater than 0.05 if the goal is to identify variables that are all statistically significant at the 0.05 level.

However, it is possible that variables with *p* values above the 0.05 significance level still provide predictive accuracy. If accuracy is the objective, you should use accuracy to decide which variables to include or exclude.

Other Practical Considerations for Logistic Regression

In addition to the principles of building logistic regression models described so far, successful modelers often include other data preparation steps.

The following four considerations are among the most common I apply during modeling projects.

Interactions

Logistic regression is a "main effect" model: The form of the model described so far has been a linearly weighted sum of inputs with no interaction terms. Sometimes, however, it is the interaction of two variables that is necessary for accurate classification.

Consider a simple example in Figure 8-13: The cross data set created from two inputs, x and y, and a binary target variable z represented by X-shapes (target variable = +1) and circles (target variable = –1). This example is just an extension of the famous "XOR" problem cited by Minsky and Pappert in their book *Perceptrons* (MIT, 1969). Clearly there is no linear separator that classifies this dataset.

A logistic regression model using only a linear combination of inputs x and y—a main effect model—only classifies approximately 50 percent of the cases correctly. The predicted values from the logistic regression model are shown in Figure 8-14—clearly not an effective classifier.

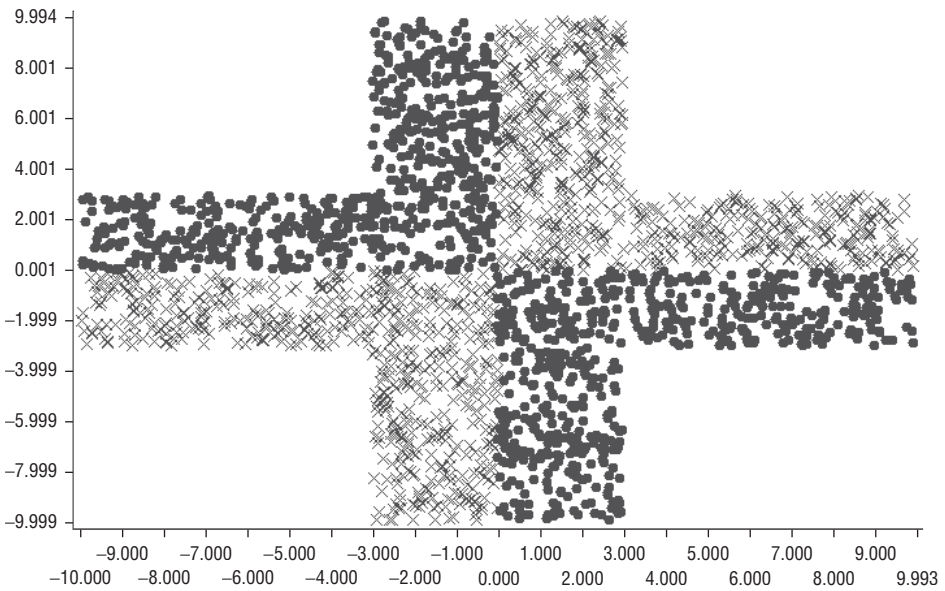


Figure 8-13: Cross data

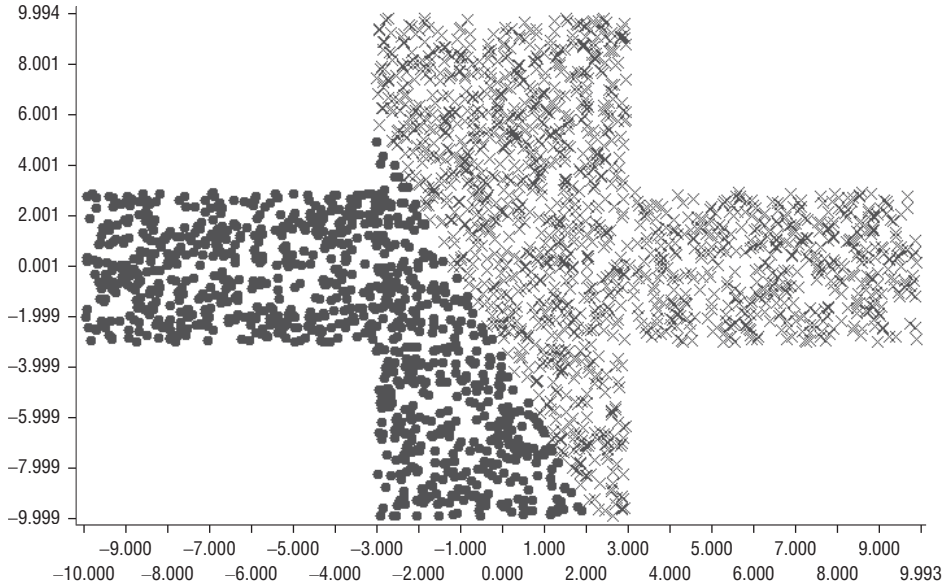


Figure 8-14: Linear separation of cross data

Consider adding an interaction term to the model: $x \times y$. A histogram of this new feature, shown in Figure 8-15, shows the value of this interaction: The model that was able to do no better than a "random" guess using x and y by themselves has become, with an interaction term, an excellent model.

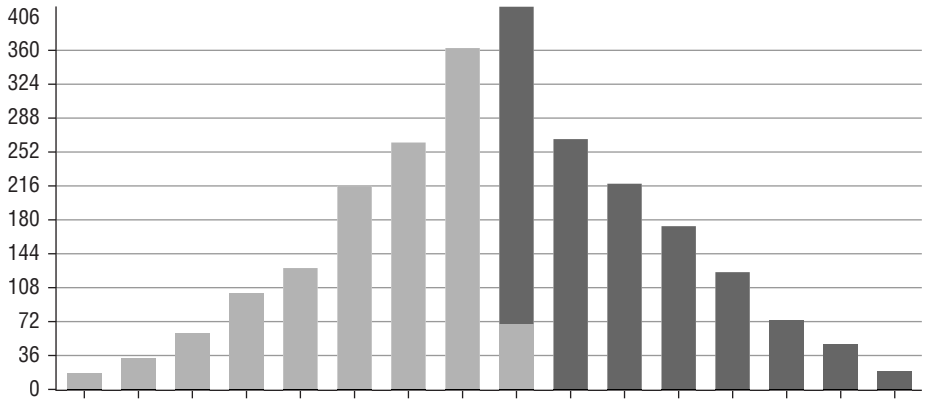


Figure 8-15: Histogram of interaction variable

Missing Values

Logistic regression cannot have any missing values. How missing values are handled is implementation dependent, although often the course of action in software is listwise deletion. All of the typical missing value imputation methods apply.

Dummy Variables

Logistic regression is a numeric algorithm: All inputs must be numeric. Categorical variables must therefore be transformed into a numeric format, the most common of which is 1/0 dummy variables: one column for each value in the variable minus one. If a categorical variable has eight levels, create seven dummy variables only; the contribution for the eighth will be taken up by the bias (constant) term. If all values are represented in the dummy variable, the multicollinearity in the data will cause the terms combing from the same target variable to have *z* values that fail significance tests.

Consider a simple logistic regression model with one categorical input having four levels: A, B, C, and D. Dummy variables are created for each level so that there are now four candidate inputs to the logistic regression model. If all four are included in the model, you may see bizarre standard error values like those in Table 8-7. The *p* values equal to 0.9999 make these four variables seem unrelated to the target. Note that even the constant term is problematic.

Table 8-7: Logistic Regression Model with *n* Dummy Variables

VARIABLE	BETA	SE	WALD TEST, Z	P>|Z|
X1=D	2.38	28,630.90	8.31E–05	0.9999
X1=E	2.80	28,630.90	9.78E–05	0.9999
X1=F	3.24	28,630.90	0.0001	0.9999
X1=G	3.49	28,630.90	0.0001	0.9999
Constant	–0.1471	28,630.90	–5.14E–06	1

The clue, however, is that the *SE* values are so large. This indicates a numerical degeneracy in their calculation. The general principle is this: For categorical variables with *n* levels, only include *n* –1 dummy variables as inputs to a logistic regression model. If you remove one of the dummy variables, such as

X1=G, the problem goes away and the model looks fine (Table 8-8). It doesn't matter which dummy variable is removed; you can choose to remove the dummy that is the least interesting to report.

Table 8-8: Logistic Regression Model with *n*–1 Dummy Variable

VARIABLE	BETA	SE	WALD TEST, Z	P>|Z|
X1=D	–1.1062	0.08	–14.0	0
X1=E	–0.6865	0.07	–10.01	0
X1=F	–0.2438	0.06	–3.7863	0.0002
Constant	3.3387	0.0558	59.7835	0

Multi-Class Classification

If the target variable has more than two levels, logistic regression cannot be used in the form described here. However, there are two primary ways practitioners extend the logistic regression classifier.

The most common way is to explode the multi-class variable with N levels into N –1 dummy variables, and build a single logistic regression model for each of these dummy variables. Each variable, as a binary variable, represents a variable for building a "one vs. all" classifier, meaning that a model built from each of these dummy variables will predict the likelihood a record belongs to a particular level of the target variable in contrast to belonging to *any* of the other levels. While you could build N classifiers, modelers usually build N–1 classifiers and the prediction for the Nth level is merely one minus the sum of all the other probabilities:

$$P(Target = N) = \sum_{i=1}^{N-1} P(Target = i)$$

When deploying models using this approach, N–1 model scores are generated and a seventh score is computed from the other six. For the nasadata dataset, six models are created which generate six probabilities. The seventh probability is one minus the sum of the other six. Typically, the maximum probability from the seven classes is considered the winner, although other post-processing steps can be incorporated as well to calculate the winning class.

The benefit of this approach is that the number of classifiers scales linearly with the number of levels in the target variable, and the interpretation of the resulting models is straightforward. However, this approach groups all the remaining target levels into a single value, 0, regardless of whether they are related to one another or not, and therefore the model may suffer in accuracy as a result.

An alternative that is used sometimes is to build a model for every pair of target variable levels: the *pairwise classifier* approach. This approach requires sampling records to find just those that apply to each pair of levels a classifier is built for. For that levels, this requires building $N \times (N–1)/2$ classifiers. For example, the nasadata contains 7 classes, and therefore building a classifier for each pair of classes requires building 21 classifiers ($(7 \times 6)/2$).

It is certainly true that the pairwise approach requires building many more models than the one-vs.-all approach; it isn't necessarily the case that computationally it is more expensive. The nasadata has 424 records, but only approximately 60 per class. Every classifier therefore is built on only 120 records rather than the full dataset, so the computation burden depends also on the efficiency of the algorithm.

Deploying the pairwise set of classifiers requires that a new score (a column) be created for every classifier. Each level will have $N–1$ classifiers with its level involved. For the nasadata example, you have 21 classifiers and each target variable level is involved in 6 of them. You can decide on the winner through a number of approaches, including averaging the probabilities for each level (6 each in the nasadata example) or counting votes where each level has the larger probability.

Beware of implementations that automatically handle multilevel target variables by creating dummies. Some implementations use the same model structure for each one-vs-all classifier even if some variables are not good predictors for a particular one-vs-all model.

Neural Networks

Artificial Neural Networks (ANN) began as linear models in 1943 with the introduction of a model that emulated the behavior of a neuron with McCulloch and Pitts. These were linear models with a threshold, although they were not introduced primarily for their biological analogy but rather for their computational flexibility. In the 1950s and 60s, there was a flurry of activity surrounding these ideas, primarily using single nodes, although some began experimenting

with multiple layers of nodes, including Frank Rosenblatt, Bernie Widrow, Roger Barron, and others. However, ANNs were primarily known by the single neuron in the early days, either the perceptron as designed by Rosenblatt or an ADALINE (Adaptive Linear Neuron) by Widrow.

The momentum gained in neural network research during the 1950s and 60s hit a wall in 1969 with the publication of the book *Perceptrons* (The MIT Press) by Marvin Minsky and Seymour Pappert, which proved that the perceptron could not predict even elementary logic functions such as the exclusive or (XOR) function. While this is true, a second hidden layer in a neural network can easily solve the problem, but the flexibility of 2-layer neural networks wasn't able to break through the impression gained from the Minsky and Pappert book about the deficiencies of perceptrons. Neural networks remained in the background until the 1980s.

The resurgence of neural networks in the 1980s was due to the research in Parallel Distributed Processing (PDP) systems by David Rumelhart, James Mclelland, and others. It was in 1986 that the backpropagation algorithm was published. Backpropagation enabled the learning of the neural network weights of a multilayer network of elements with continuous (initially sigmoidal) activation functions. It was designed to be simple from the start, or in the words of Rumelhart, "We'll just pretend like it [the neural network] is linear, and figure out how to train it as if it were linear, and then we'll put in these sigmoids." Another significant development was the increase in speed of computers so they could handle the computational load of neural networks. This allowed researchers to begin experimenting with neural networks on a wide range of applications. The first international neural network conference of the Institute of Electrical and Electronics Engineers (IEEE) was held in 1987, giving further credibility to neural networks as an accepted approach to data analysis.

In 1989, George Cybenko proved that a sigmoidal neural network with one hidden layer can approximate any function under some mild constraints. This flexibility of the neural network to adapt to any reasonable function became a powerful rallying cry for practitioners, and by the 1990s, neural networks were being widely implemented in software both in standalone, neural network–only software packages as well as options in data mining software that contained a suite of algorithms.

ANNs are a broad set of algorithms with great variety in how they are trained, and include networks that are only feedforward as well as those with feedbacks. Most often, when the phrase "neural network" is used in predictive analytics, the intent is a specific neural network called the mutli-layer perceptron (MLP). The neural networks described in this section therefore are limited to MLPs.

Building Blocks: The Neuron

The perceptron is an ANN comprised of a single neuron. In an MLP, neurons are organized in layers in a fully connected, feedforward network. Each neuron is simply a linear equation just like linear regression as shown in the following equation. Figure 8-16 shows a representation of a single neuron. Just as with the logistic regression model, the values *w0*, *w1*, and so forth are the weights, and the values *x0*, *x1*, and so forth are the inputs to the neural network.

$$y_i = w_0 + w_1 \times x_1 + w_2 \times x_2 + \ldots + w_n \times x_n$$

This equation is often called the *transfer function* in an ANN. In the early days of neural networks, in the 1960s, this linearly weighted sum would be thresholded at some value so that the output of the neuron would be either 1 or 0. The innovation in the 1980s was the introduction of a soft-thresholding approach by means of an *activation function*.

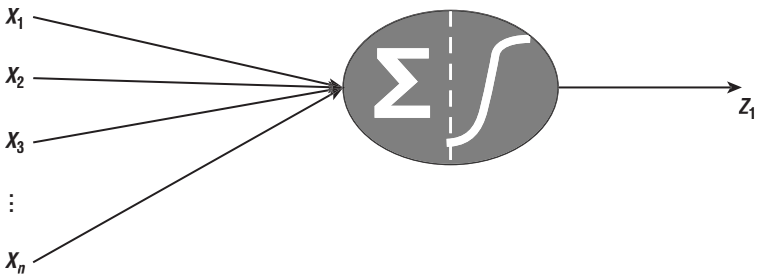


Figure 8-16: Single neuron

The neuron contains one other element after the linear function. The output of the linear function is then transformed by the activation function, often called a *squashing function*, that transforms the linear output to a number between 0 and 1. The most common squashing function is the sigmoid, which is the same function as the logistic regression logistic curve:

$$z_1 = logstic(y) = \frac{1}{1 - e^{-y}}$$

Other functions used for the squashing function include the hyperbolic tangent (tanh) and arctangent (arctan). Determining which activation function to use is not usually very important: Any of them suffice for the purpose of transforming the transfer function. One difference between these activation functions is that the logistic curve ranges from 0 to 1, whereas tanh and arctan range from –1 to +1.

$$z_1 = \tanh(y) = \frac{e^{y} - e^{-y}}{e^{y} + e^{-y}}$$

The key is that the activation functions are continuous and nonlinear. Because they are continuous, you can compute derivatives, an essential property of the neurons so that the learning algorithms used to train neural networks can be applied. Neurons, except for output layer neurons for reasons to be described later, must be nonlinear for the neural network to be able to estimate nonlinear functions and nonlinear relationships in the data. It is precisely the nonlinear aspect of the neuron that gives the neural network predictive power and flexibility.

A single neuron builds a linear model: a linear decision boundary for classification or a linear estimator for continuous-valued prediction. Such models are not particularly interesting because there are many linear modeling algorithms that are as accurate as the single neuron but are algorithmically far more efficient. However, when layers are added to the ANN, they become far more flexible and powerful as predictors.

A layer is a set of neurons with common inputs, as shown in Figure 8-17. The neurons, in general, will have different coefficients, all learned by the training algorithms used by the ANN.

Some layers in the ANN are called *hidden layers* because their outputs are hidden to the user. In the network shown in Figure 8-18, there are two hidden layers in the ANN. This network also has an output layer: one for every target variable in the data. If you build classification models for the nasadata, each of the seven target variable classes will have its own output layer neuron. Note that this is in contrast to logistic regression models where completely separate models for each target variable value are built.

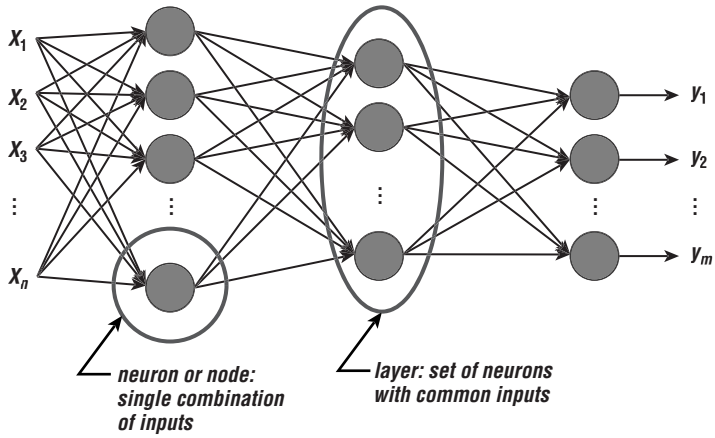


Figure 8-17: Neural network terminology, part 1

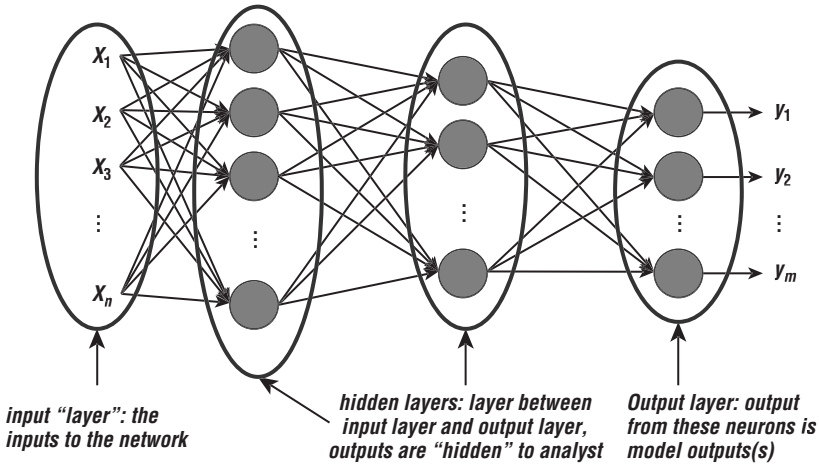


Figure 8-18: Neural network terminology, part 2

For classification modeling, the output layer neurons usually have the same sigmoid activation function already described, one that transforms the neuron to a value between 0 and 1. For continuous-valued prediction, however, software implementations of ANNs most often use what is called a "linear" activation function, which means that no transformation is applied to the linearly weighted sum transfer function. For target variables that are unbounded, you can also use an exponential activation function in the output layer neurons.

The cost function for a neural network predicting a continuous-valued output minimizes squared error just like linear regression. In fact, without the squashing functions for each neuron in a neural network, the network would just be a large, inefficient, linear regression model. The nonlinear squashing functions enable the network to find very complex relationships between the inputs and output without any intervention by the user.

For classification, many predictive analytics software packages use cross-entropy as the cost function rather than minimizing squared error because it seems to be a more appropriate measure of error for a dichotomous output classification problem. It has been shown, however, that even a squared-error cost can find solutions every bit as good as cross-entropy for classification problems.

A neural network for classification with no hidden layer and a single output neuron is essentially a logistic regression model when the neuron has a logistic activation function, especially if the cost function is cross-entropy. With the squared error cost function, there is no guarantee the model will be the same.

Neural Network Training

ANNs are iterative learners, in contrast to algorithms like linear regression, which learn the coefficients in a single processing step. The learning process is similar to how I learned to catch fly balls as a boy. First, imagine my father

hitting a fly ball to me in the outfield. In the beginning, I had absolutely no idea where the ball was going to land, so I just watched it until it landed, far to my left. Since my goal was to catch the ball, the distance between where the ball landed and where I stood was the error. Then my father hit a second fly ball, and, because of the prior example I had seen, I moved (hesitantly) toward where the ball landed last time, but this time the ball landed to my right. Wrong again.

But then something began to happen. The more fly balls my father hit, the more I was able to associate the speed the ball was hit, the steepness of the hit, and the left/right angle—the initial conditions of the hit—and predict where the ball was going to land. Major league outfielders are so good at this that for many fly balls, they arrive at the spot well before the ball arrives and just wait.

This is exactly what neural networks do. First, all the weights in the ANN are initialized to small random values to "kick start" the training. Then, a single record is passed through the network, with all the multiplies, adds, squashing computed all the way through neurons in hidden layers and the output layer, yielding a prediction from each output neuron in the neural network. The error is computed between the actual target value and the prediction for that record. Weights are adjusted in the ANN proportional to the error. The process for computing the weights for one output neuron is described here, but the same principles apply for more than one output neuron.

The process is repeated next for the second record, including the forward calculations ending in predictions from the output layer neuron(s) and error calculations, and then weights are updated by backpropagation of the errors. This continues until the entire set of records has passed through the ANN. Figure 8-19 is a representation of the process where each jump is the result of a single record passing through the network and weights being updated. The representation here shows a network converging to the minimum error solution if the error surface is a parabola, as is the case when minimizing squared errors for a linear model such as linear regression. The actual error surface is much more complex than this.

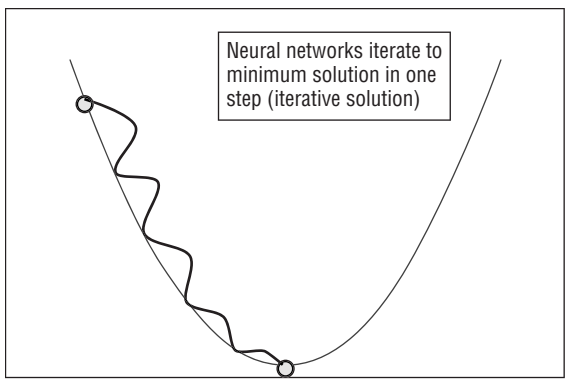


Figure 8-19: Iterative convergence to the bottom of a quadratic error curve

An *epoch* or *pass* occurs when every record in the training data has passed through the neural network and weights have been updated. Training a neural network can take dozens, hundreds, or even thousands of training epochs; you can understand why ANNs have a reputation for taking a long time to train. The training time on commonly available computer hardware is acceptable for most problems.

The forward and backward process continues until a stop condition applies to end training. The difficulty in converging to a solution is non-trivial because of the non-linearity of neural networks. Rather than the solution following the smooth quadratic error surface shown in Figure 8-19, it is more like the conceptual error curve shown in Figure 8-20 with many dips and hills. These dips are local minima. The challenge for training a neural network is to traverse the error surface at just the right pace.

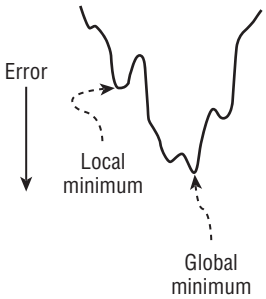


Figure 8-20: Local and global minima

If the algorithm takes too small of a step, it can easily end up in a local minimum, as shown on the left in Figure 8-21. A larger learning rate is sufficient to jump over the local minimum on the right, enabling the algorithm to find the global minimum.

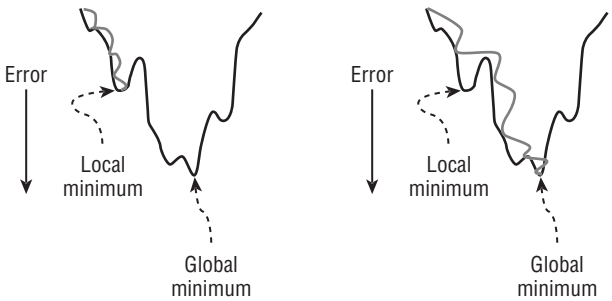


Figure 8-21: Iterating to local and global minima

You never know if the global minimum has been found because you never know if all of the points of the error surface have been found. In the left sequence of errors of Figure 8-22, you have found a minimum. You only know it is a local minimum if you build a second neural network and find a better solution, such as the sequence of errors on the right of Figure 8-22. Is this minimum a global minimum? You can never know for sure.

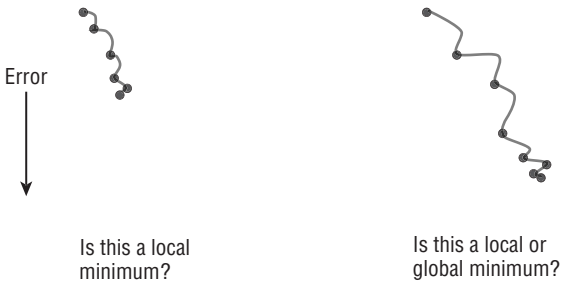


Figure 8-22: Local or global minimum?

Experimentation therefore is key with neural networks. Modelers typically build several networks with varying parameters, which include varying how many neurons to include in hidden layers, learning rates, even changing the random number seed that generates the initial weights for the network.

The Flexibility of Neural Networks

As universal approximators, neural networks are very flexible predictors. One example is the ability of the ANN to predict the sombrero function without having to create derived variables to unravel the nonlinearities in the data. The sombrero function is defined by the equation:

$$z = \frac{\sin(sqrt(x^2 + y^2))}{sqrt(x^2 + y^2)}$$

and has two classes, represented by 1 (X-shapes) and 0 (circles). The (X-shapes) class is characterized by those data points with $z > 0$ and the (circles) class with $z <= 0$. Figure 8-23 shows the function.

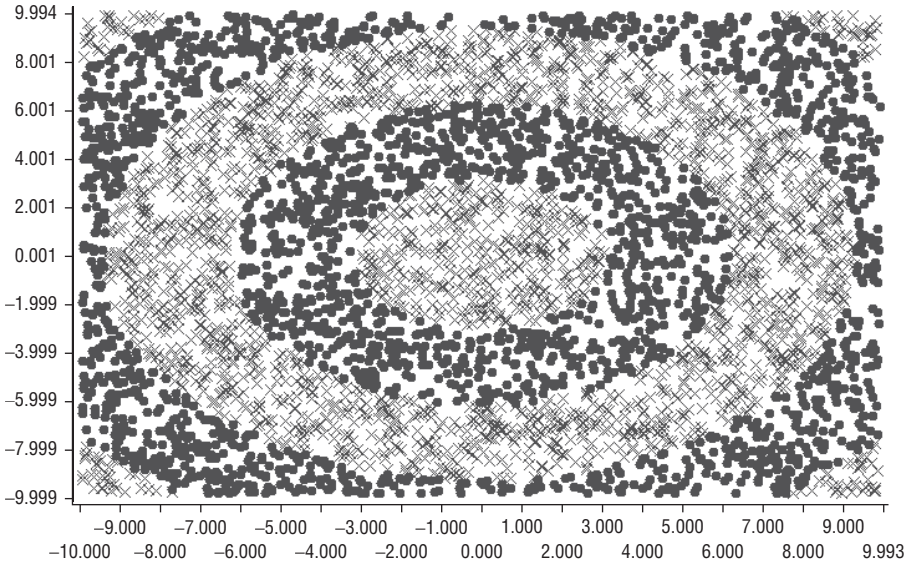


Figure 8-23: Sombrero function

This function is clearly not linearly separable; any line drawn will necessarily contain both classes on either side of the line (see Figure 8-24). This function cannot, therefore, be estimated by logistic regression models without transforming the inputs to "linearize" the relationship between inputs and the output. Likewise, decision trees will have great difficulty with this function because of the smooth curves throughout the decision boundaries between the values.

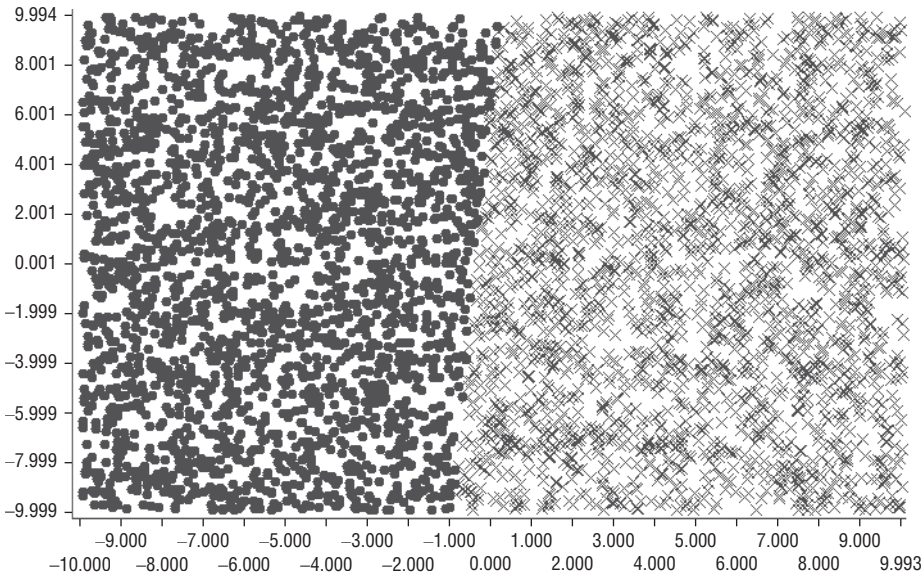


Figure 8-24: Linear classifier for sombrero function

An MLP with two hidden layers containing twelve hidden units per hidden layer was trained on this data. As training of the neural network progresses through 10, 100, 200, 500, 1000 and 5000 epochs, you see in Figure 8-25 the true shape of the sombrero function forming. With only 10 or 100 epochs, the sombrero pattern has not yet been learned. The gist of the sombrero function is seen after 500 epochs, and after 5000 epochs, the sombrero shape is clear.

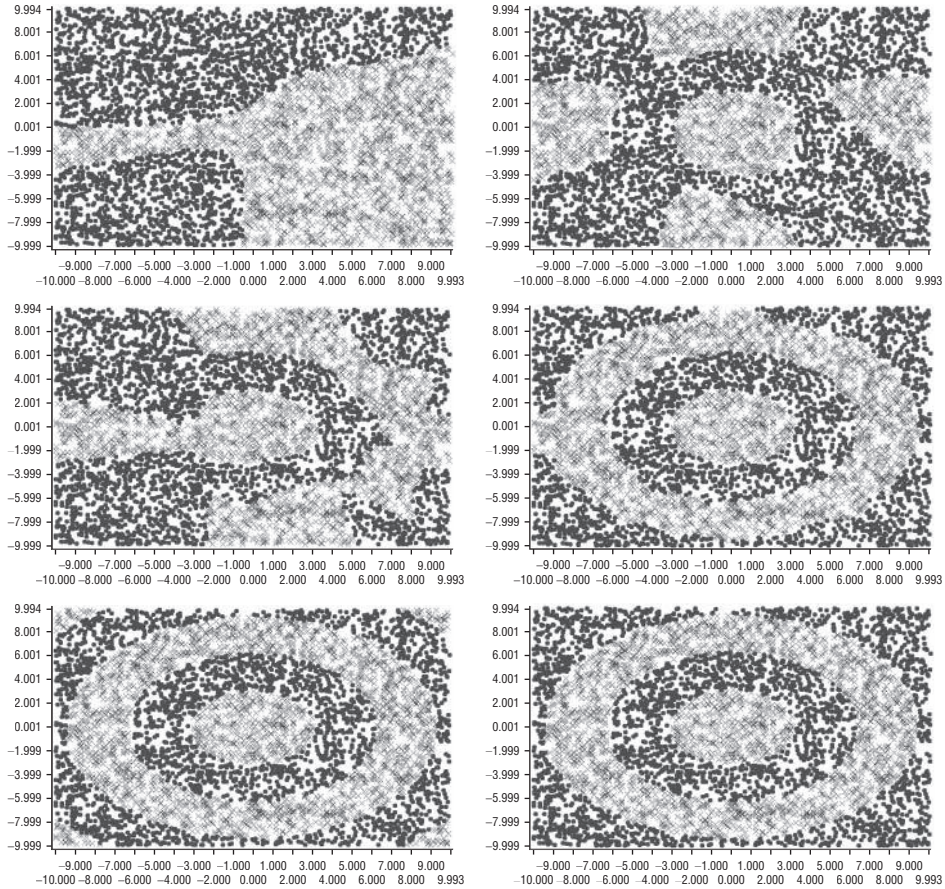


Figure 8-25: Decision boundaries as neural network learns after 10, 100, 200, 500, 1000, and 5000 epochs

Neural Network Settings

ANNs have several options available in software for the building of the neural networks:

- **Learning rate (for MLPs only):** Adjust the amount of change in weights per iteration. Larger learning rates help the MLP to train faster by making larger jumps. In addition, larger learning rates help the MLP guard against

converging to a local minimum. However, a learning rate that is too large could allow the MLP to jump over a global minimum. The solution some software uses to the dilemma of learning rate size is to schedule the learning rate so it decreases as the number of epochs increases. Good values for learning rates range from 0.01 to 0.5. The value 0.1 is a good starting value.

- **Learning rate (for MLPs only):** Adjust the amount of change in weights per iteration. Larger learning rates help the MLP to train faster by making larger jumps. In addition, larger learning rates help the MLP guard against converging to a local minimum. However, a learning rate that is too large could allow the MLP to jump over a global minimum. The solution some software uses to the dilemma of learning rate size is to schedule the learning rate so it decreases as the number of epochs increases. Good values for learning rates range from 0.01 to 0.5. The value 0.1 is a good starting value.
- **Momentum (for MLPs only):** Helps speed up convergence with backprop by smoothing weight changes by adding a fraction of the prior weight change to the current rate. It also has the effect, if the prior weight change and current weight changes have the same sign, of amplifying the magnitude of the weight adjustment. Good values of momentum range from 0.3 to 0.9, with the larger value.
- **Number iterations/epochs:** The maximum number of passes through the data for training the network. This is included for time savings and in some cases to guard against overfitting the network.
- **Number of hidden layers and number of neurons in each hidden layer:** This specifies the neural network architecture. While theoretically only one hidden layer is necessary, in practice more complex problems can benefit from adding a second hidden layer to the network. No theory exists to specify exactly how many neurons are included in a neural network, although more neurons are needed for noisier data and more complex data. Many software packages now provide the ability to search through a wide range of architectures so the modeler doesn't have to iterate manually.
- **Stopping criteria:** Some implementations of neural networks allow stopping once the training accuracy reaches an acceptable level so that overfitting the data is avoided. Some implementations allow training for a pre-specified number of minutes. Another option available in some software is to stop training when the error on testing data increases for many consecutive epochs, indicating that overfitting has begun in the network.
- **Weight updates:** Some implementations of neural networks also provide a setting for how often the weights are updated. The default is that weights are updated after every single record passes through the network. However, 100 or more records can be accumulated first, and the average error on these 100 records is computed to update the weights. This approach will speed up training considerably but at the expense of reducing the influence of some records that might produce large errors on their own.

The most popular learning algorithm for training neural networks is still the backpropagation algorithm, which is a first order gradient descent method. Dozens of alternatives to backpropagation exist and that was a great topic for doctoral dissertations in the 1990s.

- **Quick propagation:** Assumes the error surface is quadratic. If the slope of the error change with respect to the weight change (Δerror/Δweight, the error gradient) switches sign, an approximation to the bottom of the error parabola is computed, and the weights are adjusted to the value that achieves that minimum-error estimate. Do not let the change in weights exceed a maximum value. It is usually several times faster convergence than backprop.
- **Resilient propagation (Rprop):** Recognizes that the magnitude of the error gradient is often noisy, so it considers only the sign of the slope. If the error gradient is the same sign, increase the learning rate. If the value is not the same, reduce the learning rate. The amount of the increase or decrease is not always changeable.
- **Second-order methods:** These methods converge in far fewer epochs than backpropagation but require significantly more computation and memory. Algorithmically they are significantly more complex as well, which is a primary reason they are included in only a few tools. Conjugate Gradient, Gauss-Newton, and Levenberg Marquardt algorithms are three methods that are found in software tools.

Neural Network Pruning

Thus far, the assumption in building the neural network has been that an architecture is specified and a neural network is trained. But which architecture generates the smallest testing dataset error? A modeler can build multiple neural networks with different architectures and determine empirically which is best. A few software implementations build in the ability to build many networks with different architectures. However, what about the inputs? Are all of them necessary, or are some irrelevant at best or even harmful?

Pruning algorithms for neural networks exist but are rarely implemented in mainstream predictive analytics software. These algorithms progressively remove inputs and neurons from the network in an attempt to improve error on the test data. The simplest approach to removing inputs and neurons is to identify those that have the least influence on the network predictions; if the weights throughout a neural network associated with an input are small, the input is essentially being ignored by the network and can be safely removed and the network retrained without the input.

The advantage of pruning is that the final neural network will be faster to deploy because of the fewer inputs, neurons, and weights that create additional multiplies and adds in the model. Moreover, the network could become more stable by removing terms.

Building neural networks with pruning can take considerable time, however, and therefore it is often left as a final step in the modeling-building process, after a good set of inputs for the model are found and a good baseline architecture is found.

Interpreting Neural Networks

Neural networks have the reputation of being "black boxes," meaning that the interpretation of why predicted values are small or large cannot be determined. This reputation is unfair to a large degree; while neural networks are not transparent, the input variables that most influence the predictions can be determined in a variety of ways.

Most software implementations of neural networks generate a set of variable influence scores for each variable in the neural network. These scores are sometimes called *variable importance* or *influence*. The methods used to determine the influence of each variable fall into two general categories. The first category examines weights associated with each input (attached to the first hidden layer) and subsequent weights between the first hidden layer and the output layer. These approaches may use partial derivatives or just multiply the weights to derive the relative influence of the variables.

The second category determines the influence indirectly by changing values of each input and measuring the amount of change in the output. The larger the relative change, the more influence the variable has. Some techniques hold all variables but one constant at the mean and randomly vary a single variable from its mean value plus or minus one standard deviation. The variables are then scored by the relative change in the predictions as each of the inputs are wiggled. A more extreme version of this approach is to remove a single variable completely, retrain the neural network, and examine how much the error increases. This last technique has the danger, however, that the neural network might converge in such a way that the two networks with and without the input are not comparable; it is best to start the training of the network without the input with the same weights found in the network being assessed, if possible.

However, there are also other methods to determine which variables are influential. In one approach, decision trees are trained from the same inputs as were used in training the neural network, but the target variable for the tree is the neural network output rather than the actual target variable. The tree therefore finds the key patterns the neural network is finding with the advantage that the decision tree generates rules that are more easily interpreted than neural network weights.

Neural Network Decision Boundaries

Since neural networks are universal approximators and can fit very complex functions, many modelers believed they would never need to use logistic regression again. However, the reality is that sometimes the more complex neural networks either overfit the data or fail to converge to an optimum solution. In other words, just because a neural network *in theory* is a universal approximately doesn't mean that *in practice* it will find that solution. Therefore, it is usually a good idea to build models using several algorithms.

Nevertheless, neural networks are usually better predictors than k-NN, logistic regression, or decision trees. As you can see in Figure 8-26, decision boundaries for the nasadata dataset are largely linear, although there are some nonlinear regions, such as between rye, corn, and oats. Moreover, the nonlinear decision boundaries are very smooth.

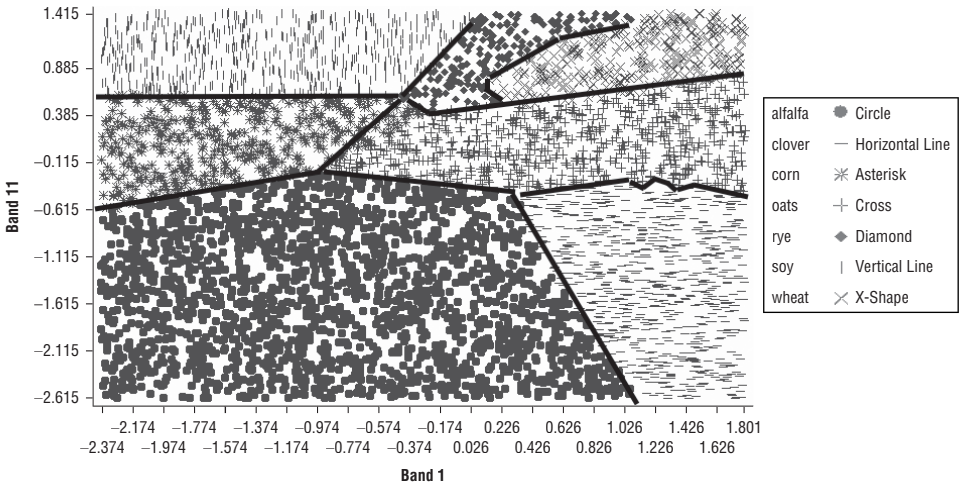


Figure 8-26: Neural network decision regions on nasadata

Other Practical Considerations for Neural Networks

ANNs, like other numeric algorithms including linear regression, logistic regression, k-nearest neighbor, and support vector machines requires numeric input data, and cannot have missing data. Typically, categorical data is represented numerically through the creation of dummy variables, as described in the discussion of logistic regression. Missing values can be imputed using typical imputation methods.

Variable selection for neural networks is done sometimes using the variable importance measure. For example, you may decide to keep only the top 20 variables in the network, and then retrain the neural network with only these variables. Some modelers use other algorithms that select variables automatically, like decision trees, to do variable selection for neural networks. This kind of variable selection is not optimal, but sometimes is practical.

K-Nearest Neighbor

The nearest neighbor algorithm is a non-parametric algorithm that is among the simplest of algorithms available for classification. The technique was first described by E. Fix and J. L. Hodges in 1951 in their paper "Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties" (a work that was later republished in 1989) and by 1967 had been studied sufficiently that its theoretical qualities were known, namely that k-NN is a universal approximator with worst-case error that is bounded. The precise nature of its theoretical properties is not a concern here; nearest neighbor has proven to be an effective modeling algorithm. Because of these properties and the ease of training the models, nearest neighbor is included in most predictive analytics software packages.

The nearest neighbor algorithm is a so-called "lazy learner," meaning that there is little done in the training stage. In fact, nothing is done: The training data is the model. In essence, the nearest neighbor model is a lookup table.

The k-NN Learning Algorithm

The k-NN learning algorithm is quite simple: It begins with the data itself. In other words, k-NN is merely a lookup table that you use to predict the target value of new cases unseen during training. The character "k" refers to how many neighbors surrounding the data point you need to make the prediction." The mathematics behind the k-NN algorithm resides in how you compute the distance from a data point to its neighbors.

Consider the simple example in Figure 8-27. There are two classes of data in the training set—one class colored black and the other colored gray—and a total of 13 data points in two dimensions (the x-axis direction and y-axis direction). The 13 data points therefore are the model. To classify a new data point, x, using the 1-NN algorithm, you compare each and every one of the data points in the training data to x, computing the distance calculated from each of these comparisons. In Figure 8-27, the closest data point is gray, so x will be labeled as gray.

However, suppose you instead use three nearest neighbors to classify x. Figure 8-28 shows the ring within which the three nearest neighbors reside. If you decide which class to predict for x by majority vote, the prediction this time is the black class, different from the prediction using the 1-NN model.

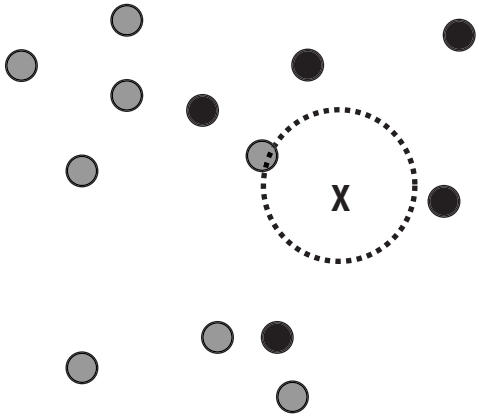


Figure 8-27: 1-nearest neighbor solution

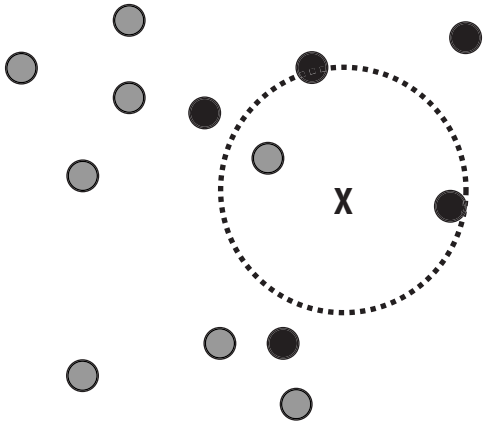


Figure 8-28: 3-NN solution

The more neighbors you include in the prediction, the smoother the predictions. For binary classification, it is common to use an odd number of nearest neighbors to avoid ties in voting.

There is no theory to tell you exactly how many neighbors to use. A small number of nearest neighbors can be very effective in finding small, subtle shifts in the data space, but can also be very susceptible to noise in the data. This is what appears to be the case in Figure 8-28: The gray dot inside the ring is an isolated example in that region, and a 3-NN solution is more robust. However, as the number of nearest neighbors increases, the classifier becomes less localized, smoother, less susceptible to noise, but also less able to find pockets of homogeneous behavior in the data, similar to a decision tree with only one split.

Therefore, the number of nearest neighbors you choose is often discovered through an iterative search, beginning with a small number of nearest neighbors, like 1, and continuing to increase the number of nearest neighbors until the testing data error rate begins to increase. Figure 8-29, for example, shows the decision regions for a 1-NN model built on the nasadata dataset with its seven crop types. The x axis is Band 1 and the y axis is Band 11. The shapes of the data points refer to the testing data classification predictions. Notice that while there are general, homogeneous regions for each crop type, there are still some local pockets of behavior with these regions. The *corn* crop type has some veins within the soy region in the upper left, and also in the *rye* region in the top-middle of the data space. The *wheat* predictions have a local pocket within the *rye* region, and *alfalfa* has a small group within the *clover* region in the lower right of the plot.

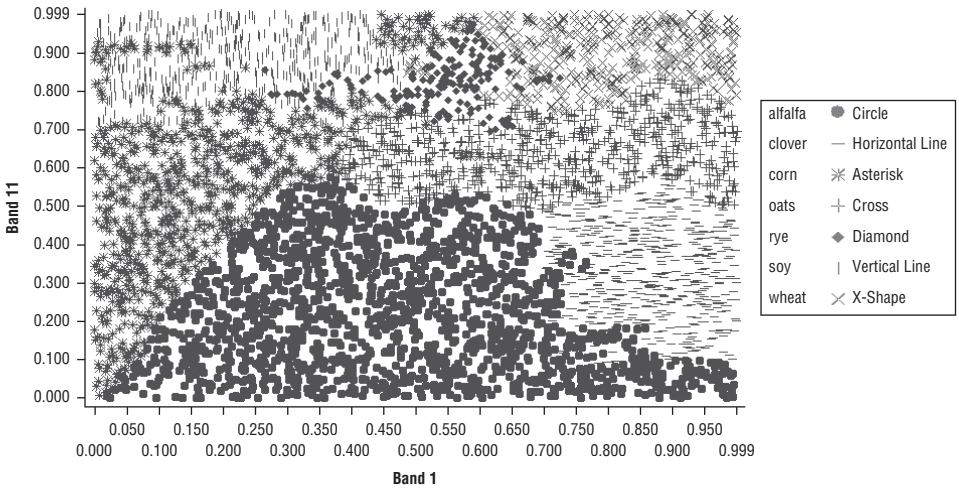


Figure 8-29: 1-NN decision regions for nasadata

Increasing the number of nearest neighbors to 3 reduces these localized pockets of behavior significantly, as shown in Figure 8-30. There are now only a few data points in the chart that do not correspond to the dominant crop type in the regions.

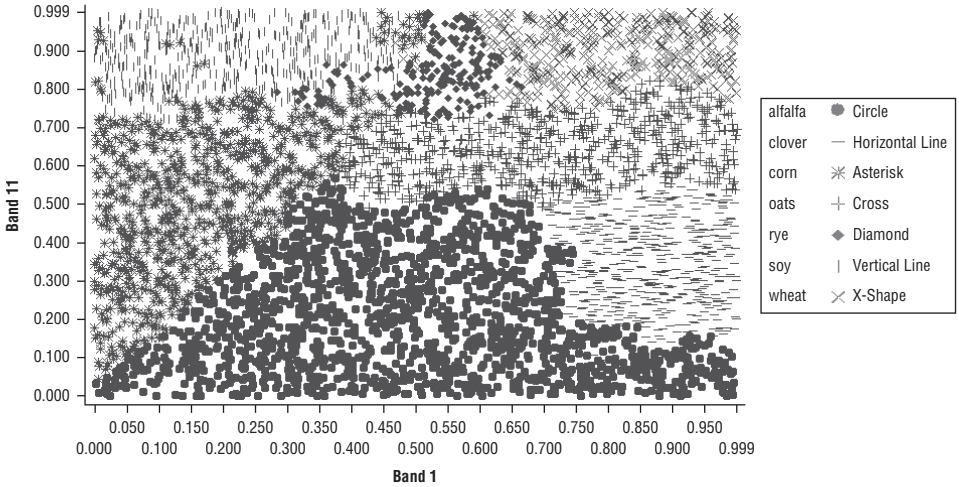


Figure 8-30: 3-NN decision regions for nasadata

When you increase k to 7, all seven regions become homogeneous, although not always smooth (see Figure 8-31).

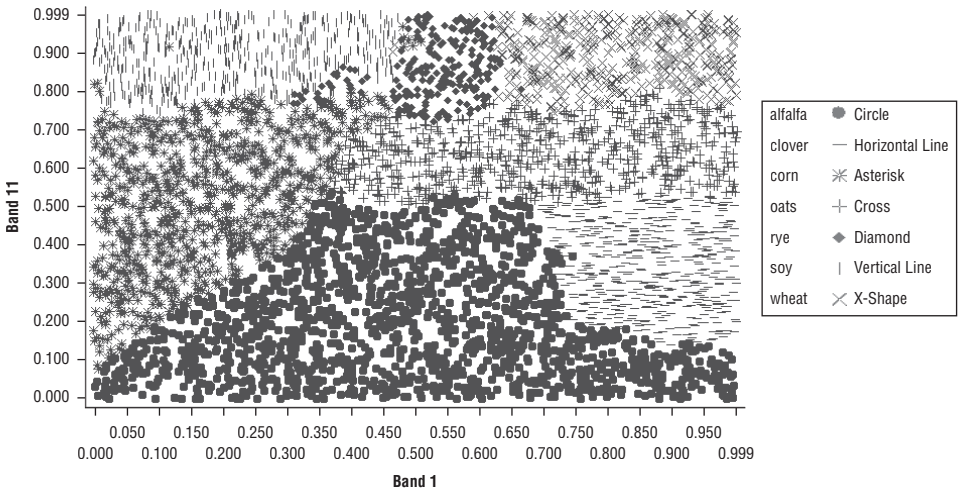


Figure 8-31: 7-NN decision regions for nasadata

In general, what value of k is correct? There is no theoretical answer to this question, but you can find an empirical solution by assessing model performance on testing data for each of the values of k you would like to test. As the value of k increases, the errors on testing data usually reach a minimum asymptotically.

For example, Table 8-9 shows the number of nearest neighbors, k, scored by AUC. The reduction in AUC from k equal to 101 to 121 is only 0.18%, showing convergence.

Table 8-9: The Number of Nearest Neighbors, K, Scored by AUC

NUMBER OF NN	AUC	% AUC REDUCTION
1	0.521	
3	0.536	2.83%
11	0.555	3.65%
21	0.567	2.14%
41	0.580	2.23%
61	0.587	1.23%
81	0.591	0.57%
101	0.594	0.66%
121	0.596	0.18%

Distance Metrics for k-NN

The primary distance metric used with the k-NN algorithm is Euclidean distance. Assume there are m records in the training data, and n inputs to the k-NN model. Collect the inputs into a vector called "x" of length n. The vector of inputs we are intending to predict based on the k-NN model is called y and is also of length n. The Euclidean distance can be expressed by the formula:

$$(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

where D is the Euclidean distance between the training vector x and the vector to classify, y. The summation merely adds the squares of the differences between training vector x and the vector to score called y. The distance will be computed m times, once for every record in the training data. The 1-NN algorithm finds the smallest of these distances and assigns to new data point y the target variable label of the nearest record.

Euclidean distance is the most commonly used distance metric, although others are sometimes found in software packages, including the Manhattan

distance, the Hamming distance, and the Mahalanobis distance. The Manhattan distance, rather than computing the squares of the differences between the training vector and the vector to score the new value, y, computes the absolute value of the differences.

$$D(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

The Mahalanobis distance is similar to the Euclidean distance except that the inputs are normalized by the covariance matrix. Covariance is analogous to variance (the square of the standard deviation) but measures the variance on two variables rather than a single one. To compute covariance, you multiply the standard deviation of variable 1, the standard deviation of variable 2, and the correlation between variables 1 and 2. The effect of normalizing by the covariance matrix is that the data is scaled to the unit sphere. The scaling takes into account the magnitudes of the variables if some variables have a larger spread than others.

Other Practical Considerations for k-NN

The k-NN algorithm usually has few options you need to set besides the value of k. Other common considerations you should consider include which distance metric to use, how to handle categorical variables, and how many inputs to include in the models.

Distance Metrics

Euclidean distance is known to be sensitive to magnitudes of the input data; larger magnitudes can dominate the distance measures. In addition, skew in distributions of the inputs can also effect the distance calculations. For example, consider a k-NN model built from two inputs from the KDD Cup 1998 data: RAMNTALL and CARDGIFT. The first 13 of 4,844 records in a sample of that data are shown in Table 8-10 and represent the training data. The mean value on the entire dataset (not just the 13 records shown here) for RAMNTALL is 112 and for CARDGIFT is 6.12, so RAMNTALL is more than 18 times larger on average than CARDGIFT.

Table 8-10: Two Inputs from KDD Cup 1998 Data

RECORD, I	RAMNTALL	CARDGIFT
1	98	6
2	119	9
3	61	10
4	123	12
5	68	6
6	102	14
7	132	5
8	94	8
9	38	3
10	30	5
11	44	2
12	25	1
13	35	2
Mean	112	6.12

If you assume that you are evaluating a new datapoint with the value of RAMNTALL equal to 98 and CARDGIFT equal to 6, Table 8-11 shows the distances from these 13 records in the training data. The distance in the first record represents the distance from the record to score to the first record in the training data. Notice that most of the distance values for RAMNTALL are much larger than they are for CARDGIFT (they are more than 10 times larger), precisely because RAMNTALL itself is larger. RAMNTALL therefore is the primary contributor to Total Distance.

Table 8-11: Euclidean Distance between New Data Point and Training Records

RECORD NUMBER IN TRAINING DATA	TRAINING RAMNTALL	TRAINING CARDGIFT	NEW RAMNTALL TO SCORE	NEW CARDGIFT TO SCORE	DISTANCE
1	0	5	98	6	5.00
2	21	3	98	6	21.21
3	37	4	98	6	37.22
4	25	6	98	6	25.71

RECORD NUMBER IN TRAINING DATA	TRAINING RAMNTALL	TRAINING CARDGIFT	NEW RAMNTALL TO SCORE	NEW CARDGIFT TO SCORE	DISTANCE
5	30	0	98	6	30.00
6	4	8	98	6	8.94
7	34	1	98	6	34.01
8	4	2	98	6	4.47
9	60	3	98	6	60.07
10	68	1	98	6	68.01
11	54	4	98	6	54.15
12	73	5	98	6	73.17

For this reason, practitioners usually transform the input variables so their magnitudes are comparable, a process often called *normalization*. Let's assume you are transforming RAMNTALL: The most common transformations applied to continuous data are shown in Table 8-12. The functions Mean, Min, Max, and StdDev are not presented in mathematical terms but rather in functional terms and refer to an operation on the entire column in the training data.

Table 8-12: Two Transformations for Scaling Inputs

TRANSFORMATION NAME	TRANSFORMATION FORMULA
Z-score	$RAMNTALL_z = \frac{(RAMNTALL - Mean(RAMNTALL))}{StdDev(RAMNTALL)}$
Min-max	$RAMNTALL_{minmax} = \frac{(RAMNTALL - Min(RAMNTALL))}{Max(RAMNTALL) - Min(RAMNTALL)}$

The z-score is very appealing for k-NN and other algorithms whose errors are based on Euclidean distance or squared error because the z-score has zero mean and unit standard deviation. A typical range for the z-scored variables will be between –3 and +3, and all variables will therefore have similar magnitudes, an important consideration for k-NN.

However, computing the z-score assumes the distribution is normal. If, for example, the distribution is severely skewed, the mean and standard deviation will be biased and the z-score units won't accurately reflect the distribution.

Moreover, and more importantly for k-NN, the values of the variable won't be in the range –3 to +3. RAMNTALL actually has z-score values as high as 18.

An alternative is to use the min-max transformation. The formula in Table 8-11 converts the variable from its original units to a range from 0 to 1. You could scale this result to the range –1 to +1 if you prefer to have a center value 0 in the transformed variable; you merely multiply the min-max transformation result by 2 and then subtract 1.

One advantage of the min-max transformation is that it is guaranteed to have bounded minimum and maximum values, and therefore the transformation will put all the transformed variables on the same scale. Also, if one is including dummy variables in the nearest neighbor model, they, too, will be on the scale of 0 to 1.

However, just as skewed distributions cause problems for z-scores, they also cause problems for the min-max transformation. Suppose RAMNTALL has one value that is 10,000, the second largest value is 1,000, and the minimum value is 0. The 10,000 value will map to the transformed value 1, but the second highest value will map to 0.1, leaving 90 percent of the range (from 0.1 to 1) completely empty except for the one value at 1. This variable would then be at a severe disadvantage for use in a k-NN model because its values are almost always less than 0.1, whereas other variables may have values distributed throughout the 0 to 1 range.

Therefore, you must take great care *before* applying any normalization that the distribution of the variable to be transformed is not heavily skewed and does not have outliers. Methods to treat these kinds of data were described in Chapter 4. The modeler must choose the course of action based on the pros and cons of each possible correction to the data.

Handling Categorical Variables

k-NN is a numerical algorithm requiring all inputs to be numeric. Categorical variables must therefore be transformed into a numeric format, the most common of which is 1/0 dummy variables: one column for each value in the variable.

If many key variables are categorical and dummies have been created to make them numeric, consider matching the continuous variables with the categorical variables by using min-max normalization for the continuous variables. However, k-NN models with many categorical variables can easily be dominated by the categorical variables because the distances generated by the dummy variables are either the minimum possible distance, 0 (if the new data point value matches a neighbor) or the maximum possible distance 1 (if the new

data point doesn't match the neighbor). The maximum distance will influence the overall distance more than continuous variables whose Euclidean distance will have the full range 0 to 1.

One alternative to min-max scaling is to rescale the dummy variables of the new data points from 1 to 0.7 so the maximum distance is reduced. A second alternative is to rescale the dummy variables so that the 0 values are coded as –2 and the 1 values are coded as +2, values that are smaller than the maximum and larger than the minimum.

The Curse of Dimensionality

One of the challenges with k-NN and other distance-based algorithms is the number of inputs used in building a model. As the number of inputs increases, the number of records needed to populate the data space sufficiently increases exponentially. If the size of the space increases without increasing the number of records to populate the space, records could be closer to the edge of the data space than they are to each other, rendering many, if not all, records as outliers. This is the curse of dimensionality. There is very little help from theory for us here; descriptions of how much space is too much, and how much additional variables can negatively impact k-NN models (if they do at all), have not been forthcoming.

One solution to the curse of dimensionality is to keep dimensionality low; include only a dozen to a few dozen inputs in k-NN models. Additionally, the quality of the inputs is important. If too many irrelevant features are included in the model, the distance will be dominated by noise, thus reducing the contrast in distances between the target variable values. Third, you should exclude inputs that are correlated with other inputs in the modeling data (exclude one of the two correlated variables, not both). Redundant variables give the false impression of smaller distances. However, reducing dimensionality too much will result in poorer predictive accuracy in models.

Weighted Votes

Some k-NN software includes an option to weight the votes of the nearest neighbors by the distance the neighbor is from the data point. The reason for this is to compensate for situations where some of the nearest neighbors are very far from the data point and intuitively would not be expected to contribute to the voting. The weighting diminishes the influence.

Naïve Bayes

The Naïve Bayes algorithm is a simplification of the Bayes classifier, an algorithm with a long and successful history. It's named after English mathematician and Presbyterian minister Thomas Bayes, developer of Bayes' theorem, or Bayes' rule, in the 1740s. But it wasn't until the 1930s that it became a contender in data analysis. It was proposed by Harold Jeffreys as an alternative to the approaches developed and advocated by English statistician Sir Ronald Aylmer Fisher, including the use of statistical tests and p-values. However, it wasn't until the 1980s that Bayesian methods become more mainstream in data analysis, and in the 1990s, the Naïve Bayes algorithm began appearing in technical writings and as an algorithm used in data mining competitions.

Bayes' Theorem

Before describing the Bayes' theorem and Naïve Bayes algorithm, first consider two variables, and a probability that each is "True" labeled A and B.

- P(A) is the probability that a variable has value A (or that A is the "true" value).
- P(~A) is the probability that a variable doesn't have the value A.
- P(B) is the probability that a second variable has the value B (or that B is the "true" value).
- P(~B) is the probability that the second variable doesn't have the value B.
- P(A and B) is the probability that both A and B are true.
- P(A or B) is the probability that either A or B is true.

The values P(A) and P(B) are called the *prior probabilities* that A is true or that B is true respectively, or just *priors*. These values are either set by a domain expert who knows the likelihood an event has occurred, calculated from historical data, or can be supposed in an experiment. Another way some describe these probabilities is as the initial degree of belief that A is true (or B is true).

Usually, in predictive modeling problems, the values are calculated from the data. However, as described in the Chapter 8 it is sometimes advantageous to override the historical values in the data if the modeler would like to alter how the algorithms interpret the variable.

A conditional probability is the probability that a condition is true, given that a second condition is true. For example,

P(A | B) is the probability that A is true given that B is true, and

P(B | A) is the probability that B is true given that A is true.

Some describe P(A|B) as a *posterior* probability: the degree of belief having accounted for B. Note that P(A|B) is not, in general, equal to P(B|A) because the two conditional probabilities presuppose different subsets of data (the former when B is true and the latter when A is true).

The conditional probabilities are formulated as follows:

$$P(A \mid B) = P(A \text{ and } B) \div P(B), \text{ and}$$

$$P(B \mid A) = P(A \text{ and } B) \div P(A)$$

If you rewrite these equations, you have:

$$P(A \text{ and } B) = P(A \mid B) \times P(B) \text{ and}$$

$$P(A \text{ and } B) = P(B \mid A) \times P(A)$$

This is sometimes called the "chain rule" of conditional probabilities.

Now consider Figure 8-32 with its abstract representation of these two conditions. The overlap between the two conditions is the "A and B" region in the figure. This region relates to the P(A and B) in the equations already described: P(A and B) is the product of the conditional probability P(A|B) and the prior of B, P(B). Or, stated another way, the intersection of the two sets is the conditional probability times the prior.

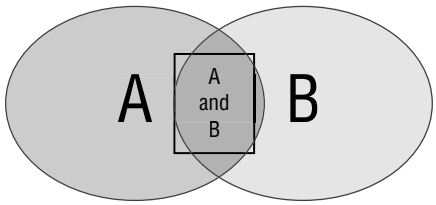


Figure 8-32: Conditional probability

Bayes' theorem tells you how to compute the probability that A is true given B is true, P(A|B), with the formula:

$$P(A \mid B) = P(B \mid A) \times P(A) \div P(B).$$

This is the form that you usually see in the literature, but the A and B values are just placeholders for actual conditions you find in their data.

For example, consider a doctor's office with a patient coming in who is sneezing (S) that wants to be able to diagnose if the patient has a cold (C) or pneumonia (P). Let's assume that there are probabilities that have already been computed or assumed, including:

$$P(S) = 0.3 \text{ (30 percent of patients come in sneezing)}$$

$$P(C) = 0.25 \text{ (25 percent of patients come in with a cold)}$$

$$P(P) = 1 \times 10^{-6} \text{ (1 in a million patients come in with pneumonia)}$$

These three are prior probabilities. The role of "A" in Bayes' theorem will be the probability the diagnosis is a cold, and then in a second computation, the probability that the diagnosis is pneumonia. In the language of predictive analytics, the target variables or the outcomes of interest are "cold" or "pneumonia."

You also need conditional probabilities for Bayes' theorem to work, namely P(S|C) and P(S|P), or in words, what is the probability that the patient sneezes because he has a cold, and what is the probability the patient sneezes because he has pneumonia. Let's assume both of these are 100 percent.

The solution for the cold is:

$$P(C \mid S) = P(S \mid C) \times P(C) \div P(S)$$

$$P(C \mid S) = 1.0 \times 0.25 / 0.3 = 0.83$$

$$P(P \mid S) = P(P \mid C) \times P(P) \div P(S)$$

$$P(P \mid S) = 1.0 \times 10^{-6} \div 0.3 = 3 \times 10^{-6}$$

In summary, the probability that the patient has a cold is 83 percent, whereas the probability the patient has pneumonia is only 3 chances in a million. Clearly, the disparity of the two probabilities is driven by the priors associated with the outcomes.

One key difference between the Bayes approach and other machine learning approaches described so far is the Bayes approach considers only one class at a time. The patient example had only two outcomes, so two calculations were made. For the nasadata, with seven target value classes, seven probabilities will be computed, one each for alfalfa, clover, corn, soy, oats, rye, and wheat. To predict a single outcome, you take the largest probability as the most likely outcome.

So far, the description of the Bayes classifier has been in probabilistic terms. Mathematically, for normal distributions, the Bayes classifier algorithm computes a mean value for every input (a mean vector) and the covariance matrix (the variance in one dimension, covariance in more than one). The maximum probability solution is the one with the smallest normalized distance to the mean of each population.

Consider the same simple data shown in Figure 8-27 for k-NN. With k-NN, the distance from the new value "x" and each of the data points was computed. The Bayes classifier, on the other hand, only needs to compute the distance from "x" to the mean of each class: gray and black in this case, or two distances. The rings around the mean values are normalized units of covariance. As you can see in Figure 8-33, "x" is closer to the gray class mean, less than one unit of covariance, than to the black class, so it is assigned the label "gray" by the classifier. In addition, the distance from the mean for each class can be appended to the data much like probabilities in logistic regression and confidences with neural networks.

This distance is not the Euclidean distance used in k-NN but a normalized distance, where the covariance matrix is the normalizer. The effect of this normalization is that magnitudes of the inputs are handled automatically (unlike k-NN), and rather than building piecewise linear decision boundaries as is done with k-NN, the Bayes classifier constructs quadratic decision boundaries, so it is more flexible and matches the normal distributions assumed in the data perfectly.

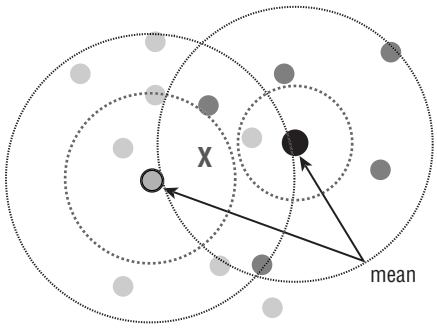


Figure 8-33: Bayes classifier distances

The Bayes classifier is an excellent algorithm, but what happens when there are many inputs to be considered in the probability calculations? Consider the patient example again, and instead of having just one symptom (sneezing), assume there are two symptoms: sneezing and fever (F). The Bayes' theorem solution for colds is now much more complex.

For two inputs, you must compute additional probabilities and conditional probabilities, including the following:

P(F) = probability of fever in patients

P(F and S | C)

P(F and ~S | C)

P(~F and S | C)

P(~F and ~S | C)

P(F and S | P)

P(F and ~S | P)

P(~F and S | P)

P(~F and ~S | P)

As the number of inputs increases, the number of conditional probabilities to compute also increases, especially those that include combinations of inputs. The problem becomes one of data size: Is there enough data to compute all of the

conditional probabilities? Have they all occurred often enough for these measures to be reliable? Mathematically, the Bayes classifier requires the computation of a covariance matrix. Is there enough data for these measures to be stable?

The Naïve Bayes Classifier

One solution to the problem of computing large numbers of conditional probabilities is to assume independence of the input variables, meaning you assume that input variables are unrelated to one another. With the assumption of independence, the conditional probabilities involving combinations of inputs are zero and you are left with only the conditional probabilities relating the output variable to the input variables, an enormous simplification. This is the assumption behind the Naïve Bayes classifier: We naively assume independence of inputs.

For the patient example, the calculation of the probability of having a cold from the measured inputs becomes:

$$P(C|S \text{ and } F) = P(S|C) \times P(F|C) \times P(C) \div P(S) \text{ and}$$
$$P(P|S \text{ and } F) = P(S|C) \times P(F|P) \times P(P) \div P(S)$$

The complex interaction of all the input variables has now become a simple multiplication of first order conditional probabilities. Not only does this simplify the building of models, but it also simplifies the interpretation.

If one applies the Naïve Bayes algorithm to the nasadata, just including Band1 and Band11 as inputs, the decision regions shown in Figure 8-34 are found. The shapes correspond to the maximum probability regardless of its value. The regions are largely linear and tend to be "boxy" due to the categorical inputs. These regions are very similar to those found by the 7-NN algorithm (refer to Figure 8-31) in the upper third of the scatterplot, though it does not have the nonlinear decision regions; the decision boundaries are all perpendicular or parallel to the Band 1 (x axis) and Band 11 (y axis).

Interpreting Naïve Bayes Classifiers

Naïve Bayes models are interpreted by examining the conditional probabilities generated in the training data. Most often, software reports these as lists of probabilities, like the ones shown in Table 8-13. The training data for the Naïve Bayes model is a stratified sample, with 50 0s and 50 percent 1s, so the baseline rate for comparison in the table is 50 percent. The model report is very easy to understand because each variable has its own list of probabilities. For RFA_2F, it is clear that when RFA_2F is equal to 4, 65 percent of the donors that match this value are responders, which is the group with the highest probability. Only

the RFA_2F equal to 1 group is lower than the average response rate equal to 50 percent.

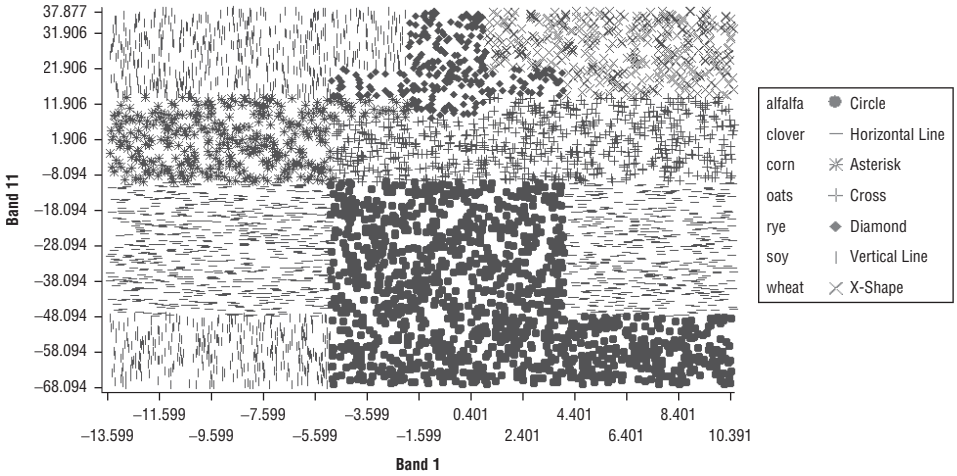


Figure 8-34: Naïve Bayes model for nasadata

Table 8-13: Naïve Bayes Probabilities for RFA_2F

CROSSTAB	RFA_2F = 1	RFA_2F = 2	RFA_2F = 3	RFA_2F = 4
Counts with TARGET_B = 0	1268	510	382	266
Counts with TARGET_B = 1, counts	930	530	471	494
Percent Target_B = 1	42.3%	51.0%	55.2%	65.0%

Each variable has its own table of results, and since Naïve Bayes assumes each input variable is independent, there are no tables of interactions between inputs. Some software packages also provide a table with a list of the most predictive input variables in the model, a report that is very useful to any analyst.

Other Practical Considerations for Naïve Bayes

Additional data preparation steps to consider when you build Naïve Bayes models include:

Naïve Bayes requires categorical inputs. Some implementations will bin the data for you, and others require you to bin the data prior to building the Naïve Bayes models. Supervised binning algorithms can help generate better bins than simple equal width or equal count bins.

Naïve Bayes is susceptible to correlated variables. Naïve Bayes can suffer from poor classification accuracy if input variables are highly correlated because of the assumption of independence and therefore the replication of probabilities from multiple variables in the model. It is best if the variables of only one of the correlated variables is included in the model.

Naïve Bayes does not find interactions. If you know that there are interactions in the data, create them explicitly for the Naïve Bayes model. This is similar to the strategy you use with logistic regression models to ensure the interactions are considered by the algorithm, and can improve accuracy significantly.

Regression Models

Regression models predict a continuous target variable rather than a categorical target variable. In some ways, this is a more difficult problem to solve than classification. Classification has two or a few values to predict: two for the KDD Cup 1998 dataset, three for the Iris data set, and seven for the nasadata dataset, to name three examples. The models have to predict the outcome correctly for these groups. Regression models must predict every value contained in the target variable well to have high accuracy.

Regression belongs to the supervised learning category of algorithms along with classification. I am using the term "regression" for this kind of model, but several other terms are used to convey the same idea. Some modelers call these models "continuous valued" prediction models; others call them "estimation" or "function estimation" models.

The most common algorithm predictive modelers use for regression problems is linear regression, an algorithm with a rich history in statistics and linear algebra. Another popular algorithm for building regression models is the neural network, and most predictive analytics software has both linear regression and neural networks. Many other algorithms are also used for regression, including regression trees, k-NN, and support vector machines. In fact, most classification algorithms have a regression form.

Throughout this section, the KDD Cup 1998 data will be used for illustrations, though instead of using the binary, categorical target variable TARGET_B that was used to illustrate classification modeling, a different target variable is predicted for regression problems, TARGET_D. This target variable is the amount a donor gave when he or she responded to the mail campaign to recover lapsed donors.

Linear Regression

Most analysts are familiar with linear regression, easily the most well-known of all algorithms used in regression modeling. Even analysts who don't know about linear regression are using it when they are adding a "trend line" in Excel. Figure 8-35 shows a scatterplot with a trend line added to show the best fit of the data points using linear regression. The linear regression algorithm finds the slope of the output with respect to the input. In the model in Figure 8-35, every time the variable on the x axis increases 10 units, the target variable on the y axis increases approximately 2.5 units.

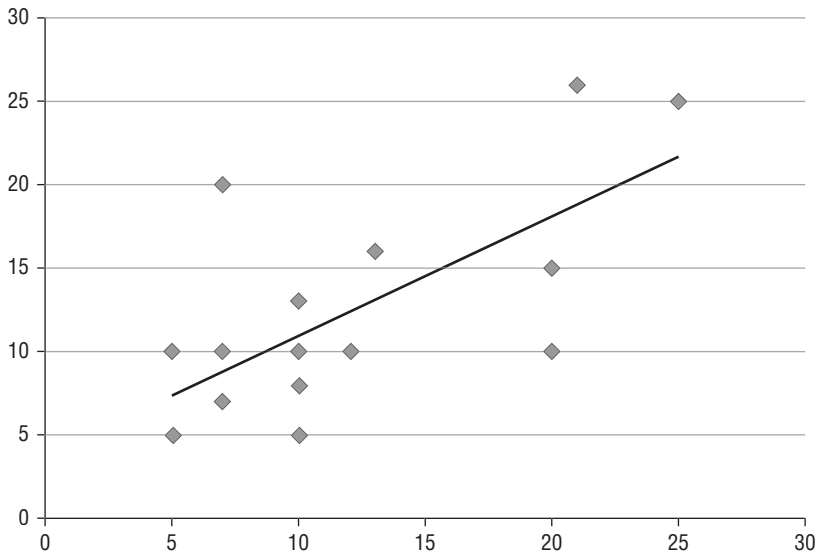


Figure 8-35: Regression line

This discussion presents linear regression from a predictive modeler's perspective, not a statistician's perspective. Predictive modelers usually use linear regression in the same way other algorithms are used, primarily to predict a target variable accurately.

The form of a linear regression model is identical to the equations already described for the neuron within a neural network:

$$y_i = w_0 + w_1 \times x_1 + w_2 \times x_2 + \ldots + w_n \times x_n$$

As before, the *y* variable is the target variable—the output. The inputs to the model are *x1*, *x2*, and so forth. The weights or coefficients are *w0*, *w1*, *w2*, and so forth. To be more precise, however, linear regression only includes the output variable, *y*, and a single input variable, *x*; we regress the target variable on the input. The full equation is multiple regression where we regress the target variable on all the input variables.

Often, the variables *y* and *x* have subscripts, like the letter *i* in the regression equation just shown, which refer to the record number; each record will have its own predicted value based on the input values in that record. In addition, the predicted target value is often represented with a carat above it, usually pronounced as "hat," indicating it is a predicted value in contrast to the actual value of the target variable, *y*.

The difference between the target variable value in the data and the value predicted by the regression model, *y* minus *y* hat, is called a *residual* and is often annotated with the letter *e*, as in the equation:

$$e = y_i - \hat{y}_i$$

Strictly speaking, residuals are different than the regression error term, which is defined as the difference between the target variable value and what statisticians call the *true regression line*, which is the perfect regression model that could be built if the data satisfied all of the regression assumptions perfectly, and the training data contained all of the patterns that could possibly exist. Predictive modelers don't usually include any discussions or descriptions of this kind of idealized model, and therefore often treat the terms residuals and errors synonymously.

Residuals are visualized by dropping a perpendicular line between the data value and the regression line. Figure 8-36 shows the target variable, TARGET_D, fit with a regression line created from input variable LASTGIFT. The residuals are shown as vertical lines, the distance between the actual value of TARGET_D and the predicted value of the TARGET_D (*y* hat in the equation) as predicted by the linear model. You can see there are large errors above and below the regression line. If one or more of the data points above the regression line were removed, the slope of the regression line would certainly change.

A plot of the residuals versus LASTGIFT is shown in Figure 8-37. Note that magnitudes of residuals are approximately the same regardless of the value of LASTGIFT, the residuals are both positive and negative, and there is no shape to the residuals. If, however, the shape of the residuals versus LASTGIFT is quadratic, there is a quadratic relationship between LASTGIFT and TARGET_D that is missing in the model and should be included by adding the square of LASTGIFT as an input. Any detectable pattern in the residual plot indicates that one or more derived variables can be included to help improve the fit of the model.

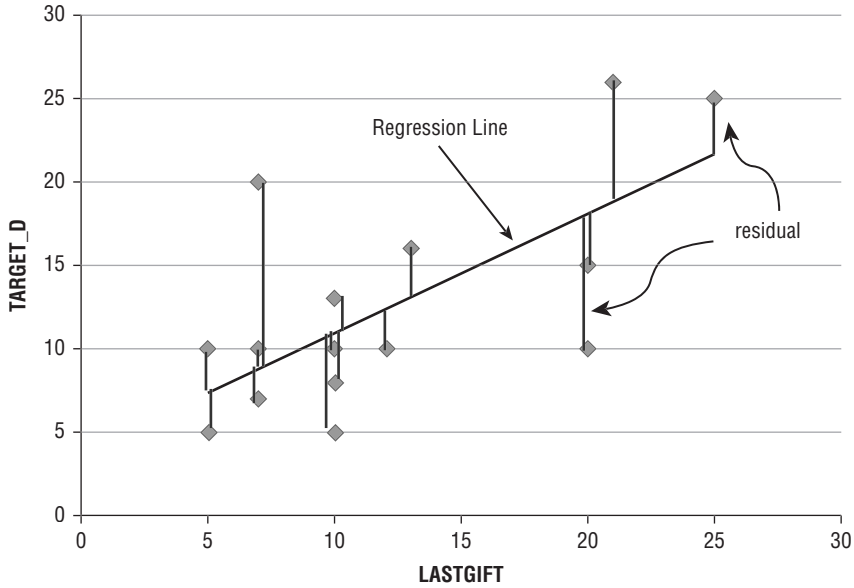


Figure 8-36: Residuals in linear regression

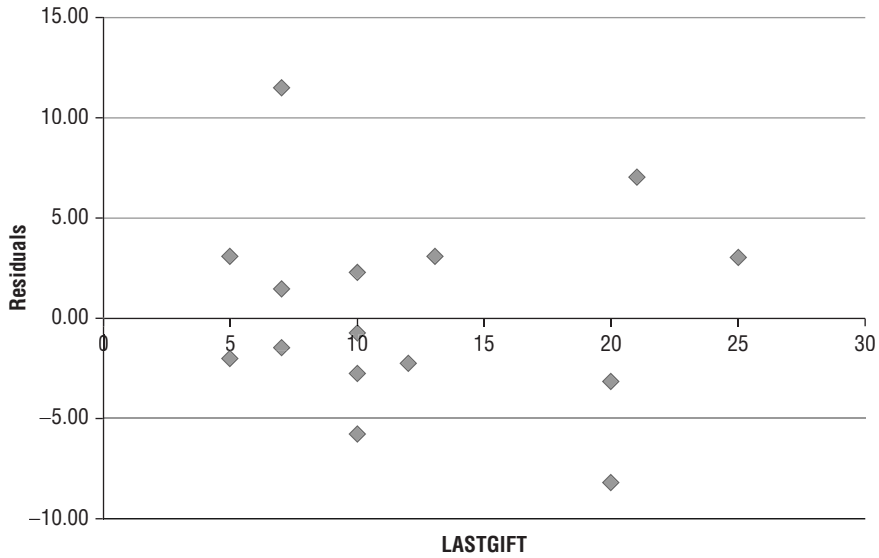


Figure 8-37: Residuals vs. LASTGIFT

The influence of large residuals is magnified because the linear regression minimizes the *square* of the residuals. The reason that outliers, especially outliers that have large magnitudes compared to the rest of the input values, create

problems for regression models is because of the squared error metric. This is also the reason that positive skew has such influence on regression models; models will be biased toward trying to predict the positive tail of a distribution because of the squared error metric.

However, this is not always bad. Consider the KDD Cup 1998 data again. TARGET_D is positively skewed, as most monetary variables are. Therefore, linear regression models trying to predict TARGET_D will be biased toward predicting the positive tail of TARGET_D over predicting the smaller values of TARGET_D. But is this bad? Don't we want to identify the large donors well, perhaps even at the expense of worse predictive accuracy for smaller donors? The question of transforming TARGET_D is not just a question of linear regression assumptions; it is also a question of which cases we want the linear regression model to predict well.

Linear Regression Assumptions

The linear regression algorithm makes many assumptions about the data; I've seen lists of assumptions numbering anywhere from four to ten. Four of the assumptions most commonly listed include the following. First, the relationship between the input variables and the output variable is assumed to be linear. If the relationship is not linear, you should create derived variables that transform the inputs so that the relationship to the target becomes linear. Figure 8-38 shows what appears to be a quadratic relationship between the input on the x axis and the output on the y axis. The linear fit from a regression model is shown by the line, which obviously doesn't fully capture the curvature of the relationship in the data: The data violates the assumption that the relationship is linear.

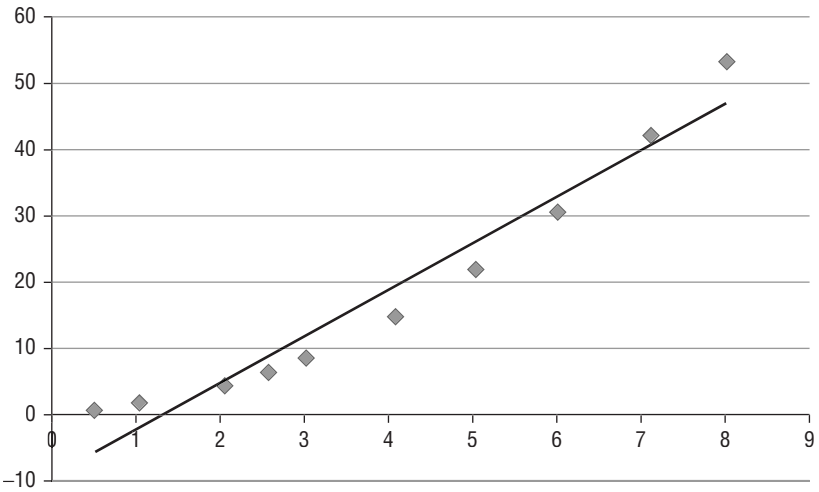


Figure 8-38: Linear model fitting nonlinear data

A residual plot, like Figure 8-37, would reveal the quadratic relationship between the residuals and the input presented by the x axis. The corrective action is to create a derived variable to linearize the relationship. For example, rather than building a model using the input directly, if the square of the input is used instead, the scatterplot is now linear, as shown in Figure 8-39.

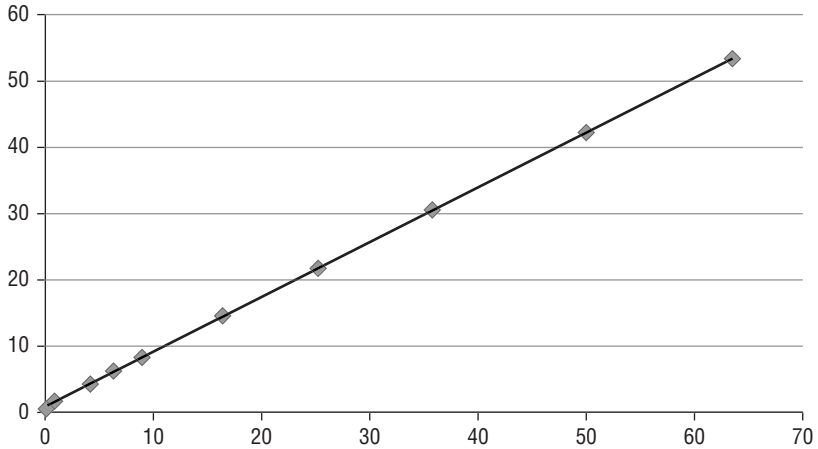


Figure 8-39: Linear model after transforming nonlinear data

The second assumption is that the inputs are uncorrelated with each other, or, in other words, the correlations between input variables are all zero. If this isn't the case, then the regression coefficient for one variable may carry some of the influence from another variable, blurring the influence of a variable on the outcome. Consider the following model for TARGET_D:

$$\text{TARGET\_D} = 2.3261 + 0.5406 \times \text{LASTGIFT} + 0.4319 \times \text{AVGGIFT}$$

This equation says that every increase in LASTGIFT of $10 will produce an increase in the amount the lapsed donor gives to the recovery mailing (TARGET_D) $5.41 (5.406 rounded to the nearest penny). However, LASTGIFT and AVGGIFT are correlated with a correlation coefficient equal to 0.817. If AVGGIFT is removed from the model, the new regression equation is:

$$\text{TARGET\_D} = 3.5383 + 0.7975 \times \text{LASTGIFT}$$

Now the interpretation of the relationship between LASTGIFT and TARGET_D is different: Every increase in LASTGIFT of $10 will produce an increase of $7.975 in the amount the lapsed donor gives. The difference is the correlation between the two variables, meaning they share variance, and therefore, they share influence on the target variable.

High levels of correlation, therefore, should be avoided if possible in regression models, though they cause more problems for interpretation of the models

rather than the model accuracy. However, if variables are correlated extremely high, 0.95 or above as a rule of thumb, the correlation can cause instability that is sometimes called *destructive collinearity*. One symptom of this is seen when the linear regression coefficients of two highly correlated terms (positively) are both very high and opposite in sign. In these cases, the two terms cancel each other out and even though the terms appear to be influential, they are not, at least not on the training data. However, if new data contains values of these two variables that deviate from the high correlation, the large coefficients produce wild fluctuations in the predicted values.

Most statistics software contains collinearity diagnostics that alert you when there is dangerous collinearity with the inputs to the model. Even if you have already removed highly correlated variables during Data Preparation, the collinearity could exist in three or more variables. Because of this, some practitioners routinely apply Principal Component Analysis to find these high levels of correlation before selecting inputs for modeling. If you used the principal components (PC) themselves as inputs to a linear regression model, you are guaranteed to eliminate collinearity because each PC is independent of all others, so the correlations are zero. If you instead use the single variable that loads highest on each PC, you will reduce the possibility for collinearity.

The remaining assumptions you find in the literature all relate to the residual, which are assumed to be normally distributed with a mean of zero and equal. In other words, residuals should contain only the noise in the data, also referred to as the unexplained variance.

This last set of assumptions, especially the normality assumption, is the reason many analysts transform inputs so that they are normally distributed. For example, with positively skewed inputs, many will transform the inputs with a log transform, and this is exactly the approach that was described in Chapter 4. This is not necessary, strictly speaking, to comply with regression assumptions, but beginning with inputs and outputs that are normally distributed increases the likelihood that the assumptions will be met.

Variable Selection in Linear Regression

Linear regression fits all of the inputs to the target variable. However, you will usually do some variable selection, especially during the exploratory phase of building models. This step is very important to modelers: Identifying the best variables to include in the model can make the difference between deploying an excellent model or a mediocre model.

Perhaps the simplest method of variable selection is called *forward selection*, which operates in the same way variable selection occurs in decision trees. In forward selection, a model is built using all of the input variables, one at a time. The variable with the smallest error is kept in the model. Then, all of the remaining variables are added to this variable one at a time, and a model is built

for each of these combinations. The best two-term model is kept. This process continues until a condition applies to stop the adding of more variables, most often to prevent the model from overfitting the training data.

Some software provides the option to do *backward selection*, where the first model is built using all of the inputs, each variable is removed one at a time from the set of inputs, a model is built using each of these subsets, and the best model is retained, now with one variable removed. The process of removing the variable that contributes the least to reducing fitting error continues until no input variables remain or a stop condition applies.

A third approach, called *stepwise* variable selection, is a combination of these two. The algorithm begins by finding the best single variable (forward selection), and alternatively adds a new variable or removes a variable until a stop condition applies. Stepwise regression is preferred because it usually finds better variable subsets to include in the regression model.

How does forward, backward, or stepwise regression decide which variables to include or remove? Chapter 4 described several methods that are frequently used, including the Akaike information criterion (AIC) and minimum description length (MDL). Other metrics that are sometimes used include the Bayesian information criterion (BIC) and Mallow's *Cp*. All of these metrics function the same way: They compute a complexity penalty for the regression model from the number of inputs in the model and the number of records the model is built from. The more records you have, the more terms that can be added to the model without incurring a higher penalty.

Figure 8-40 shows a representation of what these metrics do. As the number of inputs in a regression model increases, the error of the regression model (fitting error in the figure) becomes smaller. Each term added to the model, however, incurs a complexity penalty that grows linearly with the number of terms in the model. These two numbers added together provide a number that trades off fitting with complexity. In the figure, five inputs provide the best tradeoff between fitting error and complexity.

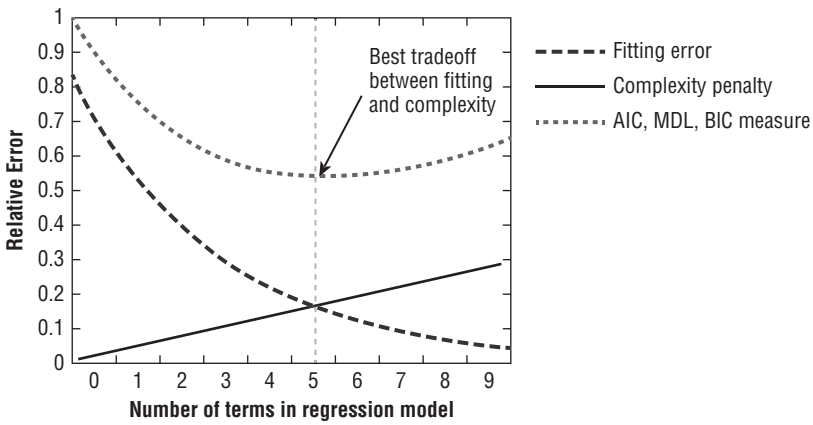


Figure 8-40: Trading fitting error and complexity

Interpreting Linear Regression Models

Linear regression equations are interpreted in two ways: through the coefficients and the *p* values. The coefficients describe the amount of change to expect in the target variable prediction for every unit change in the input variable. This kind of interpretation has already been discussed. But which input variables are most influential in predicting the target variable? If all the inputs were scaled to the same range, such as with min-max normalization or z-score normalization, the regression coefficients themselves show the influence of the inputs, and the input associated with the largest coefficient magnitude is the most important variable.

Consider the simple linear regression model summarized in Table 8-14, predicting TARGET_D. The largest coefficient in the model (excluding the constant) is RFA_2F. The values of RFA_2F are 1, 2, 3, and 4. However, the smallest *p* values are for LASTGIFT and AVGGIFT, whose *p* values don't show any value at all to four significant digits. These two are the strongest predictors in the model. An average value of LASTGIFT, 14, is nearly seven times larger than the average value for RFA_2F. If RFA_2F had the same influence on TARGET_D as LASTGIFT, its coefficient would be seven times larger than LASTGIFT.

Table 8-14: Regression Model Coefficients for TARGET_D Model

VARIABLE	COEFFICIENT	*P*
Constant	13.4704	0.0473
LASTGIFT	0.4547	0.0000
AVGGIFT	0.4478	0.0000
RFA_2F	−0.5847	0.0002
NGIFTALL	−0.0545	0.0349
FISTDATE	−0.0009	0.2237

Some modelers use a *p* value of 0.05 as the rule of thumb to indicate which variables are significant predictors and which should be removed from the model. Using this principle, FISTDATE is not a significant predictor and therefore should be removed from the model.

However, this approach has several problems. First, there is no theoretical reason why 0.05 is the right threshold. As the number of records increases, the *p* values for inputs will decrease even if the model is no more predictive just because of the way *p* is calculated. Second, there are other more direct ways to assess which inputs should be included, such as AIC, BIC, and MDL. Third, just because a variable is not a significant predictor doesn't necessarily mean

that it is harmful to the model. In this model, removing FISTDATE from the model actually reduces model accuracy on testing data according to one common measure, R^2, from 0.589 to 0.590, a tiny change. Clearly, FISTDATE isn't contributing significantly to model accuracy, nor is it contributing to overfitting the target variable.

Using Linear Regression for Classification

Linear regression is usually considered as an algorithm that applies only to problems with continuous target variables. But what would happen if linear regression tried to predict a categorical target variable coded as 0 and 1? Figure 8-41 shows what this kind of problem would look like. Linear regression finds a line that tries to minimize the squares of the errors. With only two values of the target variable, the regression line obviously cannot fit the data well at all. Notice that what it tries to do in the figure is put the line in the center of the density of data points with values 0 and 1.

A regression fit of the dummy variable, while creating a poor estimate of the values 0 and 1, can create a score that rank-orders the data. In the figure, it is clear that the smaller the values of the input, the larger the value of the predicted value. Intuitively this makes sense: Smaller values of the input are more likely to have the value 1 because they are further from the place where the target switches from the value 1 to the value 0.

In practice, I have found this phenomenon true for problems with a business objective that calls for rank-ordering the population and acting on a subset of the scored records, such as fraud detection, customer acquisition, and customer attrition. In fact, for some problems, the linear regression model sometimes is the best model for selecting records at the top end of the rank-ordered list.

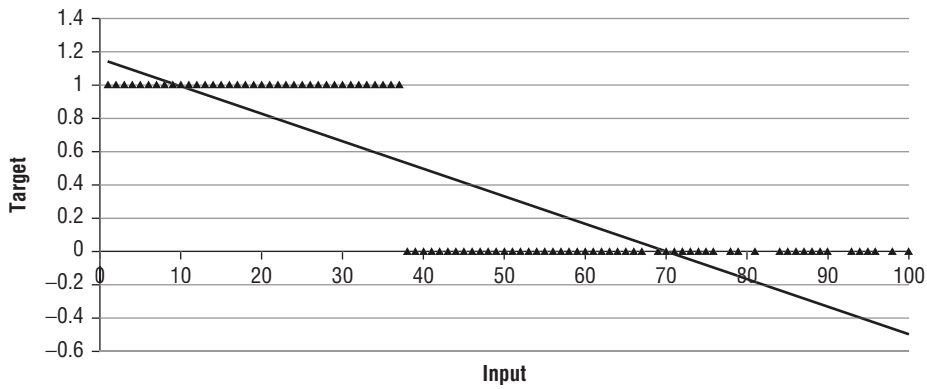


Figure 8-41: Using linear regression for classification

Other Regression Algorithms

Most classifications also have a regression form, including neural networks, decision trees, k-NN, and support vector machines. Regression Neural Networks are identical to classification neural networks except that output layer nodes usually have a linear activation function rather than the usual sigmoidal activation function. The linear activation function actually does nothing at all to the linearly weighted sum. They therefore behave very similarly to a linear regression function. Figure 8-42 shows how the linear activation function is represented in neural network pictures.

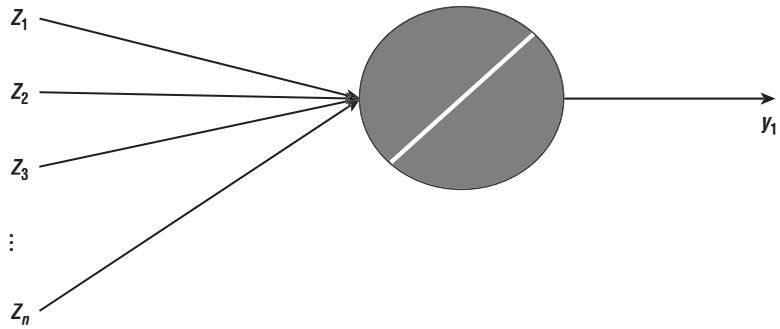


Figure 8-42: Linear activation function for regression output nodes

The remaining nodes in the neural network, including all the neurons in the hidden layer, still have the sigmoid activation functions. The neural network is still a very flexible nonlinear model. The advantage over linear regression is clear; neural networks can find nonlinear relationships between inputs and a continuous-valued target variable without the need to create derived variables.

Regression trees use the same decision tree form, but terminal nodes compute the mean or median value of the continuous target value. The ideal regression trees will create terminal nodes with small standard deviations of the target variable, and the mean or median values in the terminal nodes will be distinct from one another. Regression trees have the same advantages over numeric algorithms that classification trees have. They are insensitive to outliers, missing values, and strange distributions of inputs and the target variable.

Modifying the k-NN algorithm for regression is simple. Rather than voting to determine the value of a new data point, an average of the target value for nearest neighbors is computed. All other aspects of the k-NN algorithm are the same, including the principles for data preparation.

Summary

This chapter described the predictive modeling algorithms most commonly found in software, and therefore the algorithms most commonly used in practice. All of the predictive modeling algorithms described in this chapter fall under the category of supervised learning, which predicts one or more target variables in the data from a set of candidate input variables. Classification algorithms predict categorical target variables, and regression algorithms predict continuous target variables.

Each algorithm comes with its own set of assumptions about the data, and has its own strengths and weaknesses. I recommend that predictive modelers learn at least three different algorithms well enough to understand how to modify the parameters that change the way the models are built. Additionally, understanding the assumptions algorithms make about data helps you diagnose problems when they arise. This mindset develops as you build more models and experience models performing well and poorly.

CHAPTER

9

Assessing Predictive Models

This chapter explains how to assess model accuracy so that you can select the best model to deploy and have a good estimate of how well the model will perform operationally. In some ways, this should be a very short chapter. If you assess model accuracy by using percent correct classification for classification problems or average squared error for regression problems, your choices would be simple. However, because the choice of the model assessment metric should be tied to operational considerations rather than algorithmic expedience, you should keep several methods of model assessment in your toolbox.

Every algorithm has its own way of optimizing model parameters. Linear regression minimizes squared error. Neural networks minimize squared error or sometimes cross-entropy; k-NN minimizes the distance between data points. However, businesses often don't care about the root mean squared error, R-squared, the Gini Index, or entropy. Rather, the business may care about return on investment, expected profit, or minimizing false alarm rates for the next best 1,000 cases to be investigated. This chapter explores the most common ways models are assessed from a business perspective.

Model assessment should be done first on testing data to obtain an estimate of model accuracy for every model that has been built. Two decisions are made based on test data assessment: an assessment of which model has the best accuracy and therefore should be selected for deployment, and a conclusion if model accuracy is high enough for the best model to be used for deployment. The validation data can also be used for model assessment but usually only to

estimate the expected model accuracy and performance once it is deployed. Throughout discussions in the chapter, the assumption is that model assessment for selection purposes will be done using test data.

Batch Approach to Model Assessment

The first approach to assessing model accuracy is a *batch* approach, which means that all the records in the test or validation data are used to compute the accuracy without regard to the order of predictions in the data. A second approach based on a rank-ordered sorting of the predictions will be considered next. Throughout this chapter, the target variable for binary classification results will be shown as having 1s and 0s, although any two values can be used without loss of generality, such as "Y" and "N," or "good" and "bad."

Percent Correct Classification

The most straightforward assessment of classification models is percent correct classification (PCC). The PCC metric matches the predicted class value from a model with actual class value. When the predicted value matches the actual value, the record is counted as a correct classification, and when they don't match, the record is counted as an incorrect classification.

Consider a binary classification problem with 50 percent of the records labeled as 1 and the remaining 50 percent labeled as 0. The worst you can do with a predictive model built properly is random guessing, which can be simulated by flipping a fair coin and using heads as a guess for 1 and tails as a guess for 0; a random guess would be correct 50 percent of the time for binary classification. (It is possible for a model to predict worse than a random guess on testing data if it has been overfit on the training data, but we assume care has already been taken to avoid overfit.) A well-built classifier therefore will always have PCC on testing or validation data greater than or equal to 50 percent, and less than or equal to 100 percent.

The *lift* of a model is the ratio of model accuracy divided by the accuracy of a baseline measure, usually the expected performance of a random guess. If the proportion of 1s is 50 percent, the minimum lift of a model, if it does no better than a random guess, is 1 (50 percent ÷ 50 percent), and the maximum lift of a model, if it is a perfect predictor, is 2 (100 percent ÷ 50 percent).

What if the proportion of 1s is not 50 percent, but some smaller percentage of the data? The baseline PCC from a random guess will always be the proportion of 1s in the data, and lift is the ratio of PCC to this original proportion. If the proportion of 1s in the testing data is 5 percent, the baseline (random) expected performance of a classifier will be 5 percent correct classification, and the maximum lift of a model is now 20: 100 percent PCC ÷ 5 percent at random.

Therefore, the lower the baseline percentage of 1s in the target variable, the higher the potential lift that is achievable by a model. This method of calculating lift works for classification problems regardless of the number of levels in the target variable.

If there are more than two levels for the target variable, PCC is still computed the same way: the proportion of records classified correctly. However, the more levels in the target variable, the more difficult the classification problem is and the lower the random baseline accuracy is. For the three-class problem, if the proportion of each value is 33.3 percent, a random guess is also 33.3 percent and therefore the maximum lift is 3 instead of 2 for binary classification. In general, the maximum lift can differ by class if the proportion of each class value is different from the others. If, for example, class 1 is represented in 10 percent of the records, class 2 in 20 percent of the records, and class 3 in 70 percent of the records, you would expect that a random guess would estimate 10 percent of the records belong to class 1, 20 percent belong to class 2, and 70 percent belong to class 3. The maximum lift values for estimates of class 1, class 2, and class 3 are therefore 10 (100 percent ÷ 10 percent), 5 (100 percent ÷ 20 percent), and 1.4 (100 percent ÷ 70 percent).

Table 9-1 summarizes the differences between the possible lift compared to the baseline rate. It is immediately obvious that the smaller the baseline rate, the larger the possible lift. When comparing models by the lift metric, you must always be aware of the base rate to ensure models with a large baseline rate are not perceived as poor models because their lift is low, and conversely, models with very small baseline rates aren't considered good models even if the lift is high.

Table 9-1: Maximum Lift for Baseline Class Rates

BASELINE RATE OF CLASS VALUE	MAXIMUM LIFT
50 percent	2
33 percent	3
25 percent	4
10 percent	10
5 percent	20
1 percent	100
0.1 percent	1000

Table 9-2 shows 20 records for a binary classification problem. Each record contains the target variable, the model's predicted probability that the record belongs to class 1, and the model's predicted class label based on a probability threshold of 0.5. (If the probability is greater than 0.5, the predicted class label is assigned 1;

otherwise, it is assigned the value 0.) In this sample data, 13 of 20 records are classified correctly, resulting in a PCC of 65 percent.

Table 9-2: Sample Records with Actual and Predicted Class Values

ACTUAL TARGET VALUE	PROBABILITY TARGET = 1	PREDICTED TARGET VALUE	CONFUSION MATRIX QUADRANT
0	0.641	1	false alarm
1	0.601	1	true positive
0	0.587	1	false alarm
1	0.585	1	true positive
1	0.575	1	true positive
0	0.562	1	false alarm
0	0.531	1	false alarm
1	0.504	1	true positive
0	0.489	0	true negative
1	0.488	0	false dismissal
0	0.483	0	true negative
0	0.471	0	true negative
0	0.457	0	true negative
1	0.418	0	false dismissal
0	0.394	0	true negative
0	0.384	0	true negative
0	0.372	0	true negative
0	0.371	0	true negative
0	0.341	0	true negative
1	0.317	0	false dismissal

Confusion Matrices

For binary classification, the classifier can make an error in two ways: The model can predict a 1 when the actual value is a 0 (a *false* alarm), or the model can predict a 0 when the actual value is a 1 (a *false* dismissal). PCC provides a measure of classification accuracy regardless of the kind of errors: Either kind of error counts as an error.

A confusion matrix provides a more detailed breakdown of classification errors through computing a cross-tab of actual class values and predicted class values, thus revealing the two types of errors the model is making (false alarms and false dismissals) as well as the two ways the model is correct (true positives and true negatives), a total of four possible outcomes. In Figure 9-1, these four outcomes are listed in the column Confusion Matrix Quadrant.

The diagonal of the confusion matrix represents correct classification counts: true negatives, when the actual and predicted values are both 0, and true positives when both the actual and predicted values are both 1. The errors are on the off-diagonal, with false negatives in the lower-left quadrant when the actual value is 1 and predicted value is 0, and false positives in the upper-right quadrant when the actual value is 0 and the predicted value is 1. A perfect confusion matrix, therefore, has values equal to 0 in the off-diagonal quadrants, and non-zero values in the diagonal quadrants.

Once you divide the predictions into these four quadrants, several metrics can be computed, all of which provide additional insight into the kinds of errors the models are making. These metrics are used in pairs: sensitivity and specificity, type I and type II errors, precision and recall, and false alarms and false dismissals. Their definitions are shown in Table 9-3. Engineers often use the terms *false alarms* and *false dismissals*; statisticians often use *type I* and *type II* errors or *sensitivity* and *specificity*, particularly in medical applications; and computer scientists who do machine learning often use *precision* and *recall*, particularly in information retrieval. The table shows that the definitions of several of these measures are identical.

Some classifiers will excel at one type of accuracy at the expense of another, perhaps having a high sensitivity but at the expense of incurring false alarms (low specificity), or *vice versa*. If this is not a desirable outcome, the practitioner could rebuild the model, changing settings such as misclassification costs or case weights to reduce false alarms.

Confusion Matrix		Predicted Class		
		0 (predicted value is negative)	**1** (predicted value is positive)	Total Actual (down)
Actual Class	**0** (actual value is negative)	t_n (true negative)	f_p (false positive, false alarm)	Total actual negatives tn + fp
	1 (actual value is positive)	f_n (false negative, false dismissal)	t_p (true positive)	Total actual positives tp + fn
	Total Predicted (across)	Total negative predictions tn + fn	Total positive predictions tp + fp	Total Examples tp + tn + fp + fn

Figure 9-1: Confusion Matrix Components

Table 9-3: Confusion Matrix Measures

CONFUSION MATRIX MEASURES	WHAT IT MEASURES	QUADRANTS USED IN COMPUTATION
$PCC = \frac{t_p + t_n}{t_p + t_n + f_p + f_n}$	Overall accuracy	t_n f_p / f_n t_p
$False\ Alarm\ Rate\ (FA) = \frac{f_p}{t_n + f_p}$	Actual negative cases misclassified as positive	t_n f_p
$False\ Dismissal\ Rate\ (FD) = \frac{f_n}{t_p + f_n}$	Actual positive cases misclassified as negative	f_n t_p
$Precision = \frac{t_p}{t_p + f_p}$	Predicted positive cases classified correctly	f_p / t_p
$Recall = 1 - FD = \frac{t_p}{t_p + f_n}$	Actual positive cases classified correctly	f_n t_p
$Sensitivity = Recall = \frac{t_p}{t_p + f_n}$	Actual positive cases classified correctly	f_n t_p
$Specificity = True\ Negative\ Rate = t\_n / (t\_n + f\_p)$	Actual negative cases classified correctly	t_n f_p

$Type\ I\ Error\ Rate = FA = \frac{f_p}{t_n + f_p}$	Actual negative cases misclassified as positive — Incorrect rejection of a true null hypothesis	
$Type\ II\ Error\ Rate = FD = \frac{f_n}{t_p + f_n}$	Actual positive cases misclassified as negative — Failure to reject a false null hypothesis	

As an example of a confusion matrix, consider a logistic regression model built from the KDD Cup 1998 dataset and assessed on 47,706 records in a test set, with responders, coded as the value 1, comprising 5.1 percent of the population (2,418 of 47,706 records). The model was built from a stratified sample of 50 percent 0s and 50 percent 1s, and the confusion matrix represents the counts in each quadrant based on a predicted probability threshold of the model of 0.5. From these counts, any of the confusion matrix metrics listed in Table 9-3 could be computed, although you would rarely report all of them for any single project. Table 9-4 shows every one of these computed for a sample of 47,706 records. This model has similar false alarm and false dismissal rates (40 percent and 43.3 percent, respectively) because the predicted probability is centered approximately around the value 0.5, typical of models built from stratified samples.

The confusion matrix for this model is shown in Table 9-5. Rates from Table 9-4 are computed from the counts in the confusion matrix. For example, the false alarm rate, 40.0 percent is computed from true negatives and false positives, 18,110 ÷ (27,178 + 18,110).

A second model, built from the natural proportion (not stratified), has a confusion matrix for test data in Table 9-6. Once again, the confusion matrix is created by using a predicted probability threshold of 0.5, although because the model was trained on a population with only 5.06 percent responders, only two of the predictions exceeded 0.5. The second model appears to have higher PCC, but only because the model prediction labels are nearly always 0. Be careful of the prior probability of the target variable values and the threshold that is used in building the confusion matrix. Every software package defaults to equal probabilities as the threshold for building a confusion matrix (0.5 for binary classification), regardless of the prior probability of the target variable.

If you threshold the model probabilities by 0.0506 rather than 0.5, the resulting confusion matrix is nearly identical to the one shown in Table 9-5.

Table 9-4: Comparison Confusion Matrix Metrics for Two Models

METRIC	RATES, MODEL 1	RATES, MODEL 2
PCC	59.8 percent	94.9 percent
FA	40.0 percent	0.0 percent
FD	43.3 percent	100.0 percent
Precision	7.0 percent	0.0 percent
Recall	56.7 percent	0.0 percent
Sensitivity	56.7 percent	0.0 percent
Specificity	60.0 percent	100.0 percent
Type I Rate	40.0 percent	0.0 percent
Type II Rate	43.3 percent	100.0 percent

Table 9-5: Confusion Matrix for Model 1

CONFUSION MATRIX, MODEL 1	0	1	TOTAL
0	27,178	18,110	45,288
1	1,047	1,371	2,418
Total	28,225	19,481	47,706

Table 9-6: Confusion Matrix for Model 2

CONFUSION MATRIX, MODEL 2	0	1	TOTAL
0	45,286	2	45,288
1	2,418	0	2,418
Total	47,704	2	47,706

Beware of comparing individual metrics by themselves. One model may appear to have a low precision in comparison to a second model, but it may only be because the second model has very few records classified as 1, and therefore,

while the false alarm rate is lower, the true positive rate is also much smaller. For example, precision from Table 9-6 is 0 percent because no records were predicted to be responders and were actually value responders. In Table 9-5, precision was equal to 7.0 percent. However, the overall classification accuracy for the model in Table 9-6 is 94.9 percent (45,286 + 0) ÷ 47,706, much larger than the 59.8 percent PCC for Table 9-5.

Confusion Matrices for Multi-Class Classification

If there are more than two levels in the target variable, you can still build a confusion matrix, although the confusion matrix measures listed in Table 9-3 don't apply. Table 9-7 shows a confusion matrix for the seven-level target variable in the nasadata dataset, with the classes re-ordered to show the misclassifications more clearly. The actual values are shown in rows, and the predictive values in the columns. Note that for this classifier there are three groupings of classification decisions: alfalfa and clover form one group; corn, oats, and soy form a second group; and rye and wheat form a third group.

Table 9-7: Multi-Class Classification Confusion Matrix

MULTI-CLASS CONFUSION MATRIX	ALFALFA	CLOVER	CORN	OATS	SOY	RYE	WHEAT
Alfalfa	28	27	0	0	0	0	0
Clover	5	57	0	0	0	0	0
Corn	0	0	50	7	4	0	0
Oats	0	0	4	59	1	0	0
Soy	0	0	3	2	56	0	0
Rye	0	0	0	0	1	53	7
Wheat	0	0	0	0	1	15	44

ROC Curves

Receiver Operating Characteristic (ROC) curves provide a visual trade-off between false positives on the x-axis and true positives on the y-axis. They are essentially, therefore, a visualization of confusion matrices, except that rather than plotting a single confusion matrix, *all* confusion matrices are plotted. Each true positive/false positive pair is found by applying a different threshold on the predicted probability, ranging from 0 to 1. A sample ROC curve is shown in Figure 9-2.

At the extremes, a threshold of 1 means that you only predict 1 for the model if the predicted probability exceeds 1. Because the maximum predicted probability

is 1, the threshold is never exceeded so you never have any false alarms (the false alarm rate is 0). Likewise, you also don't have any actual target values of 1 classified correctly (the sensitivity is also 0). This is the bottom left of the ROC curve. At the other extreme, if you threshold the predictions at 0, every record is predicted to have the value 1, so the sensitivity and false alarm rates are both 1, the upper right of the ROC curve.

The interesting part of the ROC curve is what happens in between these extremes. The higher the sensitivity for low false alarm rates, the steeper the vertical rise you see at the left end of the ROC curve; a perfect ROC curve has a vertical line for sensitivity up to value 1 at the false alarm rate equal to 0, and then is horizontal at the sensitivity value equal to 1 for all values of the false alarm rate. In the sample ROC curve, thresholds between 0.2 and 0.8 at uniform intervals of 0.1 are shown. The distance between the thresholds is not uniform in this ROC curve; the distance depends on the distribution of probabilities.

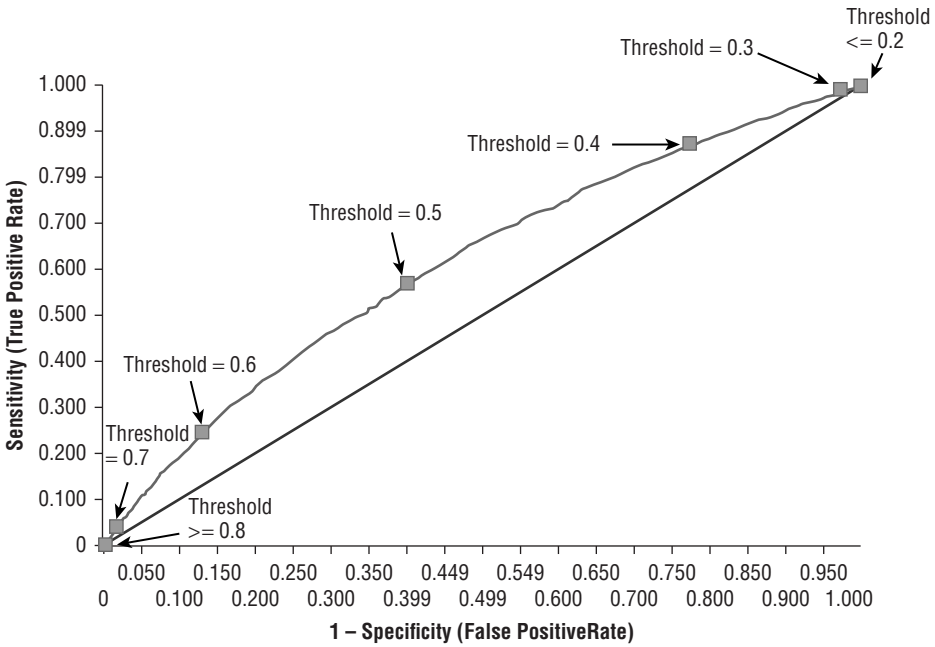


Figure 9-2: Sample ROC curve

One of the most commonly used single-number metrics to compare classification models is Area under the Curve (AUC), usually referring to the area under the ROC curve. For a ROC curve with x- and y-axes containing rates rather than record counts, a perfect model will have AUC equal to 1. A random model will have AUC equal to 0.5 and is represented by the diagonal line between the extremes: coordinates (0,0) and (1,1). A larger AUC value can be achieved by

the ROC curve stretching to the upper left of the ROC curve, meaning that the false alarm rate does not increase as quickly as the sensitivity rate compared to a random selection of records. The AUC for Figure 9-2 is 0.614.

Often, practitioners will display the ROC curves for several models in a single plot, showing visually the differences in how the models trade off false positives and true positives. In the sample shown in Figure 9-3, Model 2 is the best choice if you require false alarm rates to be less than 0.4. However, the sensitivity is still below 0.6. If you can tolerate higher false alarm rates, Model 3 is the better choice because its sensitivity is highest.

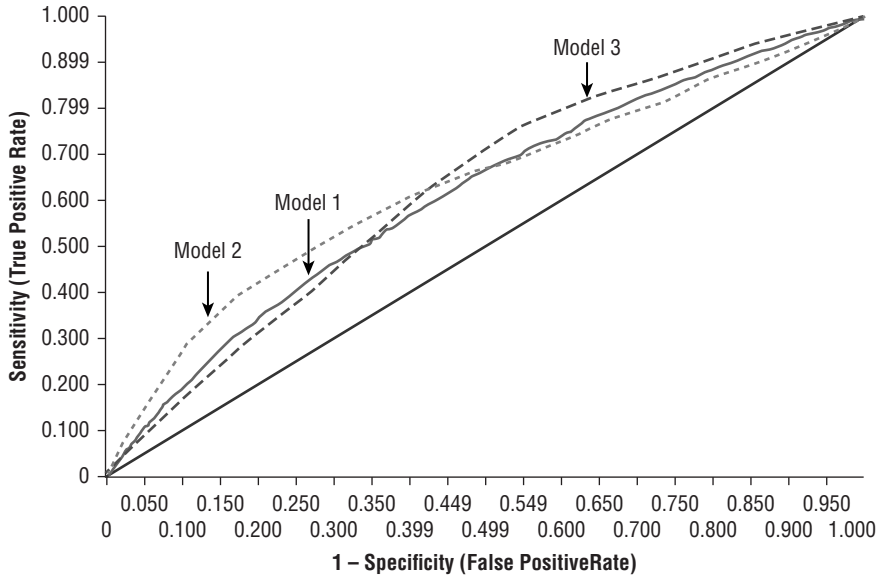


Figure 9-3: Comparison of three models

Rank-Ordered Approach to Model Assessment

In contrast to batch approaches to computing model accuracy, rank-ordered metrics begin by sorting the numeric output of the predictive model, either the probability or confidence of a classification model or the actual predicted output of a regression model. The rank-ordered predictions are binned into segments, and summary statistics related to the accuracy of the model are computed either individually for each segment, or cumulatively as you traverse the sorted file list. The most common segment is the *decile*, with10 percent of the dataset's records in each segment. Other segment sizes used commonly include 5 percent groups, called vinigtiles (also called *vingtiles*, *twentiles*, or *demi-deciles* in software) and 1 percent groups called *percentiles*. Model results in this section are shown with vinigtiles, although you could use any of the segment sizes without changing the principles.

Rank-ordered approaches work particularly well; the model will identify a subset of the scored records to act on. In marketing applications, you treat those who are most likely to respond to the treatment, whether the treatment is a piece of mail, an e-mail, a display ad, or a phone call. For fraud detection, you may want to select the cases that are the most suspicious or non-compliant because there are funds to treat only a small subset of the cases. In these kinds of applications, so long as the model performs well on the selected population, you don't care how well it rank orders the remaining population because you will never do anything with it.

The three most common rank-ordered error metrics are *gains charts*, *lift charts*, and *ROI charts*. In each of these charts, the x-axis is the percent depth of the rank-ordered list of probabilities, and the y-axis is the gain, lift, or ROI produced by the model at that depth.

Gains and Lift Charts

Gain refers to the percentage of the class value of interest found cumulatively in the rank-ordered list at each file depth. Without loss of generality, assume the value of interest is a 1 for a binary classification problem. Figure 9-4 shows a typical gains chart, where the upper curve represents the gain due to the model. This particular dataset has the target variable value of interest represented in 5.1 percent of the data, although this rate is not known from the gains chart itself. The contrasting straight line is the expected gain due to a random draw from the data. For a random draw, you would expect to find 10 percent of the 1s in the first 10 percent of the data, 20 percent in the first 20 percent of the records, and so on: a linear gain. The area between the random line and the model gain is the incremental improvement the model provides. The figure shows through the 10 percent depth, the model selects nearly 20 percent of the 1s, nearly twice as many as a random draw.

The depth you use to measure a model's gain depends on the business objective: It could be gain at the 10 percent depth, 30 percent depth, 70 percent depth, or even the area between the model gain and the random gain over the entire dataset, much like the AUC metric computed from the ROC curve. With a specific value of gain determined by the business objective, the predictive modeler can build several models and select the model with the best gain value.

If a model is perfect, with 5.1 percent of the data with a target value equal to 1, the first model vinigtile will be completely populated with 1s, the remaining 1s falling in the second vinigtile. Because the remainder of the data does not have any more 1s, the remaining values for the gains chart are flat at 100 percent. A perfect gains chart is shown in Figure 9-5. If you ever see a perfect gains chart, you've almost certainly allowed an input variable into the model that is

incorporating target variable information. Find the variable that is most important to the model and correct it or remove it from the list of candidate inputs.

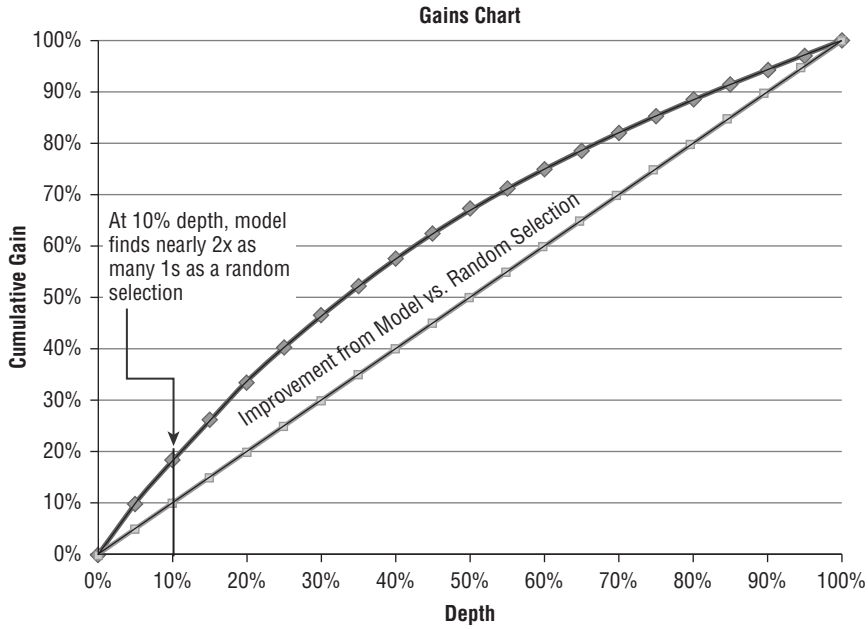


Figure 9-4: Sample gains chart

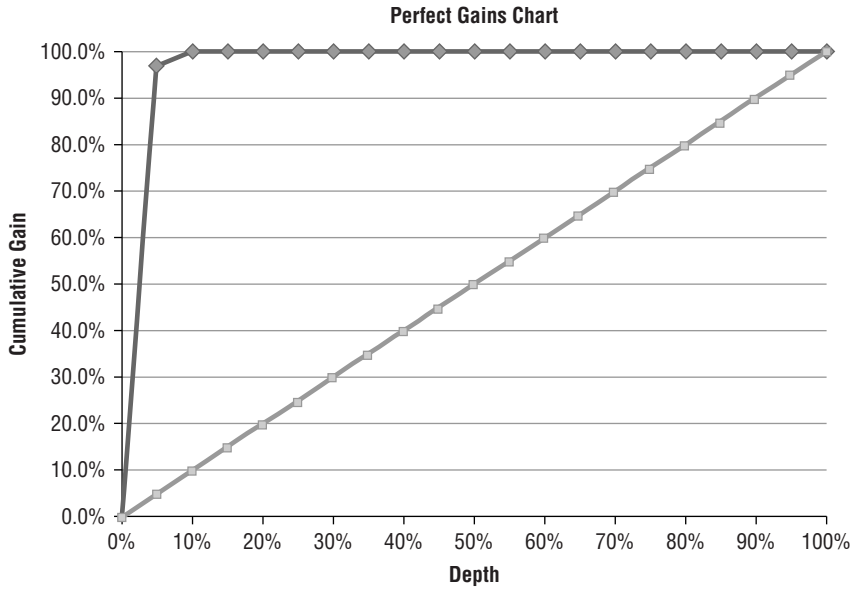


Figure 9-5: Perfect gains chart

Lift is closely related to gain, but rather than computing the percentage of 1s found in the rank ordered list, we compute the *ratio* between the 1s found by the model and the 1s that would have been found with a random selection at the same depth. Figure 9-6 shows a cumulative lift chart for the same data used for the gains chart in Figure 9-4. In a lift chart, the random selection line has a lift of 1.0 through the population. The cumulative lift chart will always converge to 1.0 at the 100 percent depth. Figure 9-7 shows the segment lift chart, the lift for each segment (vinigtiles in this chart). When computing lift per segment, the segment lift must eventually descend below the lift of 1.0, typically halfway through the file depth. One advantage of seeing the segment lift is that you can also see how few 1s are still selected by the model in the bottom segments. For Figure 9-7, the lift is just above 0.5.

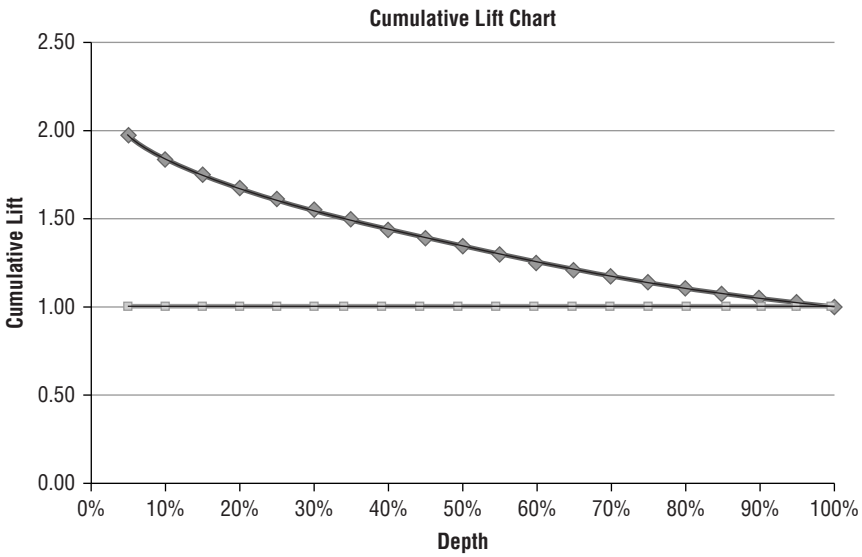


Figure 9-6: Sample cumulative lift chart

Sometimes it is useful to measure model lift from segment to segment rather than cumulatively. Cumulative lift charts aggregate records as the depth increases and can mask problems in the models, particularly at the higher model depths. Consider Figures 9-8 and 9-9, which show the cumulative lift and segment lift for a different model than was used for the lift charts in Figures 9-6 and 9-7. The cumulative lift chart is worse than the prior model but has a smooth appearance from vinigtile to vinigtile. However, in Figure 9-9, the segment lift chart is erratic, with positive and negative slopes from vinigtile to vinigtile. Erratic segment lift

charts created from testing or validation data are indicative of models that are overfit: A stable model will have monotonic lift values from segment to segment.

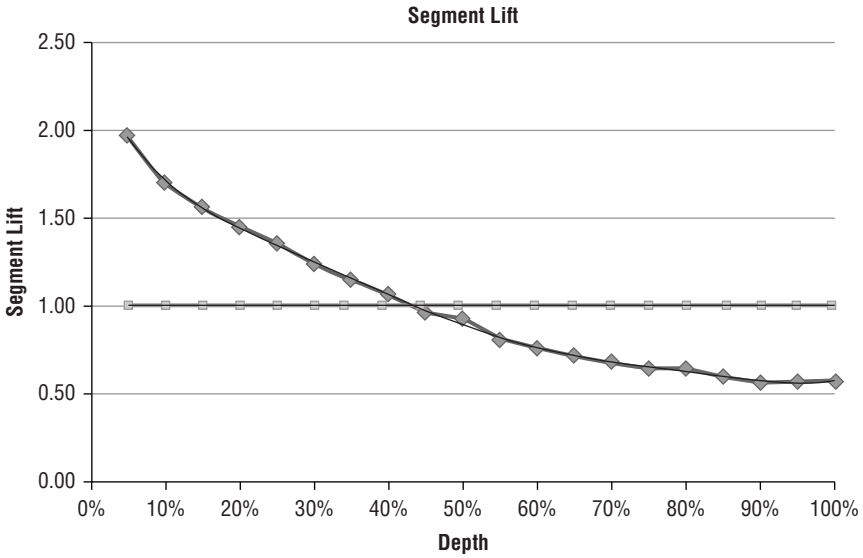


Figure 9-7: Sample segment lift chart

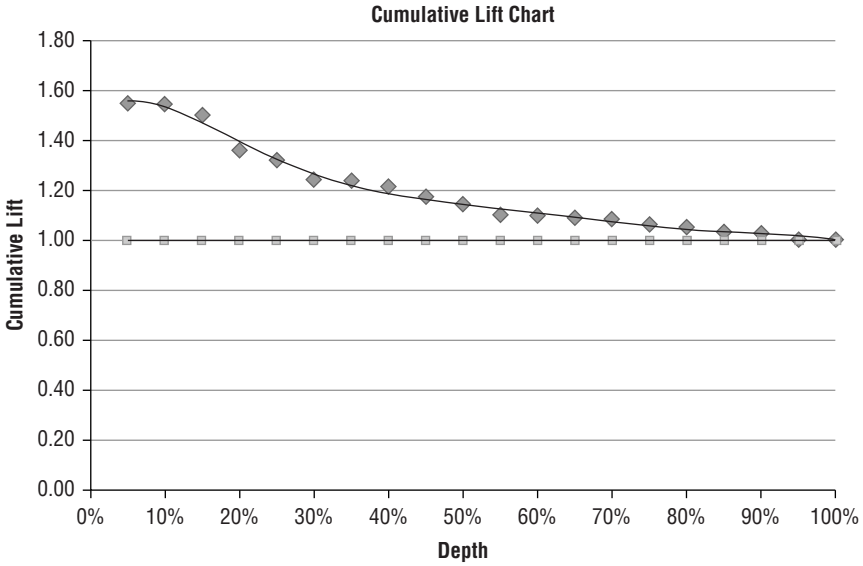


Figure 9-8: Cumulative lift chart for overfit model

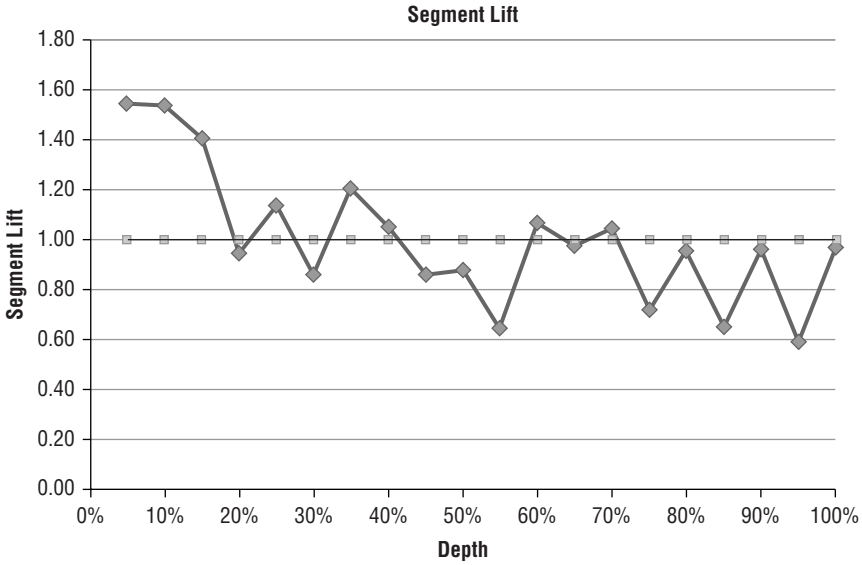


Figure 9-9: Segment lift chart for overfit model

When there are multiple levels in the target variable—multi-class classification problems—gains and lift charts can still be computed, but for only one value of the target variable at a time. For example, with the nasadata, you would build a separate lift chart for each of the seven classes.

Custom Model Assessment

When you use a rank-ordered metric to assess model performance, the actual value predicted by the model no longer matters: Only the metric matters. For gains, the metric is the percentage of 1s found by the model. For lift, the metric is the ratio of the percentage of 1s found to the average rate. For ROC, it is the comparison of true alerts to false alarms. In each of these cases, each record has equal weight. However, some projects can benefit from weighting records according to their costs or benefits, creating a custom assessment formula.

For example, one popular variation on rank-ordered methods is computing the expected cumulative profit: a fixed or variable gain minus a fixed or variable cost. Fixed cost and gain is a straightforward refinement of a gains chart, and many software tools include profit charts already. For example, the variable gain in a marketing campaign can be the purchase amount of a new customer's first visit, and the variable cost the search keyword ad cost. Or for a fraud detection

problem, the variable gain can be the amount of fraud perpetrated by the offender and the variable cost the investigation costs.

Consider an application like the KDD Cup 1998 data that has fixed cost (the cost of the mailing) and variable gain (the donation amount). Figure 9-10 shows the profit chart for this scenario. This particular model generates a maximum profit of nearly $2,500 at the depth of 25,000 donors. This maximum profit is nearly two times as much compared to contacting all the donors (depth of 43,000).

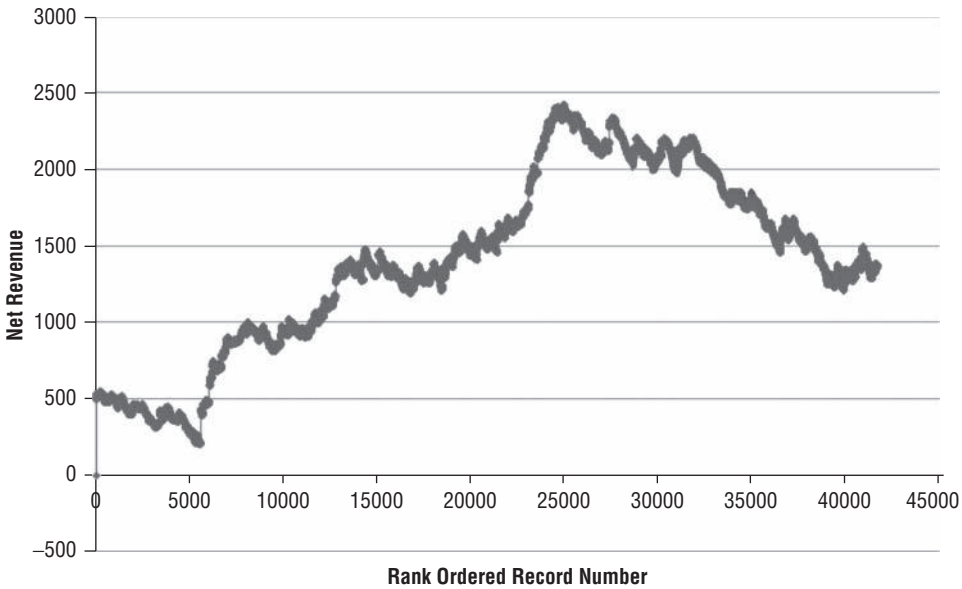


Figure 9-10: Profit chart

Custom model assessments can be applied to regression models just as easily as classification models precisely because it is only the rank-ordering that matters, not the actual prediction value. By extension, this also means that completely different models, built for different target variables, can be assessed and compared directly, on a common scale.

In addition, custom model assessments can also be applied to batch metrics, adding to the list of confusion matrix metrics already listed in Table 9-3. For example, consider a business objective that stipulates false alarms have four times the cost of false dismissals. Table 9-8 shows the same data already shown in Table 9-2 with the additional column showing the cost of an error. Correct classifications receive a cost of 0, false dismissals a cost of 1, and false alarms a cost of 4. The sum of these costs is the model score and the best model is the lowest-scoring model for the cost function.

Table 9-8: Custom Cost Function for False Alarms and False Dismissals

ACTUAL TARGET VALUE	PROBABILITY TARGET = 1	PREDICTED TARGET VALUE	CONFUSION MATRIX QUADRANT	COST OF ERROR, FALSE ALARM 4X
0	0.641	1	false alarm	4
1	0.601	1	true positive	0
0	0.587	1	false alarm	4
1	0.585	1	true positive	0
1	0.575	1	true positive	0
0	0.562	1	false alarm	4
0	0.531	1	false alarm	4
1	0.504	1	true positive	0
0	0.489	0	true negative	0
1	0.488	0	false dismissal	1
0	0.483	0	true negative	0
0	0.471	0	true negative	0
0	0.457	0	true negative	0
1	0.418	0	false dismissal	1
0	0.394	0	true negative	0
0	0.384	0	true negative	0
0	0.372	0	true negative	0
0	0.371	0	true negative	0
0	0.341	0	true negative	0
1	0.317	0	false dismissal	1

Which Assessment Should Be Used?

In general, the assessment used should be the one that most closely matches the business objectives defined at the beginning of the project during Business Understanding. If that objective indicates that the model will be used to select one-third of the population for treatment, then model gain or lift at the 33 percent depth is appropriate. If all customers will be treated, then computing AUC for a batch metric may be appropriate. If the objective is to maximize the records selected by the model subject to a maximum false alarm rate, a ROC is appropriate.

The metric used for model selection is of critical importance because the model selected based on one metric may not be a good model for a different metric.

Consider Figure 9-11: a scatterplot of 200 models and their rank based on AUC at the 70 percent depth and the root mean squared (RMS) error. The correlation between these two rankings is 0.1—almost no relationship between the two rankings; the ranking based on AUC cannot be determined at all from the ranking of RMS error. Therefore, if you want to use a model operationally in an environment where minimizing false positives and maximizing true positives is important, choosing the model with the best RMS error model would be sub-optimal.

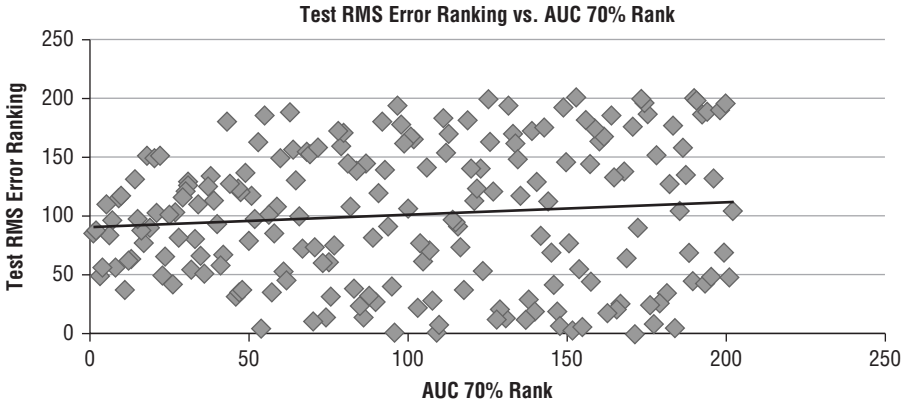


Figure 9-11: Scatterplot of AUC vs. RMS Error

Assessing Regression Models

The metrics most predictive analytics software packages provide to assess regression models are batch methods computed for the data partition you are using to assess the models (training, testing, validation, and so on). The most commonly used metric is the coefficient of determination known as R^2, pronounced "r squared." R^2 measures the percentage of the variance of the target variable that is explained by the models. You can compute it by subtracting the ratio of the variance of the residuals and the variance of the target variable from 1. If the variance of the residuals is 0, meaning the model fit is perfect, R^2 is equal to 1, indicating a perfect fit. If the model explains none of the variance in the model, the variance of the residuals will be just as large as the variance of the target variable, and R^2 will be equal to 0. You see R^2 even in an Excel trend line equation.

What value of R^2 is good? This depends on the application. In social science problems, an R^2 of 0.3 might be excellent, whereas in scientific applications you might need an R^2 value of 0.7 or higher for the model to be considered a good fit. Some software packages also include a modification of R^2 that penalizes

the model as more terms are added to the model, much like what is done with AIC, BIC, and MDL.

Figure 9-12 shows two regression models, the first with a relatively high R^2 value and the second with a nearly zero R^2 value. In the plot at the left, there is a clear trend between LASTGIFT and TARGET_D. At the right, the relationship between the variables is nearly random; there is very little explained variance from the linear model.

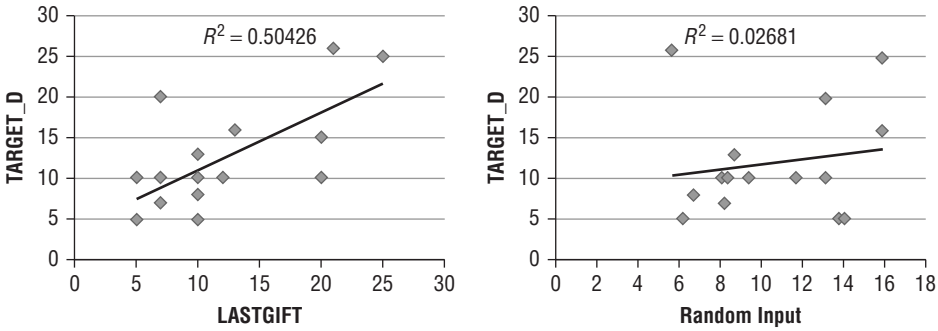


Figure 9-12: R^2 for two linear models

R^2 is only one of the many ways regression models are assessed. Table 9-9 shows a list of other common measures of model accuracy. Even if these are not included in your predictive modeling software, they are easy to compute.

Table 9-9: Batch Metrics for Assessing Regression Models

MEASURE	EXPLANATION OF MEASURE
R^2	Percent variance explained
Mean squared error (MSE)	Average error. If the model predictions are unbiased positively or negatively, this should be equal to 0.
Mean absolute error (MAE)	Compute the absolute value of the error before averaging. This provides an average magnitude of error measure and is usually preferred over MSE for comparing models.
Median error	A more robust measure of errors than MSE because outliers in error don't influence the median
Median absolute error	A more robust measure of error magnitudes
Correlation	The square root of R^2, this is an intuitive way to assess how similar model predictions are to the target values. Correlations are sensitive to outliers.

MEASURE	EXPLANATION OF MEASURE
Average percent error	Compute the percentage of the error rather than the mean. This normalizes by the magnitude of the target variable, showing the relative size of the error compared to the actual target value.
Average absolute percent error	Compute the percentage of the absolute value of the error rather than the mean. As with average percent error, this shows the relative size of the error.

The error metric you use depends on your business objective. If you care about total errors, and larger errors are more important than smaller errors, mean absolute error is a good metric. If the relative size of the error is more important than the magnitude of the error, the percent error metrics are more appropriate.

Table 9-10 shows several of the error metrics for the KDD Cup 1998 data using four regression models, each predicting the target variable TARGET_D. The linear regression model has the largest R^2 in this set of models. Interestingly, even though the linear regression model had the highest R^2 and lowest mean absolute error, the regression trees had lower median absolute error values, meaning that the most typical error magnitude is smaller for the trees.

Table 9-10: Regression Error Metrics for Four Models

MEASURE	LINEAR REGRESSION	NEURAL NETWORK	CART REGRESSION TREE	CHAID REGRESSION TREE
R^2	0.519	0.494	0.503	0.455
Mean Error	–0.072	–0.220	–0.027	0.007
Mean Absolute Error	4.182	4.572	4.266	4.388
Median Absolute Error	2.374	3.115	2.249	2.276
Correlation	0.720	0.703	0.709	0.674

Rank-ordered model assessment methods apply to regression problems in the same way they apply to classification problems, even though they often are not included in predictive analytics software. Rank-ordered methods change the focus of model assessment from a record-level error calculation to the ranking, where the accuracy itself matters less than getting the predictions in the right order.

For example, if you are building customer lifetime value (CLV) models, you may determine that the actual CLV value is not going to be particularly accurate. However, identifying the customers with the most potential, the top 10 percent of the CLV scores, may be strategically valuable for the company. Figure 9-11 showed that batch assessment measures might not be correlated with rank-ordered assessment methods for classification. The same applies to regression.

For example, Table 9-11 shows a decile-by-decile summary of two models built on the KDD Cup 1998 data. The top decile for the linear regression model finds donors who gave on average 32.6 dollars. The average for donors who gave a donation was 15.4, so the top decile of linear regression scores found gift amounts with a lift of 2.1 (32.6 ÷ 15.4).

Table 9-11: Rank-Ordering Regression Models by Decile

DECILE	LINEAR REGRESSION, MEAN TARGET_D	NEURAL NETWORK, MEAN TARGET_D
1	32.6	32.3
2	22.1	21.4
3	18.9	19.1
4	17.1	16.8
5	15.3	15.0
6	13.2	13.1
7	11.3	11.2
8	10.5	10.2
9	7.6	7.8
10	6.0	6.6

The deciles are shown in tabular form, but they can also be shown in graphical form, as in the chart in Figure 9-13. The figure shows that the model predicts the highest donors in the top decile particularly well; the average TARGET_D values for the top decile is larger than you expect based on the decreasing trend of the remaining deciles.

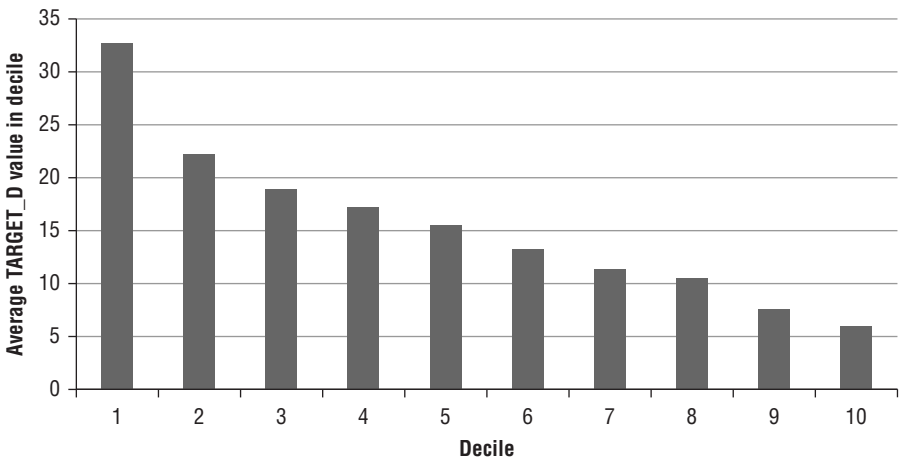


Figure 9-13: Average actual target value by decile

Summary

A predictive model is only good if it answers a business question effectively. Predictive modelers determine the meaning of the words "good" and "effectively" during Business Understanding by translating the business objectives into modeling objectives. When these objectives are defined clearly, identifying the best model is easy: The best model is the one that optimizes that objective.

The objectives sometimes require metrics that consider all the records, what I called batch objectives. PCC and the confusion matrix metrics are the most commonly used batch objectives for classification. Even here, however, there are differences between the ways a confusion matrix can be used to assess models, depending on how you trade off false alerts, false dismissals, true alerts, and true dismissals. For regression problems, the most commonly used metrics are R^2 and average squared error.

Very often, however, the models are built to select a sub-population to treat. In these problems, a rank-ordered metric is more appropriate. For classification models, ROC curves, gains charts, lift charts, and profit charts are the most popular methods to assess these problems. In addition to these methods, some business objectives lead modelers to create customized model assessment metrics. Similar rank-ordered metrics are appropriate for regression problems.

Regardless of the method, you should match the metric to the business objective or else you may pick a model that won't be the best operationally.

CHAPTER

10

Model Ensembles

Model Ensembles, or simply "ensembles," are combinations of two or more predictions from predictive models into a single composite score. The chapters on building supervised and unsupervised learning models focused on how to build the best single model and how to determine which of the single models you should select for deployment. The ensemble approach turns this thinking around. Rather than building models and selecting the single best model to deploy, why not build many models and use them all in deployment?

This chapter describes how to build model ensembles and why understanding how to build them is important—some would say essential—for predictive modelers.

Motivation for Ensembles

Practitioners build model ensembles for one reason: improved accuracy. In study after study over the past two decades, ensembles nearly always improve model predictive accuracy and rarely predict worse than single models. In the 1990s, when ensembles were beginning to appear in the data-mining literature,

the improved accuracy of ensembles on held-out data began to appear in an almost magical way. By the 2000s, ensembles became essential to winning data mining competitions.

Consider two examples, and these are not unique. First, the Pacific-Asia Conference on Knowledge Discovery (PAKDD) has a data mining competition each year called the PAKDD Cup. In 2007, twenty-eight submissions were received and the top five solutions were model ensembles, including three submissions with ensembles of trees, one with an ensemble of Probit models (similar to logistic regression models), and one neural network ensemble.

A second recent example of ensembles winning competitions is the Netflix prize, an open competition that solicited entries to predict user ratings of films based on historical ratings. The prize was $1,000,000 for any team that could reduce the root means squared error (RMSE) of the existing Netflix internal algorithm by 10 percent or more. The winner, runner-up, and nearly all the leaders used model ensembles in their submissions. In fact, the winning submission was the result of an ensemble containing hundreds of predictive models.

Model ensembles not only can improve model accuracy, but they can also improve model robustness. Through averaging multiple models into a single prediction, no single model dominates the final predicted value of the models, reducing the likelihood that a flaky prediction will be made (so long as all the models don't agree on the flaky prediction).

The improved accuracy comes with costs, however. First, model interpretation, if it ever existed, is lost. A single decision tree is a favorite choice of predictive modelers when model interpretation is paramount. However, if an ensemble of 1,000 trees is the final model, the determination of why a prediction was made requires assessing the predictions of 1,000 models. Techniques exist for aiding in interpreting the ensemble but the transparency of a single decision tree is gone.

Second, model ensembles can become too computationally expensive for deployment. In the Netflix prize, the winning submission was never used directly to make movie recommendations because of its complexity and inability to scale and make the movie predictions quickly enough. Sometimes accuracy itself isn't enough for a model or ensemble of models to be useful.

The Wisdom of Crowds

In the popular book *The Wisdom of Crowds* (Random House, 2005), author James Surowiecki proposes that better decisions can be made if rather than relying on a single expert, many (even uninformed) opinions can be aggregated into a decision that is superior to the expert's opinion, often called *crowdsourcing*.

Surowiecki describes four characteristics necessary for the group opinion to work well and not degenerate into the opposite effect of poor decisions as

evidenced by the "madness of crowds": diversity of opinion, independence, decentralization, and aggregation (see Figure 10-1). The first three characteristics relate to how the individual decisions are made: They must have information different than others in the group and not be affected by the others in the group. The last characteristic merely states that the decisions must be combined.

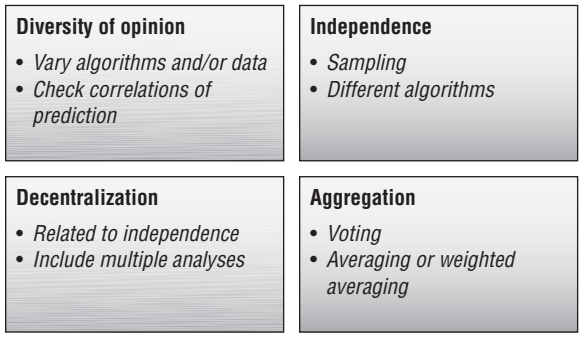


Figure 10-1: Characteristics of good ensemble decisions

Model ensembles have very similar characteristics. Each predictive model has a voice in the final decision. The diversity of opinion can be measured by the correlation of the predictive values themselves: If all of the predictions are highly correlated with one another, or in other words, if the models nearly all agree, there is no point in combining them. The decentralization characteristic can be achieved by resampling data or through case weights: Each model uses either different records from a common data set or at least uses the records with weights that are different from the other models.

Bias Variance Tradeoff

Before describing model ensembles, it is useful to understand the principle in statistical literature of the *bias-variance tradeoff*. Bias refers to model error and variance refers to the consistency in predictive accuracy of models applied to other data sets. The best models have low bias (low error, high accuracy) *and* low variance (consistency of accuracy from data set to data set).

Unfortunately, there is always a tradeoff between these two building predictive models. You can achieve low bias on training data, but may suffer from high variance on held-out data because the models were overfit. The k-NN algorithm with k=1 is an example of a low bias model (perfect on training data), but susceptible to high variance on held-out data. Conversely, you can achieve low variance from data set to data set but at the cost of higher bias. These are typically simple models that lack the flexibility and power to predict accurately, such as single split trees called decision stumps.

In a simple example, Figure 10-2 shows a data set with a simple linear regression model.

This model will have relatively low variance as it is a smooth predictor, although it has bias. A second curve fit is shown in Figure 10-3, this time with low bias, but because of the complexity of the model it is in danger of overfitting the data and will likely have large errors on subsequent data. These larger errors are shown in Figure 10-4. Four vertical, dashed lines show the errors of these four data points, clearly large errors induced by the differences between the diamond data points the model was built from and the second box data. Every new data set, with comparable variations in the data, will also have large errors like those shown in Figure 10-4, resulting in high error variance from this model.

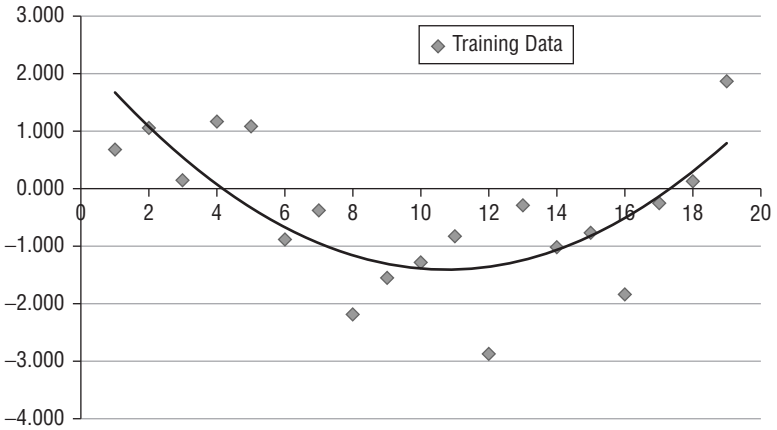


Figure 10-2: Low variance fit to data

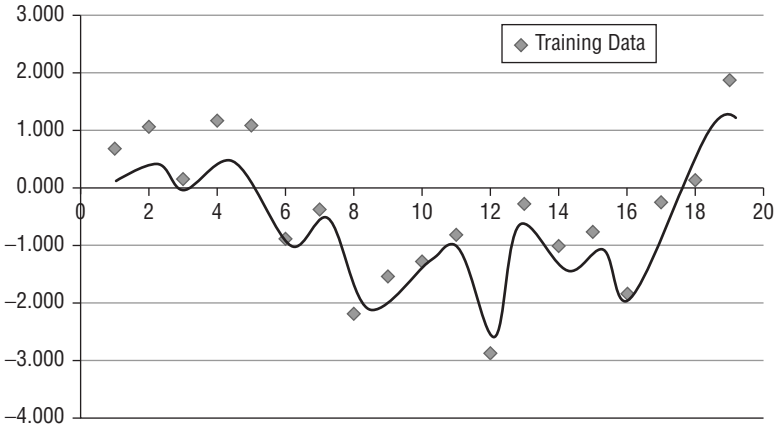


Figure 10-3: Low bias fit to data

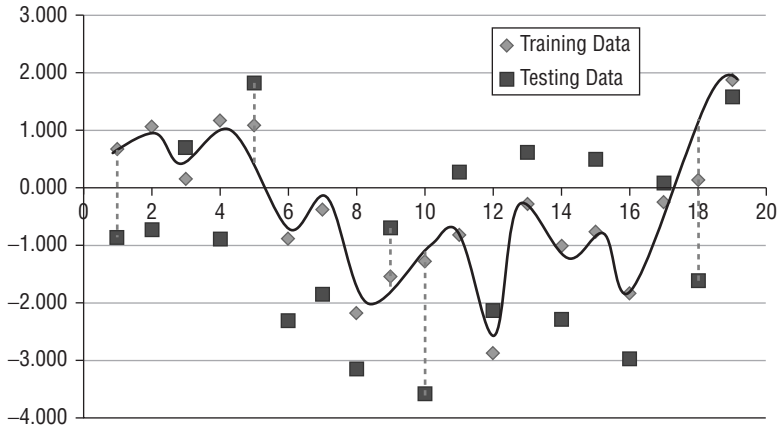


Figure 10-4: Errors on new data

Bagging

Leo Brieman first published a description of the Bootstrap Aggregating (bagging) algorithm in 1996. The idea is quite simple yet powerful: Build multiple decision trees from resampled data and combine the predicted values through averaging or voting. The resampling method Breiman used was bootstrap sampling (sampling with replacement), which creates replicates of some records in the training data, although on average, 37 percent of the records will not be included at all in the training data.

Although bagging was first developed for decision trees, the idea can be applied to any algorithm that produces predictions with sufficient variation in the predicted values. Other algorithms that are good candidates include neural networks, Naïve Bayes, k-nearest neighbor (for low values of k), and to a lesser degree, even logistic regression. k-nearest neighbor is not a good candidate for bagging if the value of k is already large; the algorithm already votes or averages predictions and with larger values of k, predictions are already very stable with low variance.

How many bootstrap samples, sometimes called replicates, should be created? Brieman stated, "My sense of it is that fewer are required when y is numerical and more are required with an increasing number of classes." He typically used 10–25 bootstrap replicates, with significant improvements occurring with as few as 10 replicates.

Overfitting the models is an important requirement to building good bagged ensembles. By overfitting each model, the bias is low, but the decision tree will

generally have worse accuracy on held-out data. But bagging is a variance reduction technique; the averaging of predictions smoothes the predictions to behave in a more stable way on new data.

For example, consider models based on the KDD Cup 1998 data set. Thirty (30) bootstrap samples were created from a stratified sample of the target variable. From these 30 samples, 30 decision trees were built. Only LASTGIFT and RFA_2F were included as inputs to simplify visualization of results. Each model was a CART-styled decision tree that was somewhat overfit, per Breiman's recommendation.

Results shown are based on test data only, not training data or out-of-sample data from the bootstrap samples. Table 10-1 shows the average model Area under the Curve (AUC) for these 30 models. The 30 models were remarkably consistent in their AUC metrics; the AUC values ranged from 0.530 to 0.555. If the models built to be included in the ensemble are poor predictors, or in other words have high bias, it is better to exclude those models from the ensemble as they will degrade the accuracy of the ensemble.

Bagged ensembles were created from averaging the model predicted probabilities rather than voting. A 10-model ensemble is shown in the table as well: Its AUC is better than even the best individual tree. Finally, a 30-model ensemble was created and had the best AUC on test data. Bagged ensembles with more than 30 trees were not any better than using 30 trees. While the gains are modest on this data set, they are statistically significant. These results are summarized in Table 10-1.

Table 10-1: Bagging Model AUC

METHOD	AUC
Average tree	0.542
Best tree	0.555
10-tree ensemble	0.561
30-tree bagging ensemble	0.566

The diversity of model predictions is a key element in creating effective ensembles. One way to measure the diversity of predictions is to examine the correlation of predicted values. If the correlations between model predictions are always very high, greater than 0.95, there is little additional predictive information each model brings to the ensemble and therefore little improvement in accuracy is achievable. Generally, it is best to have correlations less than 0.9 at most. The correlations should be computed from the model propensities or predicted probabilities rather than the {0,1} classification value itself.

In the modeling example, the 30 models produce pairwise correlations that range from 0.212 to 0.431: all quite low. When estimates produce similar performance scores (such as the AUC here) but are not correlated with each other, they achieve their accuracy by predicting accurately on different records, the ideal scenario for effective ensembles. Records that are easy to classify have no need for ensembles; any single model will classify these records correctly. Likewise, accuracy on records that are never classified correctly in any model cannot be improved; these records are lost causes. It is the disagreement between models that provides the potential. The hope is that the majority of models in the ensemble will predict these correctly and thus "outvote" the poor decisions.

A final way to examine how the Bagging Ensemble turns individual models into a composite prediction can be seen through the decision boundaries created by the models. Figure 10-5 shows decision regions for nine of the models. The x-axis in each plot is LASTGIFT; the y-axis is RFA_2F. RFA_2F values (1, 2, 3, or 4) have been randomly jittered so they can be seen more clearly. A predicted class "1" is shown as a light gray square and a predicted class "0" by a dark gray square. It is easy to see from these plots why the correlations between models is so low: each model creates different decision regions.

After averaging 30 models, the resulting decision regions that produced the increase in AUC are shown in Figure 10-6. The regions are generally more homogeneous than in single models.

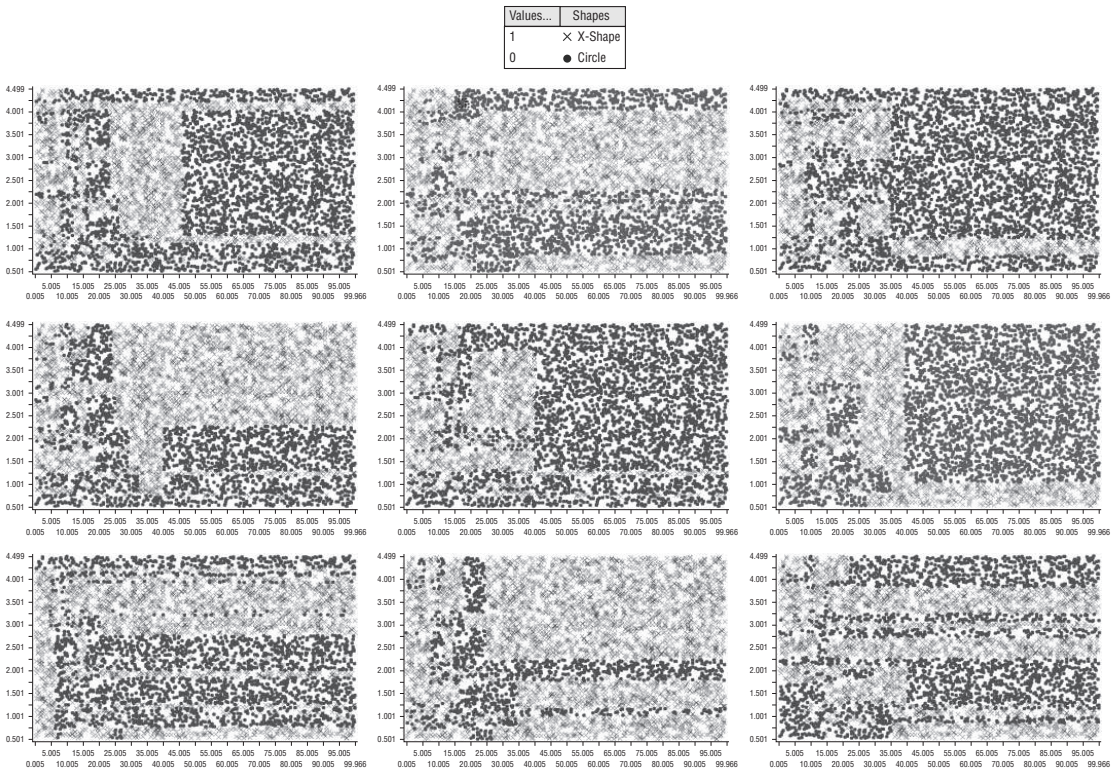


Figure 10-5: Decision regions for nine bagged trees

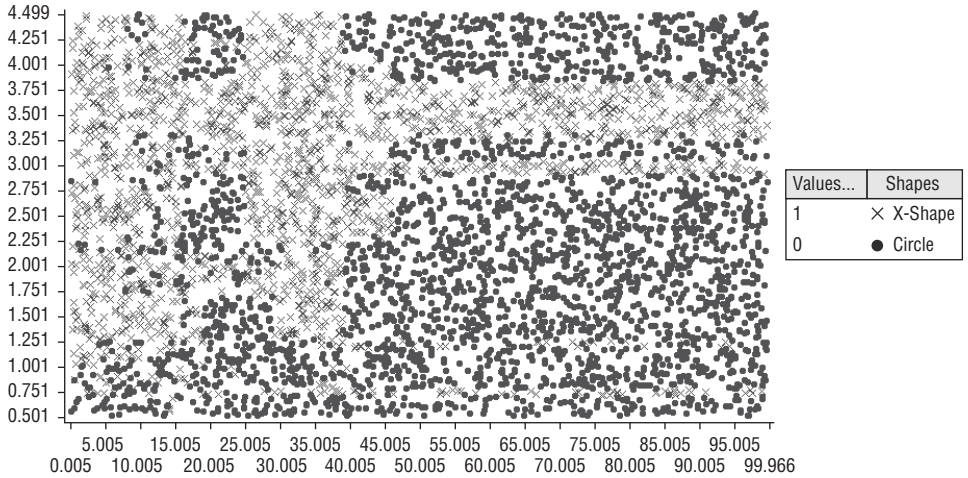


Figure 10-6: Decision region for bagging ensemble

Contrast these figures to a heat map of the actual response rates plotted on the same axes as shown in Figure 10-7. Since the target variable, TARGET_B, only has values 0 and 1, continuous values comparable to the model probabilities are created by binning the two variables (LASTGIFT and RFA_2F) and computing the average response rate in each bin. Each square was colored using colors comparable to the points in the scatterplots shown in Figures 10-5 and 10-6. The bagging ensemble clearly has identified the trends in the data (Figure 10-6), especially the light gray region at the upper left and the dark gray region at the bottom right. The vertical stripe in Figure 10-6 can be seen in the vertical band for LASTGIFT having values between 7 and 10 dollars.

Bootstrap sampling in bagging is the key to introducing diversity in the models. One can think of the bootstrap sampling methodology as creating case weights for each record: Some records are included multiple times in the training data—their weights are 1, 2, 3, or more—and others are not included at all—their weights are equal to 0.

Some software implementations of decision trees will include an option for bagging or provide the capability to construct bagged models. Even if this is not the case, however, bagging is straightforward to implement in most predictive analytics software packages. For example, if a modeler is building logistic regression models, creating bagged ensembles of logistic regression models requires only the creation of 10 samples, 10 models, and logic to combine the predictions. The same is true for creating bagged ensembles using any algorithm.

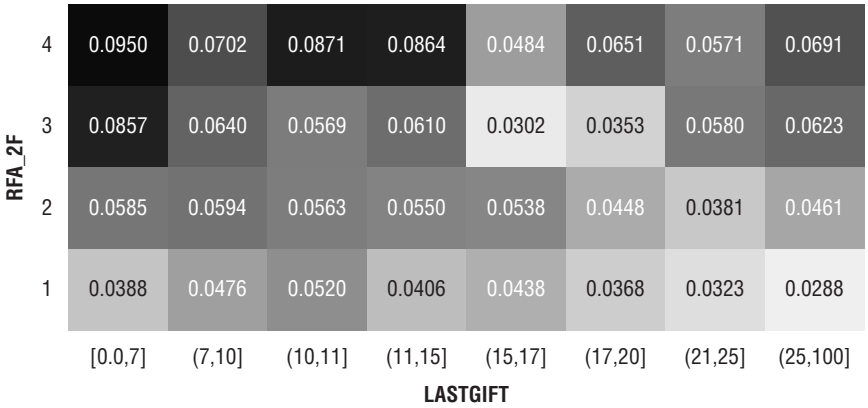


Figure 10-7: Decision region for actual target variable

Other techniques also exist for varying case weights rather than using a bootstrap sample. One could select randomly which records to include in training and which to exclude, which assigns case weights of 1 and 0 to each record respectively. Cross-validation sampling is one way to accomplish this weighting. One could even create a random, real-valued case weight for each record. These are not as commonly found in commercial software but are easy to implement through custom manipulation of the data.

Boosting

Robert Shapire and Yoav Freund introduced the boosting algorithm in separate publications in 1990. In a 1995 joint publication they first described the AdaBoost algorithm (pronounced "add-a-boost"). The idea, like bagging, is also straightforward.

First, one builds a simple classification model; the model only needs to be slightly better than random chance, so for a binary classification problem, only slightly better than a 50 percent correct classification. In this first pass, each record is used in the algorithm with equal case weights, as one would do normally in building a predictive model. The errors in the predicted values are noted. Records correctly classified have their case weights reduced and records incorrectly classified have their case weights increased, and a second simple model is built. In other words, for the second model, records that were incorrectly classified are encouraged through case weights to be considered more strongly in the construction of the model. The records that are difficult to classify, meaning that initially, the models classify them incorrectly, keep getting case

weights increased more and more, communicating to the algorithm to pay more attention to these records until, hopefully, they are finally classified correctly.

Boosting is often repeated tens or even hundreds of times. After the tens or hundreds of iterations, the final predictions are made based on a weighted average of the predictions from all the models.

Consider the simple example in Figure 10-8 based on only 10 data points. The data contains two class values represented by a dark gray circle and a light gray square. The size of the symbol represents the case weight for that data point, with the initial size representing a weight of 1. In the first decision stump, cases to the left of the split, represented by the vertical dotted line, are predicted to be light gray, and those to the right are predicted to be dark gray. Therefore, there are three misclassified data points (all light gray), and the remaining data points, including all the dark gray points, are classified correctly. These misclassified data points are given increased weight and the correctly classified data points decreased weight, as shown in the upper right of the figure and labeled "Reweighting." The amount of the reweighting shown in Figure 10-8 only represents the increase or decrease and should not be interpreted as showing a precise reweighting.

The second decision stump is then computed based on the reweighted data, and the process of reweighting the data points is repeated, resulting in the plot to the right of the second stump split. After the third reweighting, you can see that the light gray data point at the upper right has now been misclassified three times and therefore has the largest weight of any data point. Two data points have been classified correctly three times, and their weights are the smallest; they are at the top left (light gray) and bottom right dark gray.

If you examine all three splits from the three decision stumps, you can see in Figure 10-9 that there are six decision regions defined by the three splits. Each region will have a different model score based on the weighted sum of the three models, with the region at the upper left having a probability of being light gray equal to 1 and the region to the lower right having a probability of being light gray equal to 0. If additional iterations of the boosting algorithm were to take place, the high weight of the light gray square at the upper right would undoubtedly result in a decision stump that separates this data point from the others in the data to classify it correctly.

As a reminder, this process is done automatically in software implementations of boosting and does not have to be done by the modeler in a step-by-step manner. For example, the C5 classification tree algorithm has a boosting algorithm built in, using a variant of the original AdaBoost algorithm.

Boosting methods are designed to work with weak learners, that is, simple models; the component models in a boosted ensemble are simple models with high bias, though low variance. The improvement with boosting is better, as

with Bagged models, when algorithms that are unstable predictors are used. Decision trees are most often used in boosted models. Naïve Bayes is also used but with fewer improvements over a single model.

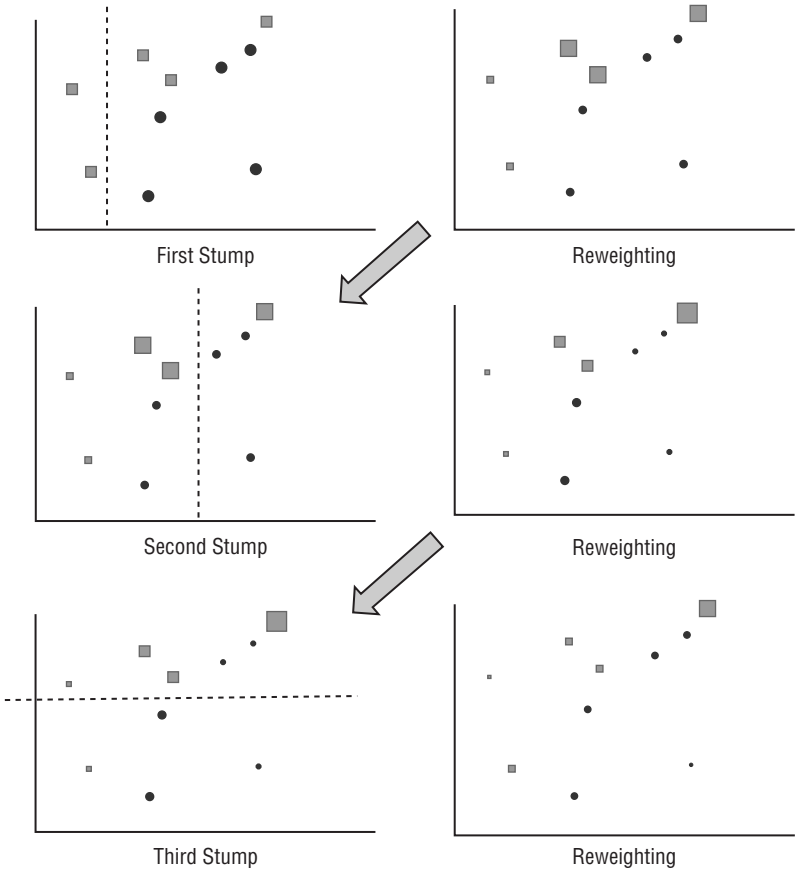


Figure 10-8: AdaBoost reweighting

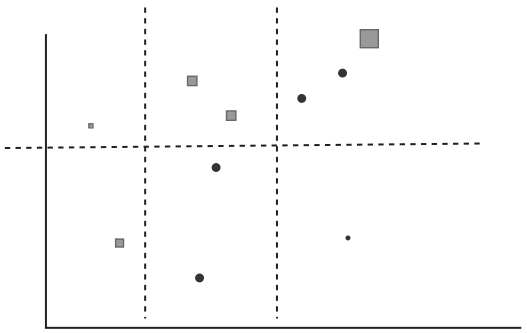


Figure 10-9: AdaBoost decision regions for a simple example

Model accuracy from boosting typically is better than single decision trees or even from bagging. Figure 10-10 shows the AUC for 30 individual trees, averaging predictions from 3 groups of 10 trees (non-overlapping), bagging, and AdaBoost built from 100 iterations. All results are from test data. In each tree, only LASTGIFT and RFA_2F were used as inputs. Clearly, AdaBoost has the highest accuracy; this was true even with only 10 iterations. The box and whiskers plot for the single models shows the minimum and maximum AUC values. Note that bagging has significantly higher AUC than even the best single model, and AdaBoost significantly higher than bagging.

Boosted trees created from the same KDD Cup 1998 data as the results shown in Figures 10-6 have far simpler decision regions than individual trees and the bagged trees, even after 100 iterations. Figure 10-11 shows the decision regions for the AdaBoost predictions. The regions from AdaBoost have smoother transitions between the decisions and the regions themselves are more homogeneous.

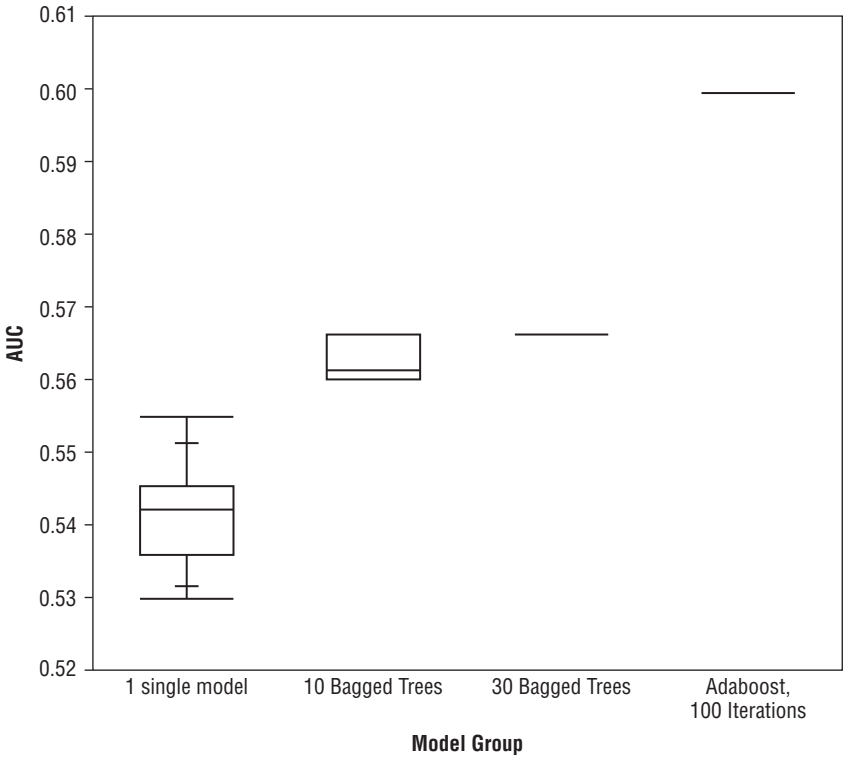


Figure 10-10: Comparison of AUC for individual trees and ensembles

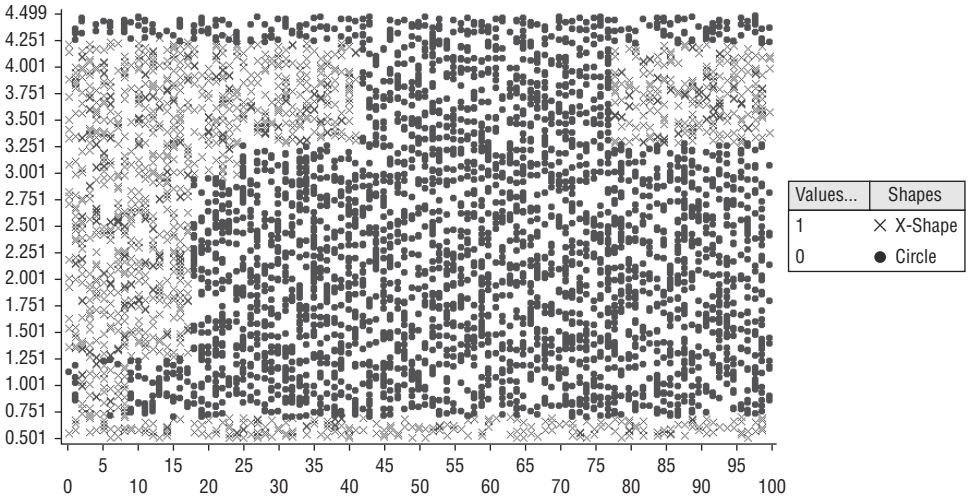


Figure 10-11: Decision regions for the AdaBoost ensemble

Improvements to Bagging and Boosting

Bagging and boosting were the first ensemble methods that appeared in predictive analytics software, primarily with decision tree algorithms. Since their introduction, many other approaches to building ensembles have been developed and made available particularly in open source software. The three most popular and successful approaches are random forests, stochastic gradient boosting, and heterogeneous ensembles.

Random Forests

Random Forests (RF) was introduced by Breiman in 2000 as a modification to the bagging algorithm. As with bagging, the algorithm begins with bootstrap sampled data, and one decision tree is built from each bootstrap sample. There is, however, an important twist to the RF algorithm: At each split in the tree, rather than considering all input variables as candidates, only a random subset of variables is considered.

The number of variables one considers is a parameter in the algorithm specification, but the default is to use the square root of the total number of candidate inputs. For example, if there were 100 candidate inputs for the model, a random 10 inputs are candidates for each split. This also means that it is unlikely that the same inputs will be available for splits at parent and children nodes in the

tree, forcing the tree to find alternate ways to maximize the accuracy of subsequent splits in the tree. Therefore, there is a two-fold diversity in the trees. First, diversity is encouraged by case weights: Some patterns are over-emphasized and others under-emphasized in each tree. Additionally, diversity is encouraged through input weighting ("on" vs. "off").

RF models are typically more accurate than bagging and are often more accurate than AdaBoost.

Stochastic Gradient Boosting

AdaBoost is only one of many boosting algorithms currently documented in the literature, though it and the C5 boosting algorithm are most common in commercial predictive modeling software; dozens of boosting algorithms can be found in the open source software packages.

One interesting boosting algorithm is the Stochastic Gradient Boosting (SGB) algorithm created by Jerry Friedman of Stanford University. Friedman developed an algorithm published in 2001 that he called Multiple Additive Regression Trees (MART) and was later branded as TreeNet® by Salford Systems in its software tool. Like other boosting algorithms, the MART algorithm builds successive, simple trees and combines them additively. Typically, the simple trees are more than stumps and will contain up to six terminal nodes.

After building the first tree, *errors* (also called *residuals*) are computed. The second tree and all subsequent trees then use residuals as the target variable. Subsequent trees identify patterns that relate inputs to small and large errors. Poor prediction of the errors results in large errors to predict in the next tree, and good predictions of the errors result in small errors to predict in the next tree. Typically, hundreds of trees are built and the final predictions are an additive combination of the predictions that is, interestingly, a piecewise constant model because each tree is itself a piecewise constant model. However, one rarely notices because typically hundreds of trees are included in the ensemble.

The TreeNet algorithm, an example of stochastic gradient boosting, has won multiple data mining modeling competitions since its introduction and has proven to be an accurate predictor with the benefit that very little data cleanup is needed for the trees before modeling.

Heterogeneous Ensembles

The ensemble techniques described so far achieve variety through case weights, whether by resampling (bootstrap sampling) or reweighting records based on prediction errors (boosting). These techniques use the same algorithm for every model, typically decision trees. However, variety can also be achieved through varying the algorithm as well, creating a *heterogeneous ensemble*, comprised of two or more different algorithms.

As you saw in Chapter 8, algorithms sometimes achieve the same accuracy on data sets in different ways, disagreeing on individual records even while, on average, having the same accuracy. We observed bagged trees where the correlation between model predictions ranged from 0.212 to 0.431 even though they had the same accuracy.

Heterogeneous ensembles are combined similarly to bagged trees, most often through simple averaging or voting. However, one can also experiment with other ways to combine the models, such as selecting the maximum probability of the models or weighting the predictions by model accuracy (the better models contribute more to the weighted average).

In a simple example using data known as the glass identification data set, Figure 10-12 shows minimum, maximum, and average error rates for six different algorithms on validation data. The best single model on validation data had a 28.1 percent error rate, the worst a 37.5 percent error rate, and the average error rate is 30.8 percent.

The figure shows all possible ensembles built from these six models: all possible two-way combinations, three-way combinations, four-way, five-way, and the single six-way combination. Note that the average error rates on validation data decreases as more models are included in the ensembles. In fact, the *worst* three-model ensemble is better than the *average* single model. The six-model ensemble is among the best of all the ensembles, but it is not the best possible; you never know which particular combination will be the best on validation data.

Ensembles are described as methods that increase accuracy. Based on Figure 10-12, it is clear that ensembles also reduce the risk of deploying a poor model.

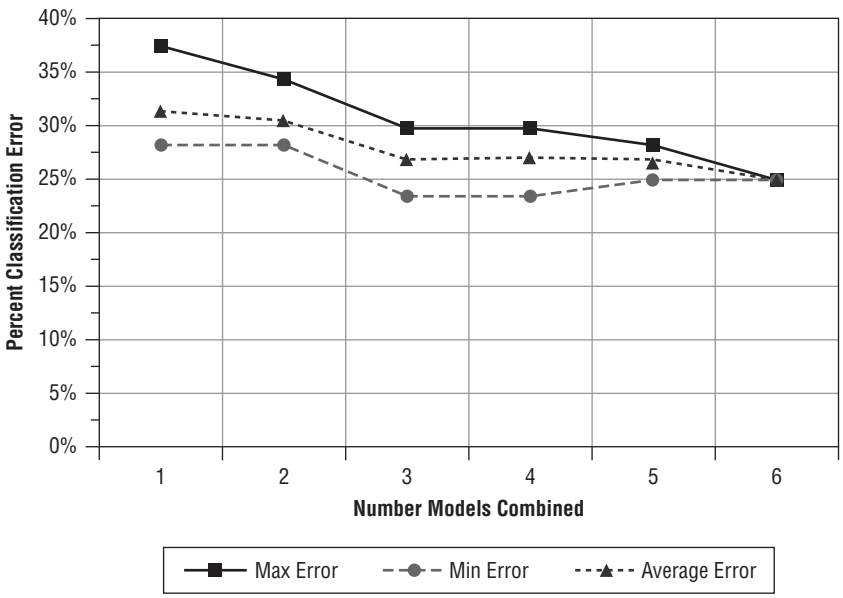


Figure 10-12: Heterogeneous ensemble example

Model Ensembles and Occam's Razor

Occam's razor is a core principle many predictive modelers use. The idea is that simpler models are more likely to generalize better, so it is better to regularize complexity, or in other words, simplify the models so that the inclusion of each term, coefficient, or split in a model is justified by its reducing the error sufficiently to justify its inclusion. One way to quantify the relationship between accuracy and complexity is taken from information theory in the form of information theoretic criteria, such as the Akaike information criterion (AIC), the Bayesian information criterion (BIC), and Minimum Description Length (MDL). Statisticians sometimes use these criteria for model selection during stepwise linear regression, for example. Information theoretic criteria require a reduction in model error to justify additional model complexity.

Do model ensembles violate Occam's razor? Ensembles, after all, are much more complex than single models. Random Forests and Boosted Trees could have hundreds of models combined into the final prediction. Neural network ensembles could contain dozens or hundreds of networks, each of which contain hundreds to thousands of weights. If the ensemble accuracy is better on held-out data than single models, yet these models are far more complex than single models, do we need another paradigm to help us choose which models will generalize well? The answer is "no" as long as we think about the complexity of a model in different terms.

John Elder describes the concept of Generalized Degrees of Freedom (GDF) that measures the *behavior* of models, or as he writes, the *flexibility* of a modeling process. The complexity of linear regression models increases linearly with the number of terms in the model. GDF confirms this as well: A linear increase in coefficients in a model produces a linear increase in the complexity of behavior in the model as well. However, this is not the case for all algorithms. Elder shows an example where Bagged trees built from 50 bootstrap samples have a lower GDF measure than a single tree. Bagging, in fact, smoothes the predictions as a part of the averaging of predicted probabilities, making the model's behavior smoother and therefore simpler.

We should not fear that adding computational complexity (more terms, splits, or weights) will necessarily increase the behavioral complexity of models. In fact, sometimes the ensemble will significantly reduce the behavioral complexity.

Interpreting Model Ensembles

The interpretation of ensembles can become quite difficult. If you build an RF ensemble containing 200 trees, how do you describe why a prediction has a particular value? You can examine each of the trees individually, though this is clearly not practical. For this reason, ensembles are often considered black

box models, meaning that what they do is not transparent to the modeler or domain expert.

Nevertheless, one way to determine which inputs to the model are most important is to perform a sensitivity analysis much like one does with neural networks: When inputs are changed by some deviation from the mean, and all other inputs are held constant, how much does output probability change? One can then plot the sensitivity of the change versus the value to visualize the sensitivity, seeing if the relationship between the input values and the predictions are linear or nonlinear, or if they are monotonic or non-monotonic.

A simple sensitivity analysis can be achieved by building single-split decision trees using an algorithm that allows for multi-way splits (such as CHAID) for each variable. All the variables can be scored, and the variable with the highest chi-square statistic can be considered the most important single variable. As an alternative, if the inputs to the model are continuous variables, an ANOVA can be computed with the predicted target variable class as the grouping variable.

Consider the KDD Cup 98 data set, this time including 12 candidate input variables to the ensemble models. Table 10-2 shows the ranking of the 12 variables based on an ANOVA, where the *F* statistic is the test of significance. The *F* statistic is computed on the test data set with 47,706 records. All of the variables have statistically significant differences in mean values, largely because of the large number of records in the test set. RFA_2F is by far the best predictor, its *F* statistic being more than seven times larger than the second variable. This gives you a sense for which variables are the largest contributors to the ensemble model.

Table 10-2: AdaBoost Variable Ranking According to ANOVA (*F* Statistic)

VARIABLE	F STATISTIC, ADABOOST	P VALUE, ADABOOST	RANK, ADABOOST
RFA_2F	53,560.3	0.0	1
NGIFTALL	7,862.7	0.0	2
MINRAMNT	6,400.0	0.0	3
LASTGIFT	5,587.6	0.0	4
FISTDATE	3,264.9	0.0	5
NUMPROM	2,776.7	0.0	6
MAXRAMNT	2,103.0	0.0	7
RAMNTALL	621.4	0.0	8
NUMPRM12	343.8	0.0	9
HIT	60.9	0.0	10
WEALTH1	60.0	0.0	11
AGE	36.5	0.0	12

A histogram of the top predictor, RFA_2F, color coded by the predicted target variable from the AdaBoost ensemble is shown in Figure 10-13. The figure shows the strong relationship between RFA_2F and the predictions; if RFA_2F is 4, the vast majority of predicted values are 1 (light gray).

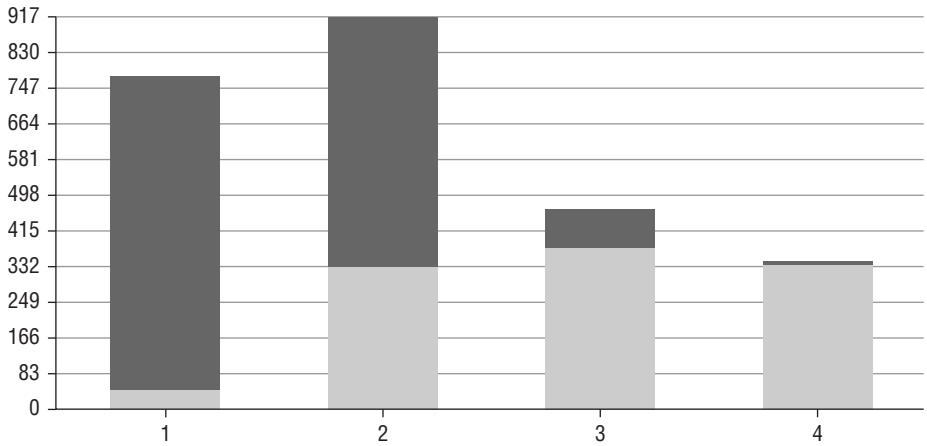


Figure 10-13: RFA_2F histogram with AdaBoost ensemble overlay

If one computes an ANOVA on testing data for Random Forests, AdaBoost, and bagging models, the resulting rankings in Table 10-3. There is great consistency in the story told by all three ensembles: RFA_2F is the top predictor, and the top six predictors are consistently top predictors.

Table 10-3: Comparison of Variable Ranking for Three Ensemble Methods by ANOVA (F Statistic)

VARIABLE	RANK, RANDOM FORESTS	RANK, ADABOOST	RANK, BAGGING
RFA_2F	1	1	1
NGIFTALL	2	2	3
MINRAMNT	3	3	4
LASTGIFT	5	4	2
NUMPROM	4	6	6
FISTDATE	6	5	7
MAXRAMNT	8	7	5
NUMPRM12	7	9	8
RAMNTALL	9	8	11
AGE	10	12	9
HIT	11	10	12
WEALTH1	12	11	10

Summary

Model ensembles are a new frontier for predictive modelers who are interested in accuracy, either by reducing errors in the models or by reducing the risk that the models will behave erratically. Evidence for this is clear from the dominance of ensembles in predictive analytics and data mining competitions: Ensembles always win.

The good news for predictive modelers is that many techniques for building ensembles are built into software already. The most popular ensemble algorithms (bagging, boosting, and random forests) are available in nearly every commercial or open source software tool. Building customized ensembles is also supported in many software products, whether based on a single algorithm or through a heterogeneous ensemble.

Ensembles aren't appropriate for every solution—the applicability of ensembles is determined by the modeling objectives defined during Business Understanding—but they should be in every predictive modeler's toolkit.

CHAPTER

11

Text Mining

In the early 2000s, I was a consultant on a contract that was analyzing maintenance data for the Harrier aircraft. The business objective of the modeling project was to identify pre-cursor events that led to more expensive maintenance actions so that proactive steps could be taken before expensive engine failures occurred.

Data was collected in two forms. During each flight, data was collected from a variety of sensors and included alerts that indicated potential problems during the flight, so the form of this data was time series with thousands of data points per aircraft per flight. A unique record in this data was a distinct flight-date-time triplet. In addition, historical maintenance actions for each aircraft were stored in a database, including the part used to complete the maintenance action.

The data was transformed so each record represented a maintenance event, including features of the most recent in-flight data (thousands of time-series data points flattened into a single record), recent maintenance actions (prior to the current maintenance action), and the most recent maintenance action taken. Each aircraft could have several entries in the data set: one record per flight the aircraft took.

The models were moderately effective, but something clearly was missing. There was considerable "noise" with maintenance actions for several reasons. First, the application of a maintenance action didn't necessarily mean the maintenance action was needed. In fact, for some mechanics, there were clear favorite actions;

they applied particular maintenance actions well in excess of norms. Second, just because a maintenance action wasn't done doesn't mean it wasn't needed.

However, there was another source of data that, according to domain experts, would have reduced the noise considerably and improved the target variable. These were logs the mechanics kept at the depot. These books, sometimes just hand-written notes, contained explanations for the maintenance actions and other. In addition, they contained "watch" items that could provide clues to future maintenance actions the aircraft would need. Unfortunately, these notes were not included in the database, and even if they were, no processes were put in place to extract the key information from the text.

This story is typical of many industries: Key data is either not in the transactional data stores, or is in the form of free text that cannot be used directly in predictive modeling.

Motivation for Text Mining

The value of predictive analytics is limited by the data that is used in the analysis. Thus far in the chapters of this book, whenever data is described, it is numeric or text stored in a database or in a flat file. However the data is stored, it is transformed, through queries or ETL operations, into rectangular data: Each record contains the exact same number of rows.

Text mining is not yet a part of mainstream predictive analytics, though it is on the short list for many organizations. But text mining is difficult, requiring additional expertise and processing complementary to predictive modeling, but not often taught in machine learning or statistics courses.

Nevertheless, text mining is being used increasingly as organizations recognize the untapped information contained in text. Social media, such as Twitter and Facebook, have been used effectively by organizations to uncover positive and negative trends that, when identified through text mining, can be used to leverage the positive trends and provide corrective action to counteract any negative comments.

Perhaps the most recent powerful example of text mining is the stunning success of IBM's Watson computer that combines text mining with predictive modeling and artificial intelligence techniques. Its first success on the television show Jeopardy!, while entertaining to watch, only scratches the surface of its potential in other fields such as healthcare analytics.

Surveys frequently contain free-form text fields to capture opinions not easily quantified as survey questions, but containing key views and explanations for why responders selected particular response values. News events, if captured in real time, help predict stock trends. Specifics of help-desk calls can help organizations identify immediately and automatically how to route the call to provide the best customer experience.

A Predictive Modeling Approach to Text Mining

Text mining can uncover insights into data that extend well beyond information in structured databases. Processing steps that interpret the text to uncover these patterns and insights can range from simple extraction of words and phrases to a more complex linguistic understanding of what the data means. The former treats words and phrases as dumb patterns, requiring minimal linguistic understanding of the data. This approach can apply, with minor modifications, to most languages. The latter requires extensive understanding of the language, including sentence structure, grammar, connotations, slang, and context.

This chapter takes the pattern extraction approach to text mining rather than a natural language processing (NLP) approach. In the former, the purpose of text mining is to convert unstructured data to structured data, which then can be used in predictive modeling. The meaning of the text, while helpful for interpreting the models, is not of primary concern for creating the features. While text mining of patterns is simpler, there is still considerable complexity in the text processing and the features can provide significant improvements in modeling accuracy.

Structured vs. Unstructured Data

"Structured data" is a label for the data that has been used throughout the chapters of this book. It is data that can, at least conceptually, be loaded into a spreadsheet. There are rows and columns in the data, where each row represents the unit of analysis for modeling, such as a customer, a business, a credit-card transaction, or a visit by a customer. The columns provide measurements, descriptions, and attributes related to the unit of analysis. Each cell in structured data is either filled or could be filled. The same number of columns exists for every row and the definition of the cell in a column is consistent for all rows.

Structured data often comes from databases, whether transactional, NoSQL, or ad hoc data stores. The data used for modeling is often created by combining tables from multiple data stores into a single view, table, document, or data file. Structured data is predictive modeling–friendly because data in this form can be loaded into any predictive analytics software package in a straightforward manner. All of the data described so far in this book is structured data.

Unstructured data is, in many ways an unfortunate label as there is almost always structure to "unstructured data." The layout of unstructured data is usually significantly different than structured data. Rows may contain only a single text chunk, or there may be a variable number of attributes in a single row. Even the concept of a row may vary within a single file or document. Columns don't exist naturally but could be generated through feature extraction. The

problem of a variable number of columns could be exasperating because of the variety of information in the unstructured data.

Some examples of unstructured data are listed in Table 11-1.

Table 11-1: Short List of Unstructured Data

DATA DESCRIPTION	WHAT MAKES IT UNSTRUCTURED
Microsoft Word, Adobe PDF files	Rows are not a unit of analysis, no columns in the data. Significant invisible characters without information valuable for text mining.
Powerpoint documents	Slides are not rows. Metadata contains formatting information that usually is not interesting for text mining.
HTML, XML	Rows are not necessarily a unit of analysis. Tags provide structure (semi-structured data), but should be stripped out before extracting text patterns; not all information within tags is interesting for modeling. No columns in data.
E-mail	Unit of analysis could be an e-mail or a recipient. Data is semi-structured, more valuable to group features (subject, body, signature, and so on).
Images, video	Rows and columns have meaning within a single instance, not usually as a description of multiple images or videos.
Audio/Music	Unit of analysis could be a song or a piece of a song. Columns would have to be generated from features of the music, like time series analysis.
New stories	Length of article and subject matter varies.

As you can see, the idea of "semi-structured" data, sometimes referred to as weakly unstructured data, is mentioned a few times in the table. This type of data has some cues to inform the analyst of the nature of text in the section, but provides only a modest amount of help to the analyst with feature creation.

Why Text Mining Is Hard

Text mining is generally much more difficult than typical predictive modeling problems. Problems include handling the variety of data, converting data from free-form text into structured data through feature creation, and managing the vast number of features that can (and often are) created. But at the core, the most difficult part of text mining is the mode of data itself: language. The discussion here focuses on English, but most of the principles also apply to other languages.

First, language is ambiguous and context is often required to understand the meaning of words and phrases. Consider the two sentences:

The beam could bear more weight; we could both walk on it safely.

The heavy bear could still walk in the forest without a sound.

In the first sentence, bear is a verb, meaning "to support or carry," whereas in the second sentence it is a noun, meaning "a large animal . . ." Obviously, the sentences have very different meanings even though they share this key word. Parts of speech could differentiate these uses of the same word, though part of speech is not always enough.

Types of ambiguity are homonomy, synonymy, polysemy, and hyponymy. With *homonomy*, the word is the same but the meaning is different through the evolution of the words through history. For example, consider two different meanings for the same word "bank":

Cindy jogged along the bank of the river.

CityBank is the oldest bank in the city.

In the first case, "bank" clearly has different meanings even though the part of speech is the same. In one case, it refers to a geographical description, in the other a specific financial institution.

With *polysemy*, the same word is used but has different, albeit related, meanings.

The Central Bank raised its prime interest rates yesterday.

The new coffee shop was built right next to the old bank.

The first bank was established in Venice in 1857.

In all three cases, the concept of the bank as a financial institution is in mind. However, in the first case, "bank" is a broad concept, referring to an overseeing entity. In the second case, "bank" refers to a specific company, operating in the city. In the third case, "bank" is a historical idea; while referring to a specific building, it refers to more than just that bank; it implies a continuation of the history of the banking industry from that time to the present.

Synonomy refers to different words that have similar meetings, sometimes interchangeable meanings. For example, in these two sentences, "large" and "big" are used interchangeably:

The pile of trash was very large.

The pile of trash was very big.

You must be careful of connotations, however, as sometimes the meanings can be different enough to convey a different kind of meaning:

Beth, now 25 years old, became a kind of big sister to Jenny.

Beth, now 25 years old, became a kind of large sister to Jenny.

Obviously in the first case, the connotation is positive, with Beth acting in a generous way. In the second case, the connotation is negative referring indirectly

to Beth's weight rather than her actions. This is only because of the use of these words colloquially in English.

A fourth source of ambiguity in language comes from *hyponymy*, a concept hierarchy that exists in words, where one word is a subset or subclass of another. Dogs and cats are subclasses of animals, or broken legs and contusions are subclasses of injuries. Depending on the purpose of the analysis, the broader concept (animal or injury) may be sufficient or not specific enough.

A fifth type of ambiguity comes from spelling variants or misspellings of words. Misspellings are sometimes obvious and easily corrected, though sometimes the intended word is not clear except from the context. Spelling variants become similar to synonyms for the analyst: two different patterns that should be interpreted the same way. Some variants are differences between American English and British English, such as the pairs color and colour, favorite and favourite, labeled and labelled, or theater and theatre. There are still some variants even within American English, such as donut and doughnut, gray and grey, or worshiped and worshipped.

A sixth type of ambiguity comes from abbreviations and acronyms:

> Renders search engines, SQL queries, Regex, . . . ineffective

Text Mining Applications

Predictive analytics has four kinds of analysis—classification, regression, clustering and rule mining—which can be grouped into the two broader categories, supervised and unsupervised learning. Text mining, in contrast, is often characterized as having seven or more types of analysis, five of which are enumerated in Table 11-2. These are the analyses that predictive modelers are most likely to do.

Table 11-2: Text Mining Applications

TEXT MINING APPLICATION	DESCRIPTION
Information retrieval	Match documents from a list of keywords. Examples include website searches, patent searches, and predictive modeling conference abstracts.
Document clustering	Group documents based on keywords and phrases from each document.
Document classification	Label documents from keywords and phrases for each document.

TEXT MINING APPLICATION	DESCRIPTION
Sentiment analysis	Special case of document classification. Predict positive, negative, or neutral sentiment of tweets, comments, or customer reviews based on keywords and phrases.
Information extraction	Extract and summarize keywords and phrases that match document, for example, extracting courses taken and degrees from resumes.

Data Sources for Text Mining

Data containing text for predictive modeling can include a wide variety of sources. Text mining software can typically load each of these types of data:

- **Web page scraping:** An algorithm crawls through a website. Each web page is a document. Usually, metadata and tags are stripped out.
- **Twitter/Facebook feeds:** Usually accessed via an API
- **Folders of text files:** Each folder stores a particular class value for document classification or sentiment analysis. Document types can be plain text, PDF, or Word documents.
- **Text fields within structured data:** Free-form text field within a structured data set (for example, comment fields in a survey)

Languages common to predictive modelers, such as R and Python, have APIs that can import these and a wide variety of additional data for text mining.

Data Preparation Steps

Typical data preparation steps are shown in Figure 11-1. Not all of these steps necessarily must be undertaken depending on the specific application.

POS Tagging

After any initial cleaning of the text, such as correcting misspellings, a Part of Speech (POS) tagger is often invoked. Eighty-nine percent of English words have only one part of speech, and therefore labeling POS doesn't provide any additional information. However, as noted already, many words are ambiguous and POS tagging provides one method of disambiguating the text. In addition,

it can help recognize and group nouns or noun phrases, often a key part of a text mining analysis.

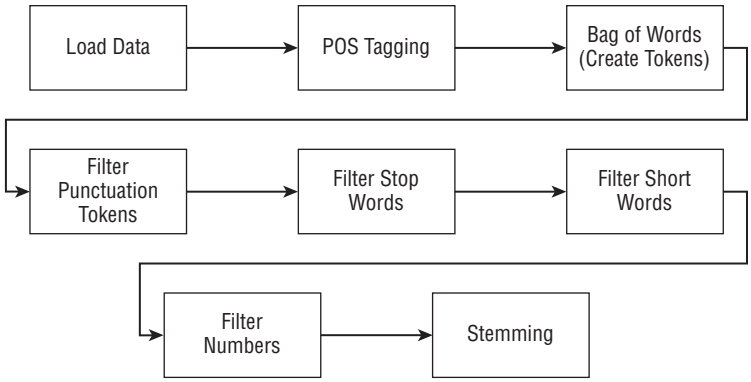


Figure 11-1: Typical text mining preprocessing steps

A minimum set of POS includes nouns, verbs, adjectives, and adverbs. A slightly expanded list will sometimes include prepositions and conjunctions. Some lists contain more than 100 parts of speech. The most popular list comes from the Penn Treebank, containing the 36 parts of speech shown in Table 11-3.

Table 11-3: Penn Treebank Parts of Speech (Word Level)

TAG	DESCRIPTION	TAG	DESCRIPTION
CC	Coordinating conjunction	PRP$	Possessive pronoun (prolog version PRP-S)
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential there	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
JJ	Adjective	TO	to
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non–third person singular present

NNP	Proper noun, singular	VBZ	Verb, third person singular present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WP$	Possessive wh-pronoun (prolog version WP-S)
PRP	Personal pronoun	WRB	Wh-adverb

Text mining software typically has a POS tagger; rule-based taggers, such as the Brill POS tagger, are almost always available. Stochastic taggers are also common and can even be developed by predictive modelers if the built-in taggers are not sufficient. Consider the following sentence with parts of speech labeled above each word. All of the words are straightforward to tag except for the word "race." In the first tagged sentence, race is labeled a verb (VB) and in the second sentence a noun (NN). Which is correct?

NNP VBZ VBN TO VB NR
Secretariat is expected to race tomorrow

NNP VBZ VBN TO NN NR
Secretariat is expected to race tomorrow

A stochastic POS tagger uses a corpus of documents already tagged, such as the *Wall Street Journal* corpus, to compute probabilities of pairs of words, such as the likelihood a verb (POS VB) follows the word "to" (POS TO) and the likelihood a noun (POS NN) follows TO. You might find the following probabilities and conclude that the most likely POS for race is VB:

$P(NN \mid TO) = 0.00047$

$P(VB \mid TO) = 0.83$

When tagging a larger block of text, the tags are sometimes shown within the text. Consider the following text posted in a predictive analytics blog:

In this talk, Mr. Pole discussed how Target was using Predictive Analytics including descriptions of using potential value models, coupon models, and . . . yes . . . predicting when a woman is due (if you aren't the patient type, it is at about 34:30 in the video).

These models were very profitable at Target, adding an additional 30% to the number of women suspected or known to be pregnant over those already known to be (via self-disclosure or baby registries).

The tagged version appears like this:

*In/**IN** this/**DT** talk/**NN** ,/, Mr./**NNP** Pole/**NNP** discussed/**VBD** how/**WRB** Target/**NNP** was/**VBD** using/**VBG** Predictive/**NNP** Analytics/**NNP** including/*

VBG descriptions/NNS of/IN using/VBG potential/JJ value/NN models/NNS ,/, coupon/NN models/NNS ,/, and . . . yes . . . predicting/VBG when/WRB a/DT woman/NN is/VBZ due/JJ (/(if/IN you/PRP are/VBP n't/RB the/DT patient/JJ type/NN ,/, it/PRP is/VBZ at/IN about/IN 34:30/CD in/IN the/DT video/JJ)/) ./.

These/DT models/NNS were/VBD very/RB profitable/JJ at/IN Target/NNP ,/, adding/VBG an/DT additional/JJ 30/CD %/SYM to/TO the/DT number/NN of/IN women/NN suspected/VBD or/CC known/VBN to/TO be/VB pregnant/JJ over/IN those/DT already/RB known/VBN to/TO be/VB (/(via/IN self-disclosure/JJ or/CC baby/NN registries/NNS)/) ./.

Often, the tagged version of the text is not exposed in text mining software but is tracked for use once the tokens are created.

Tokens

Tokenization is a process that takes a stream of characters and splits them into individual tokens, most often words. In English, the most obvious clues for how to create tokens come from characters that split words, such as spaces and punctuation. The creation of tokens from streams of text is sometimes called creating a *bag of words*, or BoW.

White space does not guarantee the separation of tokens, such as in Asiatic languages (Chinese, Japanese, Korean) and German (compound nouns) where several tokens comprise a single character or word, and therefore these algorithms are language specific. Tokenization algorithms, therefore, are specific to a language

Stop Word and Punctuation Filters

Stop words are words that are uninteresting for the analysis. They are removed because they may occur so often that they don't differentiate documents or sentiment. Removing stop words is done primarily to reduce processing time and dimensionality of the tokens to be processed. A typical list of typical stop words is:

> a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with

Care must be taken in determining which words are uninteresting prior to analysis. Common stop words such as "the" may be essential in identifying particular names such as "The Who" or identifying superlatives such as "the worst" or "the best."

Punctuation filters remove the punctuation tokens from the bag of words. Usually, though not always, punctuation is not interesting for predictive models.

Applications may arise, for example, when knowing a token appears as the first or the last token in a sentence may be interesting in an analysis.

Character Length and Number Filters

Usually, short words and long words are not of interest. One or two character words are usually listed as stop words, though others such as "me," "we," and "so" are usually not interesting. Software often has options to limit tokens to a minimum and maximum length. Numbers are also typically filtered out to speed up processing.

Stemming

Stemming normalizes a word by reducing it to its most basic element or stem. Typical normalization steps with stemming include:

- Removing affixes (prefixes and suffixes)
- Removing plurals
- Normalizing verb tenses

Table 11-4 shows a short list of words and their stemmed form. The result of stemming is a reduction in the number of unique tokens, simplifying subsequent processing steps.

Table 11-4: Stemmed Words

ORIGINAL WORD	STEMMED WORD
walks	walk
walk	walk
walked	walk
walking	walk
walker	walker
walkers	walker
denormalization	denorm
normalize	normal
norms	norm
normed	norm
renormalize	renorm
title	titl

However, there are also potential problems with stemming. First, stemming is language dependent; each language must have its own stemming rules. Second, if stemming is too aggressive, the results will be confusing and misleading. If stemming is too gentle, there will be little reduction in the number of tokens.

The most popular stemming algorithm is the Porter Stemmer, usually considered a mild stemming algorithm. The Snowball Stemmer is perhaps the next most popular and is usually considered superior to the Porter Stemmer. A third algorithm, the Lancaster Stemmer, is very aggressive and should be used with care.

Table 11-5 shows a comparison of the stemming results after applying each of these three algorithms. Note that the Snowball Stemmer is the only one of the three that produces a variety of stemming results for this group of words, as you would expect from this particular group. On the other hand, the token "organization" reduces, surprisingly, to the stem organ with both Porter and Snowball.

Table 11-5: Comparison of Stemming Algorithms

WORD	PORTER STEMMER	SNOWBALL STEMMER	LANCASTER STEMMER
generate	gener	generat	gen
generates	gener	generat	gen
generated	gener	generat	gen
generating	gener	generat	gen
general	gener	general	gen
generally	gener	general	gen
generic	gener	generic	gen
generically	gener	generic	gen
generous	gener	generous	gen
generously	gener	generous	gen
organization	organ	organ	org
urgency	urgen	urgen	urg

Dictionaries

A standard bag of words will identify as a token any combination of characters. Dictionaries can be valuable to reduce the set of tokens to be extracted by identifying only the words of particular interest in an analysis. The dictionaries can include proper nouns, special phrases, acronyms, titles, numbers, or any sequence of characters important in the analysis. They can also include all

acceptable variations of a token, such as including MD, M.D., MD., and M D. Unless the dictionary is already defined, such as a medical dictionary of terms, it can be very time consuming to create.

The Sentiment Polarity Movie Data Set

The data used for examples in this chapter comes from the Sentiment Polarity Data set introduced by Pang and Lee in 2004 (`http://www.cs.cornell.edu/people/pabo/movie-review-data/`). The original data contained 1,000 positive and 1,000 negative movie reviews, each review in a separate text file and labeled with the sentiment. The data described here was reduced to 101 positive and 101 negative reviews. These reviews have on average 700 words each.

After applying the processing steps outlined in Figure 11-1, Table 11-6 shows the number of rows and terms remaining after each filtering step.

Table 11-6: Record Counts and Term Counts after Data Preparation

DATA PREPARATION STAGE	NUMBER OF ROWS	NUMBER OF UNIQUE TERMS
All reviews	202	NA
Bag of Words Creation	71,889	18,803
Punctuation Eraser	70,348	18,711
Stop Word Filter	46,029	17,959
N Char Filter	45,058	17,778
Number Filter	44,876	17,683
Porter Stemmer	44,786	17,171
Remove POS tags	40,984	10,517
Group By Document	203	10,517

The processing begins with 202 reviews from the sentiment polarity data: 101 positive reviews and 101 negative reviews. After identifying tokens and running the POS tagger, 18,803 tokens (terms) exist in the 202 documents. The term count refers to one or more occurrences of the term in the document rather than a full listing of every occurrence of each term. The 71,889 rows represent document-term pairs. Removing punctuation terms, stop words, 1- or 2-letter words, and numbers reduces the number of terms down to 17,683, represented in 44,876 rows.

Stemming reduces the number of terms in this data set modestly. Interestingly, removing the POS tags in this data reduces the number of terms. Table 11-7 shows a single term, "boring," represented as three different parts of speech as it exists in five different documents. After stripping the POS tags, documents

72 and 97 would have its TF value change as a result of grouping the values for the two parts of speech.

Table 11-7: Example of Terms Grouped after Stripping POS

TERM [POS]	DOCUMENT NUMBER
boring[VBG(POS)]	61
boring[VBG(POS)]	72
boring[NN(POS)]	72
boring[JJ(POS)]	73
boring[VBG(POS)]	81
boring[JJ(POS)]	97
boring[VBG(POS)]	97

Text Mining Features

After all the data preparation and preprocessing has been completed, the data is still only a listing of tokens. For predictive modeling, these terms have to be converted into a form useful for predictive modeling algorithms. If the unit of analysis is the document, then the terms are exploded into additional columns in the data, one column per term. For the movie review data, if no stemming is done and POS tags are retained, 17,683 terms remain (Table 11-6). Since a feature is created for each term, 17,683 new columns are created in the data.

This large number of new columns is often impractical, but more important, the vast majority of these terms are rare—the long tail effect. In fact, nearly two-thirds of the terms identified in the data appear in only one document, making them completely unusable for modeling.

Table 11-8 shows the typical long-tail effect of term extraction. Note that 11,439 terms appear in only one document of the 202 review (65 percent). Terms that appear in only one or even in a few reviews, of course, will not be useful in predicting sentiment. These can safely be removed from a document clustering or classification application, though not necessarily from information retrieval.

Table 11-8: Number of Times Terms Appear in Documents

NUMBER OF DOCUMENTS IN WHICH THE TERM OCCURS	NUMBER OF TERMS WITH THIS OCCURRENCE COUNT
1	11,439
2	2,526
3	1,120
4	668
5	435
6	249
7	224
8	154
9	114
10	107
11–20	391
21–30	122
31–50	51
51–150	33

Term Frequency

Term Frequency (TF) is a count of how often a term is used in each document. The simplest representation of TF is to simply create a 1/0 label—a flag—indicating the existence of the term in the document. This is the same idea as was described in Chapter 4 as "exploding" a categorical variable into dummies. Sometimes this 1/0 form of TF is described as the "Boolean frequency," although because it is not a frequency but merely an indicator of existence, this is a stretch in terminology. Nevertheless, it is a helpful description in comparison to other forms of TF.

Table 11-9 shows a small table with five terms and the TF flag form of the term representation of the data. This version did not differentiate between different parts of speech, and therefore did not resolve situations where both parts of speech showed in a single document. These can be tracked separately or together depending on the interest of the analyst. But this table is not yet ready

for modeling because each record does not represent a document (a review in this case), but rather represents a term within a document.

Table 11-9: Boolean TF Values for Five Terms

DOCUMENT ID	TERM[POS]	DESPITE	WORST	ENJOY	MOVIE	FILM
1	despite[IN(POS)]	1	0	0	0	0
1	enjoy[VB(POS)]	0	0	1	0	0
1	film[NN(POS) VB(POS)]	0	0	0	0	1
2	movie[NN(POS)]	0	0	0	1	0
2	film[NN(POS)]	0	0	0	0	1
5	worst[JJS(POS)]	0	1	0	0	0
5	movie[NN(POS)]	0	0	0	1	0
5	film[NN(POS) VB(POS)]	0	0	0	0	1
77	worst[JJS(POS)]	0	1	0	0	0
77	enjoy[VB(POS)]	0	0	1	0	0
77	movie[NN(POS)]	0	0	0	1	0
77	film[NN(POS)]	0	0	0	0	1

Table 11-10 shows the form of the data that would be used in modeling once the data is collapsed to one record per document (i.e., group by Document ID).

Table 11-10: Boolean TF Values for Five Terms after Rolling Up to Document Level

DOCUMENT ID	DESPITE	WORST	ENJOY	MOVIE	FILM	SENTIMENT
1	1	0	1	0	1	pos
2	0	0	0	1	1	pos
5	0	1	0	1	1	neg
77	0	1	1	1	1	neg

TF based on the actual count is the next level of complexity in representing terms. Rather than simply flagging if a term exists in the document, the actual number of times the term appears in the document is shown. If a term appears more often in a document, it may be more strongly related to the target variable. Most often, TF will have a value equal to 1, just like the Boolean form. But

sometimes the value could be quite large. Table 11-11 shows the same documents and terms with their TF values.

Table 11-11: Boolean Form of TF Features

DOCUMENT ID	DESPITE	WORST	ENJOY	MOVIE	FILM	SENTIMENT
1	1	0	1	0	14	pos
2	0	0	0	2	2	pos
5	0	1	0	6	6	neg
77	0	1	2	13	8	neg

In the sentiment polarity data the term "movie" has the maximum TF value (28), which is no surprise for movie review data. Note that the longer the document, the more likely that TF will have larger values. If the documents are long and the raw TF value has a long tail, computing the log transform of TF can help normalize the values as described in Chapter 4. Table 11-12 shows the values of TF and the corresponding values for the log transform of TF—actually, log10(1 + TF)—for the sentiment polarity data. The Number of Terms Matching column counts how many terms had a TF value in the TF column.

The vast majority of terms occurred only one time in any review (10,074). If TF is used directly in modeling, a few terms would have values that are larger—only 19 terms were in a review 10 times. It is therefore heavily skewed and may bias models or summary statistics. The log transform will reduce the amount of skew in TF. The Boolean frequency for each value of TF would be 1.

Table 11-12: TF and Log10(1+TF) Values

TF	NUMBER OF TERMS MATCHING	LOG10 TF
1	10,074	0.301
2	2,427	0.477
3	679	0.602
4	260	0.699
5	118	0.778
6	84	0.845
7	42	0.903
8	28	0.954
9	15	1.000
10	19	1.041

Another technique that is sometimes used is to smooth the TF value by scaling the value by the maximum TF value—the min-max transform that puts TF on a scale from 0 to 1. This approach reduces the magnitude of TF but does not eliminate the long tail.

Inverse Document Frequency

Document frequency (DF) represents the number of times the term appears in all documents in the corpus. The intent of this feature is to measure how popular a term is. If the term is found in only a few documents, it could be perceived that it is more specific to those particular documents. This is especially helpful for document clustering and document classification where the terms are helping to differentiate between documents. It is also helpful in Information Retrieval because the term will identify documents to be returned by the request because the intent of the request is clearer.

A lower value of DF, therefore, is better if the intent is finding words and terms that are specific to the documents of interest. However, many analysts prefer larger values of the feature to indicate better, leading them to compute the inverse document frequency (IDF). IDF, then, is the total number of documents in the corpus divided by the number of documents with the term of interest in it.

This quotient is often heavily skewed as well, just like TF tends to be. Because of this, analysts usually take the log transform of the quotient and call the log-transformed quotient IDF, usually with 1 added to the quotient before taking the log transform. IDF is thus a constant for each term; the value will be repeated for every document that term appears in, as shown in Figure 11-13.

Table 11-13: Log Transformed IDF Feature Values

DOCUMENT ID	DESPITE	WORST	ENJOY	MOVIE	FILM	SENTIMENT
1	0.990	0	1.045	0	0.333	pos
2	0	0	0	0.374	0.333	pos
5	0	0.928	0	0.374	0.333	neg
77	0	0.928	1.045	0.374	0.333	neg

TF-IDF

IDF on its own is not usually a good feature for building predictive models. However, if a term is rare, but occurs frequently within a single document, it may be a term that conveys meaning and insight to that document. If, for example,

you are trying to cluster a corpus of abstracts from predictive analytics conferences, terms that appear several times in only a few of the abstracts could be good surrogates for what differentiates those talks from others. A few of the abstracts may include "decision trees" several times, but not the majority of abstracts. A few other abstracts may include the text "social media" or "social network" several times. The combination of high TF and high IDF (meaning terms that are relatively rare in the corpus) could be the best separator of documents.

The simple product TF × IDF is a good feature to include as a potential input for predictive models. If a term appears frequently within a document (high TF) and doesn't appear frequently in the corpus (high IDF), then this term is highly connected to the particular document. If a term appears frequently within a document but also in many other documents (low IDF), then this term is not highly connected to the particular document.

Why multiplication? The multiplication operator is often used to measure interactions between variables. One difference between using a multiplicative approach rather than an additive approach to combining TF and IDF—creating a feature TF + IDF—is that the additive version will produce a relatively higher score value if only one of the two values is high. The multiplicative approach requires both values to be higher for a high score to be the result. Nevertheless, TF × IDF is the commonly used feature used in text mining.

The TF × IDF values for the same documents and features from Tables 11-12 and 11-13 are shown in Table 11-14. The term "worst" has the same values for both Document ID 5 and 7 because the TF value is the same for both, whereas "enjoy" has different TF values for Document IDs 1 and 77, yielding different values.

Table 11-14: TF-IDF Values for Five Terms

DOCUMENT ID	DESPITE	WORST	ENJOY	MOVIE	FILM	SENTIMENT
1	0.2982	0.000	0.3147	0.000	1.2533	pos
2	0.000	0.000	0.000	0.1125	0.1003	pos
5	0.000	0.2795	0.000	0.3159	0.9006	neg
77	0.000	0.2795	0.4987	0.4284	0.3181	neg

TF-IDF is used particularly in information retrieval applications because of the beneficial normalizing effect of the IDF component. For document classification and sentiment analysis, it may be worth trying as a feature, but doesn't necessarily provide value; just because terms are specific to a particular document doesn't mean that the terms are good at classifying groups of documents from other groups (positive sentiment from negative sentiment, for example).

Cosine Similarity

Cosine similarity is the classical approach to measuring how similar two documents are to each other. The measure used could be TF-IDF, Boolean TF, TF, or some other feature. If the Boolean TF value is used for each term, the similarity measure computes how many terms the documents have in common.

If there are 200 documents in the corpus, there are $200 \times 200 = 40{,}000$ similarity measures; the measure doesn't necessarily scale well to large numbers of documents. However, if you first have clustered the document data, the similarity measure can be used after clustering to measure the distance of a document to each cluster. If there are M clusters, this is now $M \times 200$ measures, a significant reduction in computational complexity.

The cosine similarity, however, does not consider term order within the document, only the existence of the terms in the document.

Multi-Word Features: N-Grams

All of the features so far have focused on single terms. Clearly, multiple terms could and often are more predictive than single terms alone. These multiple word phrases are called *N grams*, defined as n-word phrases of adjacent words. The most common n-grams are 2-word phrases called *bigrams* or *digrams*. Trigrams are 3-word phrases. Longer n-gram phrases are labeled by the number of adjacent words: 4-grams, 5-grams, and so on.

Consider the sentence, "It's one of the worst movies of all time." The bigrams for this sentence are shown in the bulleted list where each bullet contains a single bigram. Note that even a period character (".") represents a term.

- it 's
- 's one
- one of
- of the
- the worst
- worst movies
- movies of
- of all
- all time
- time .

These bigrams then become new patterns that can be used as features if they appear frequently enough in the corpus; they are evaluated the same way single-word terms are evaluated.

Reducing Keyword Features

A common problem with text mining is the number of features that are created. Table 11-6 showed that after processing the sentiment polarity data set, 17,683 terms remained. If all of these variables converted to Boolean term frequency features, there would be 17,683 new columns to use as inputs to a predictive model. Clearly this is too many inputs for most modeling approaches.

A second problem with text mining features is sparseness; most of the terms found in a corpus of documents occur rarely, resulting in poor predictive power. Of the 17,683 terms found in the corpus:

- 754 terms appear in 10 or more documents (5 percent of the documents)
- 231 terms appear in 20 or more documents (10 percent of the documents)
- 84 terms appear in 30 or more documents (30 percent of the documents)

Retaining only those terms that occur in sufficient numbers of documents (reviews in the sentiment data) will reduce dimensionality significantly. In supervised learning problems, you can also remove terms that by themselves are poor predictors of the target variable.

Grouping Terms

Some methods of grouping terms together have already been described, such as the use of stemming and synonyms. Stemming groups words related by a common stem, and synonyms group terms that have similar meaning. Identifying synonyms and replacing a term with its synonym can also reduce the number of terms. (Synonyms should be identified prior to removing sparsely populated terms.)

Even after reducing terms by eliminating sparse terms and grouping similar terms, numeric techniques can be used to reduce dimensionality. Principal Component Analysis (PCA) can be applied to TF or TF-IDF features to reduce the number of inputs to predictive models while still describing a high percentage of the variance in the data. The components themselves can be used as the inputs to the supervised or unsupervised learning models.

Modeling with Text Mining Features

The steps described so far in this chapter outline how to convert unstructured data into structured data that can be used for predictive modeling. The original text fields become columns in a structured representation of the terms in the text. Now that the data has been converted to a structured format, the principles for supervised or unsupervised predictive models described in Chapters 6–10 can be applied.

As one simple example of creating models with the sentiment polarity data, consider a K-Means model built from Boolean TF features. The full set of terms extracted by text mining techniques is far too large, far too correlated, and far too sparsely populated to apply directly. Only terms that were present in at least 20 reviews but not more than 100 reviews are included to reduce the number of inputs to the model. If the term is present in fewer than 20 reviews, there is concern that patterns could occur by chance. If the term is present in more than 100 reviews, it is too common a term. These two thresholds are just rules of thumb and should not be considered rules to apply to every data set.

A 5-cluster model was built, summarized in Table 11-15, with only a few of the terms included in the model displayed. The percentages convey what percentage of the reviews in the cluster have the sentiment (positive or negative) or include the term (worst, entertainment, etc.). Cluster 3 contains predominantly positive reviews; 91 percent of the reviews in Cluster 3 are positive. The terms that appear most often in Cluster 3 include "entertainment," "entire," "family," and "effective." In fact, 20 of the 23 (87%) of the reviews in Cluster 3 contain the term "entire". Also note that the term "worst" never appears in these reviews.

The most negative cluster is Cluster 2 and contains terms "stupid," "entire," and "worst" in relatively high proportions. The term "entire," therefore, can have positive or negative sentiment, depending on the other terms that exist with it in the review.

The key point is that the cluster model provides insight to how terms interact with each other in the reviews, and can reveal how those interactions relate to sentiment.

Table 11-15: Five-Cluster Model Results

MEASURE OR TERM	CLUSTER 1	CLUSTER 2	CLUSTER 3	CLUSTER 4	CLUSTER 5
Number of Reviews	73	8	23	29	29
Negative Reviews, percent	58	63	9	31	62
Positive Reviews, percent	42	38	91	69	38
worst, percent	21.9	50.0	0.0	3.4	20.7
entertainment, percent	8.2	0.0	26.1	10.3	27.6

entire, percent	0.0	100	87.0	0.0	0.0
history, percent	11.0	50.0	8.7	6.9	13.8
stupid, percent	0.0	62.5	0.0	10.3	48.3
supporting, percent	0.0	0.0	26.1	6.9	55.2
performances, percent	0.0	25.0	17.4	100	0.0
family, percent	13.7	0.0	34.8	17.2	0.0
enjoy, percent	15.1	25.0	17.4	6.9	3.4
effective, percent	15.1	12.5	21.7	10.3	6.9

Regular Expressions

Regular expressions (REs) are power pattern matchers for strings. Their power comes from the syntax that includes a variety of wildcards to match strings in very flexible ways, enabling REs to match variable-length patterns of characters, existence or non-existence of one or more characters, repeated patterns of one or more characters, and even gaps between characters or words of interest. The types of patterns that regular expressions can find extend well beyond finding single terms in a corpus of documents, and are far more flexible than standard string operations found in Excel, C, Java, or SQL. Packages, however, have been developed not just for Python, Perl, R, and other languages, but also for C, Java, and other languages.

REs are brute-force pattern matchers, however, and not semantic learners. They don't care about language (English, Spanish, German, and so on), nor do they care about the meaning of what they find. Nevertheless, REs are often the backbone text search and extraction.

The purpose of this section is not to provide a comprehensive description of REs, but rather to provide an overview of the power of REs and some typical uses of them. Some examples of REs to find specific kinds of patterns are

shown in Table 11-16. The quotation marks in the third column only set off the characters that match or don't match the regular expressions, but are not a part of the match itself.

Table 11-16: Key Regular Expression Syntax Characters

RE SYNTAX	DESCRIPTION	EXAMPLE
.	Any single character	mo.el matches "model," "motel," "mobel," "mo!el," and so on.
^	Matches the beginning of a line	^A.* matches "A Saturday" but not "Saturday."
$	Matches the end of a line	end.$ matches "the end." but not "the ends."
\b	Matches a word boundary	model\b matches "model" from "model prediction" but not "model" in "modeling."
*	Matches zero or more occurrences of previous expression	.* matches any character, m* matches "", "model," and "mom," but not "ant."
+	Matches one or more occurrences of previous expression	m+ matches "model," "mom," but not "" or "ant."
?	Matches 0 or 1 occurrences of previous expression	"mo?el" matches "model," "motel," and "moel," but not "mdel."
[ ]	Matches one character inside the square brackets	"mo[dt]el" matches "model" and "motel," but not "Motel" or "mobel."
[a-zA-Z0-9]	Matches any lowercase character between a and z, any uppercase character between A and Z, and any number between 0 and 9	Matches "m,", "1," and "M," but not a comma, space, colon, or dollar sign, to name a few non-matches.
{a,b}	Matches from a to b occurrences of the previous expression	a{1,3} matches "a," "aa," and "aaa."

Uses of Regular Expressions in Text Mining

REs are useful in text mining as an addition to standard processing techniques usually provided in text mining software. The following are some examples:

- **Non-dictionary terms:** Some applications of text mining have language very specific to the domain. Consider auto insurance claims investigations that include auto accidents with injuries to passengers. Most medical descriptions can be found in medical dictionaries available publicly. However, shorthand terminology and insurance-specific terms may not be available in dictionaries. REs can capture keywords specific to adjusters with their variants as applied by different adjusters.
- **Abbreviations and Acronyms:** This is related to the identification of non-dictionary terms. Some domains have a wealth of abbreviations and acronyms that are not specified in a dictionary, such as MD and USA.
- **Multi-word features that may be separated by a number of words:** Sometimes n-grams are a useful idea, but the key words are not adjacent. Moreover, word order may be critical to conveying the meaning of the multiple word features. REs provide an easy way to capture the existence of these patterns.

  For example, flagging the existence of "negative" for identifying negative sentiment is a good idea, but the word "but" could negate the negative. "Negative . . . but" may be a helpful word pair to identify. Often, this could be found with the trigram "but" followed by a comma, followed by "but." However, consider the following sentence:

  > The film has gotten some negative reviews (a friend of mine actually thinks it's the worst in the series), but I'm not really sure why.

  The RE "negative\W+(?:\w+\W+){1,14}?but" finds the word "negative" followed by one or more non-word characters, followed by 1 to 14 word/non-word pairs, followed by the word "but," and will therefore find "negative" followed by 1 to 14 words, followed by the word "but," so it will find the underlined phrase within the sentence. The feature (let's call it "negative . . . but") would then be populated with a 1 if the pattern matched, 0 otherwise.

  > The film has gotten some **negative reviews (a friend of mine actually thinks it's the worst in the series), but** I'm not really sure why.

- **Misspellings of words uncorrected by dictionary or phonetic spelling correction algorithms:** Some words can be misspelled in the most unforeseen, even clever, ways. In fraud detection applications, some analysts have found that misspellings are done purposefully to help some transactions avoid detection. Or, when text is typed or transcribed, seemingly random spelling variants are sometimes introduced.

For example, if a call center is identifying if a printer is "sticking," an RE can find variants that won't be found with stemming algorithms, such as "stck," "stic," "stick," "stickin," "sticking," "sticks," and "sticky."

Three examples of common REs used commonly for text mining on e-mail messages or web pages are:

- **E-mail addresses:** \b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}\b
- **Phone numbers:** \(?[0-9]{3}\)?[-][0-9]{3}[-][0-9]{4}
- **URLs:** ^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$

These are shown as examples of the power of REs to find simple, highly structured patterns, like phone numbers, or more complex patterns like those found in e-mail addresses and URLs.

Summary

Text mining is still a leading-edge approach for organizations, though its use is rapidly increasing. The mathematics behind text mining is fundamentally different from what is found in predictive analytics algorithms, and therefore requires that an analyst add different skills than he or she uses for predictive analytics. Fortunately, text mining add-ins or packages are available in most of the most commonly-used predictive analytics software packages so that the results from the text processing can be added to the structured data easily and therefore be used with standard supervised and unsupervised modeling algorithms.

Predictive modelers who need to customize text processing often rely on regular expressions, though not all predictive analytics software packages support regular expressions well or at all. Modelers may have to use a separate scripting or programming language to leverage regular expressions. The benefits, however, often justify the burden of learning the new language.

CHAPTER

12

Model Deployment

Model Deployment is the sixth and final stage of the CRISP-DM process, incorporating activities that take place after models have been built, validated, and are ready for use. These activities include documenting the modeling project, using the models operationally, monitoring the models as they are used operationally, and planning for rebuilding the models if and when necessary.

Moving models through obstacles unrelated to predictive modeling within an organization so they can be deployed is often an achievement in of itself; many good models are never deployed because of technical reasons related to turning models into operational decisions or because internal politics create roadblocks preventing the models from being deployed. And even when deployment is the goal of an organization, it can take weeks or months before the hurdles related to deployment are overcome. In some domain areas, government regulatory constraints present additional obstacles to deployment.

It is curious that very little information exists, in the form of general principles, on how to deploy models. Information can be gleaned in blogs and a few noteworthy books, but for the most part, analysts learn deployment by actually doing it. Most predictive analytics software vendors have developed infrastructure for how to deploy models effectively, including the development of specialized software designed specifically for the task of deploying, monitoring, and maintaining predictive models.

General Deployment Considerations

The CRISP-DM description of model deployment consists of four stages as shown in Figure 12-1: planning deployment, monitoring and maintaining models, creating a final report, and reviewing the project and generating a list of lessons learned. This chapter focuses on the first two steps: the plan and implementation of the plan.

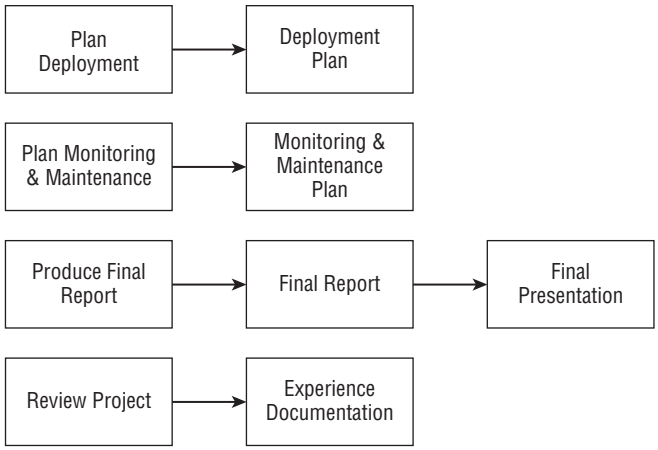


Figure 12-1: CRISM-DM deployment steps

Several general considerations should be included in the planning for model deployment. First, plan for deployment at the beginning of the project. Knowing where the models are to be deployed and how quickly they need to operate is critical in the development of data preparation and predictive models. If the model must generate predictions in real time, such as within 6 milliseconds, a 1,000-model ensemble solution may not be the best approach. On the other hand, if the model can take minutes or more before the scores are generated, the models can be very complex and still be used.

Second, predictive models themselves do not do anything except generate scores; the scores have to be interpreted and converted into actions for the model to have any effect on the organization. Some models need only a threshold applied to the scores to turn them into a rule to use. Some models, like recommendation models, cross-sell models, and upsell models could have dozens, hundreds, or even thousands of scores that must be combined in some way to generate a list of "next best offers," ranging from simple "highest probability wins" logic to very complex, multidimensional, optimized decision logic. Analysts and modelers need to coordinate with domain experts to determine the best course of action for applying model predictions to the business objectives.

Third, deployment should be monitored to ensure the models are behaving as they are expected to behave. Monitoring includes ensuring input data is consistent with the data used to build the models; if the data is changing to the point that it differs significantly from the data used to build the models, the models themselves will not necessarily behave well any longer. Monitoring also includes examining the model scores to ensure they are consistent with expected scores. When deviations from the expected occur, it is time to rebuild the models.

Deployment Steps

Deployment of predictive models includes several steps, outlined in Figure 12-2.

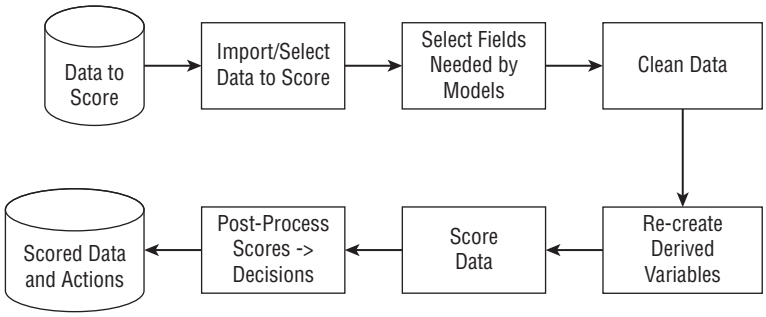


Figure 12-2: Deployment steps

First, the data needing to be scored must be pulled from the data store, whether it is a SQL or a NoSQL database. The data must be in the same format as the data used for modeling, including having the records representing the same unit of analysis and the columns representing the same information as was needed by the models.

Remember that any fields not used in the models directly or indirectly do not have to be pulled from the database. Removing processing steps for variables the modeler included while experimenting but which were not used in the final models can reduce processing time significantly. For example, if 500 variables were candidate inputs, but only 20 variables were used in the final models, 480 variables do not need to be pulled from the database; have missing values imputed; and do not need to be scaled, normalized, or otherwise transformed. Also, no target is needed to generate model scores, so the processing needed to generate the target variable for modeling does not have to be replicated in scoring.

Next, the data must be prepared in a manner consistent with the data preparation accomplished during model building, including binning, exploding

into dummies, missing value imputation, de-skewing, min-max or z-score normalization, or any other operations done to the data. Some special considerations should be remembered for data preparation, including those described in Table 12-1. Some of the considerations relate to how data is scaled or normalized in deployment compared to how they were scaled or normalized in training of the models. Usually, scaling and normalization factors applied in deployment will be the same as those created during model training.

Table 12-1: Data Preparation Considerations Prior to Model Scoring

DATA PREPARATION OPERATION	CONSIDERATIONS
Missing Value imputation based on normal or uniform distribution	Retain mean/standard deviation or min-max values found during model training rather than re-computing on the scoring population.
Missing Value imputation based on original (modeling) data	Must keep a representative sample of the variable to draw from randomly.
Min-max normalization	Keep min and max value for each variable in lookup table; if value falls outside min-max, consider clipping to min/max values in training data, treating as missing, or excluding record from scoring.
z-score normalization	Keep mean and standard deviation of each variable to be z-scored from training data.
Binning	Keep binning transformation lookup table. Be prepared to place variable values that were unseen during training into an "other" bin, treat as missing, or exclude from scoring.
Exploding into dummies	Beware of values not seen during training. They should be coded as 0 or be excluded from scoring
Log transforms, polynomial transforms, and so on	Proceed normally.
Ratio features	Beware of divide by zero.
Variable ranges	Beware of values in scoring data that exceed minimums and maximums of any data seen during modeling, or categorical variable values that were not present during training.

Where Deployment Occurs

Scoring can take place anywhere the organization decides is best for generating the scores. The Business Understanding stage should describe where deployment

takes place and how quickly scores are needed to make the models operational. Model scores are generated primarily in the predictive analytics software, on a server running predictive analytics software deployment solutions, on a local server as a standalone process, in a database, or in the cloud. A summary of the pros and cons of these methods appears in Table 12-2.

Table 12-2: Summary of Deployment Location Options

DEPLOYMENT METHOD	WHEN TO USE IT	BENEFITS	DRAWBACKS
In predictive analytics software	When scores are not time-critical or the software provides easy deployment steps	All data prep already done; no model translation steps needed	Often the slowest of methods; usually requires an analyst to initiate scoring manually
In a predictive analytics software deployment solution	For larger projects that justify additional expense; for larger organizations with many models to deploy	Tight integration with modeling software; often very efficient; often contains model monitoring and maintenance options	Often an expensive add-on; sometimes very complex software requiring additional training
Headless predictive analytics software (no GUI), batch processing	When you need automation of model deployment	All data prep already done; no model translation steps needed; automated deployment	Errors in running software can stop scoring from completing (must make modeling steps "fool proof").
In-database	When you need tight integration with operational system, real-time or near real-time scoring, or large data	Fast deployment	Must translate data preparation and models to a form the database can use (if data preparation and modeling were done outside of the database).
Cloud	When you need to scale to bigger data on an occasional or regular basis	Easy scaling of hardware; can be cost effective in comparison to buying software and hardware resources	Data prep, when not encoded in PMML, must be translated; bandwidth limited by Internet connections.

Deployment in the Predictive Modeling Software

Deployment in the predictive modeling software is perhaps the most common method of deployment. It is by far the simplest of the deployment methods because if the models were built in the software, all of the steps the modeler did to generate models and scores already exist and can be resused without modification. When models are scored only occasionally or on an ad hoc basis, this method is effective because no additional software or hardware resources are needed.

However, deploying models in the predictive analytics software is a manual process; the analyst has to launch the software, load the data, and export the scores to another location, often a database.

Deployment in the Predictive Modeling Software Deployment Add-On

Most predictive modeling software has solutions for model deployment, usually sold as an add-on to the modeling software. While these add-ons can be expensive, they integrate tightly with the modeling software, usually making the transition from modeling to deployment easy for the analyst. Moreover, the deployment options usually contain additional features to help the organization manage, monitor, and maintain models, saving the analyst from having to build custom software to accomplish all of these tasks.

Some features typically included in these software add-ons include support for alerting decision-makers when models have been run, what errors and anomalies were discovered in the data or model scores, and when the models need to be refreshed and rebuilt.

Deployment Using "Headless" Predictive Modeling Software

Some PA software allows the analyst to run a model deployment script without launching the software's GUI, enabling the analyst to automate when the software is run, alleviating the analyst from having to run the models manually. No additional data preparation or translation of the models needs to be done because the same software used to build the models is also used to deploy the models.

Deployment In-Database

Deployment done in-database is an excellent option for real-time decisions, when the volume of data is too large to make extracting scoring data from a database practical, or when the organization is database-centric and prefers to maintain a database solution. Examples of situations that call for real-time decisions include digital models selecting display ads, rendering content on

websites based on visitor behavior, call center cross-sell and up-sell recommendations, health insurance claims fraud, and credit card fraud alerts.

To run the models in a SQL or NoSQL database, all of the processing steps done to the data in the predictive analytics software must be translated into a language the database can use. This includes all data preparation operations (missing value imputation, scaling, normalization, and so on) as well as the models. Most predictive analytics software will translate models into the Predictive Model Markup Language (PMML), and some will also translate models into other languages such as C, C#, Java, SAS, or even SQL.

However, data preparation steps are not usually translated by modeling software, leaving the analyst to convert these steps manually into code that can be used in the database. Great care must be taken to ensure the data preparation steps are done correctly and completely so the models behave properly.

Deployment in the Cloud

Cloud deployment is the newest of the deployment options in the analyst's toolbox. Deployment can be done using a service company to consume the data preparation and modeling code, most often PMML, or the organization can create processing steps much like the headless deployment option, except that the hardware the models run on is in the cloud rather than a server within the organization.

Obviously, the greatest advantage is scalability; as the number of records to be scored increases, more cloud resources can be purchased to keep with the scoring needs. For retail organizations, scaling on Black Friday, for example, can be done in the cloud with short-term expansion of resources rather than having to purchase hardware and software to handle the worst-case scenarios.

Encoding Models into Other Languages

Models deployed in the predictive analytics software can be used in the format native to the software itself. However, when models are deployed outside of the predictive modeling environment, they have to be translated into another form that other environments can consume. The most commonly used languages for encoding models are PMML, SQL, C#, Java, and SAS. But depending on the needs of an organization and the expertise of analysts and programmers within the organization, other languages are sometimes used as well, including Python, .NET, or even NoSQL scripting languages.

Figure 12-3 shows a simple decision tree to predict target variable TARGET_B from one input variable, LASTGIFT. Decision trees are easy models to convert to rules and SQL because of their simplicity. The following Java code for this tree includes a condition to apply if LASTGIFT is not defined:

```
if ( LASTGIFT > 14.030 && LASTGIFT > 17.015 ) return(3.686);
if ( LASTGIFT > 14.030 && LASTGIFT <= 17.015 ) return(4.434);
if ( LASTGIFT <= 14.030 && LASTGIFT > 8.250 ) return(6.111);
if ( LASTGIFT <= 14.030 && LASTGIFT <= 8.250 ) return(8.275);
if ( LASTGIFT == null ) return(-1);
```

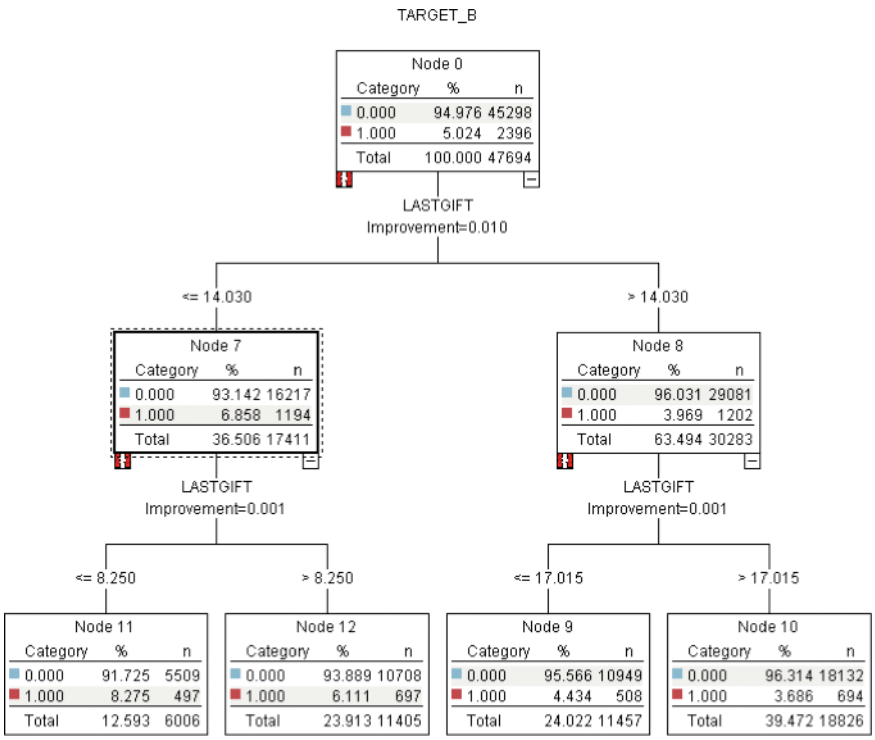


Figure 12-3: Simple decision tree to be deployed

A widely supported language for encoding models is PMML, an XML document specifically designed to support predictive modeling data preparation, models, and scoring. PMML has been defined since 1997 and continues to be supported by nearly every predictive modeling software package in one form or another. Most data preparation steps are supported by PMML, though few predictive modeling software packages support the creation or use of PMML code for data preparation. However, most tools support encoding models into PMML. One simple example of a model in PMML is the logistic regression model shown in the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML version="4.1" xmlns="http://www.dmg.org/PMML-4_1">
<DataDictionary numberOfFields="2">
```

```
    <DataField dataType="string" name="TARGET_B" optype="categorical">
      <Value value="1"/>
      <Value value="0"/>
    </DataField>
    <DataField dataType="integer" name="LASTGIFT" optype="continuous">
      <Interval closure="closedClosed" leftMargin="0.0"
                                       rightMargin="1000.0"/>
    </DataField>
  </DataDictionary>
  <GeneralRegressionModel modelType="multinomialLogistic"
              functionName="classification"
              algorithmName="LogisticRegression"
              modelName=" TARGET_B Logistic Regression">
    <MiningSchema>
      <MiningField name="LASTGIFT" invalidValueTreatment="asIs"/>
      <MiningField name="TARGET_B" invalidValueTreatment="asIs"
                   usageType="predicted"/>
    </MiningSchema>
    <ParameterList>
      <Parameter name="p0" label="Intercept"/>
      <Parameter name="p1" label="LASTGIFT"/>
    </ParameterList>
    <FactorList/>
    <CovariateList>
      <Predictor name="LASTGIFT"/>
    </CovariateList>
    <PPMatrix>
      <PPCell value="1" predictorName="LASTGIFT" parameterName="p1"/>
    </PPMatrix>
    <ParamMatrix>
<PCell targetCategory="1" parameterName="p0" beta="-2.5401948236443457"
       df="1"/>
<PCell targetCategory="1" parameterName="p1" beta="-0.02410922320457224"
       df="1"/>
    </ParamMatrix>
  </GeneralRegressionModel>
</PMML>
```

Comparing the Costs

Which of these methods is most cost effective is specific to an organization. Cost tradeoffs depend not only on hardware and software license costs for model deployment but also on human resources needed to build and support infrastructure around each of these methods. For example, deploying models in the predictive analytics software may not require any additional software costs, but will require an analyst to be dedicated to running the models each time scores are needed. The analyst will also have to develop custom reports that describe the model scores to the extent the organization needs them.

On the other hand, purchasing software add-ons to score and manage models may seem like a one-time, up-front purchase. However, they often require maintenance and monitoring themselves by an analyst. The human resource cost depends on the organization's in-house capabilities and the complexity of the software.

Post-Processing Model Scores

The last step in the deployment process is post-processing the models scores. Predictive models create scores that, in of themselves, don't do anything; they must be acted on for the scores to be useful. The way they are used is defined as business objectives created during the Business Understanding stage of the project.

Several options for post-processing scores can be specified during Business Understanding, including the following:

- Threshold the score by value into two or more groups: two groups ("select" and "not select"), three groups ("low," "medium," and "high"), or more. These groups are sometimes described as segments. Some typical methods for building the segments is described in more detail in the section that follows.
- Combine segments with suppression rules to remove records from treatment, or business rules to tune the segments based on non-analytic criteria. The business rules add context to segments determined by the scores alone and add business value that the scores did not include.
- Route segments to other humans for further review.
- Incorporate model scores into an additional analytic system to combine the scores and other rules to improve decisions. This is often an optimization system that uses the scores along with other costs and constraints before making a final decision. For example, in claims fraud, the model score may be one component along with additional business rules and rules derived from regulatory constraints, all combined to generate the next best case to investigate.

After applying one or more of these post-processing strategies, decisions are generated that result in an action: for example, a triggered contact to the customer, an investigation by an agent (a phone call, a letter, or a visit, depending on the model score and other rules), or even the routing of the transaction to another model.

Creating Score Segments

Some models are applied to the entire population of records; every record is scored and acted on. Cross-sell and up-sell models in a call center are examples of this: Every customer is scored and product recommendations are made.

However, if the predictive model is used to select a population to act on, and by extension, select another population to not act on, what threshold should be used to decide which to select? Generally, there are two approaches to selecting the population: Pick the population that exceeds a model score directly, or pick the population based on cumulative statistics of the rank-ordered population.

Picking the "Select" Population Directly from Model Score

Some models generate predictions that are immediately interpretable and directly usable. For example, the model may predict the actual probability an outcome will occur, and the business objective intends to select every customer that has more than an *x* percent likelihood of responding. Or the model may predict the customer lifetime value (CLV), and if it is accurate, the action prescribed by the business objective may be to contact all customers with CLV > $1,000. In these cases, deployment consists of using the model score directly to select the records to be included in the population to treat.

Figure 12-4 shows one such case. For this histogram, the scores are segmented into two groups: a select segment and a non-select segment. The upper bound of Bin04 (and lower bound of Bin05) is the score 0.0535—the threshold that was selected to divide the scores into the two groups. Once these labels are applied, one of the prescribed post-processing strategies then takes place: direct action of the select group, combining the select group with other business rules before making a decision on the best action to take, or incorporating the select and non-select groups into an optimization procedure.

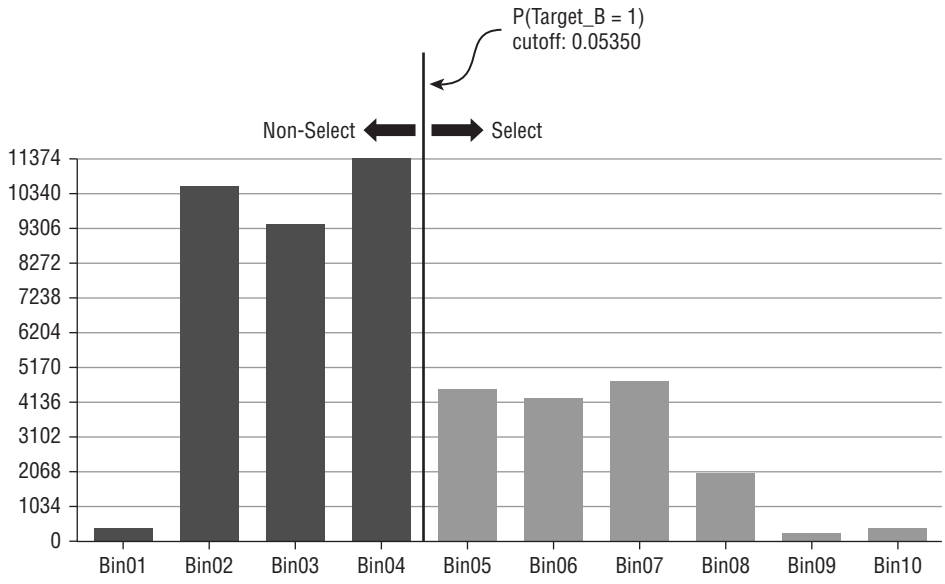


Figure 12-4: Histogram with a score cutoff

Picking the "Select" Population Based on Rank-Ordered Statistics

Often, however, the selection is not based on a specific score, but rather the rank of the score or a cumulative statistic. The most common rank-ordered methods measure gain and lift, and therefore these will be described in more detail here. However, any rank-ordered method can apply the same steps to identify a threshold for creating the select and non-select populations. Two other common rank-ordered metrics are profit charts and ROC curves.

As an example, consider models built from the KDD Cup 1998 data set. The y-axis shows the gain found in the model, meaning the percentage of lapsed donors found using the model after rank-ordering by the model score. The upper horizontal line is drawn to show where the 80 percent gain appears in the gains curve. If the model were to be deployed so that the expectation is that 80 percent of the lapsed donors would be found by the model, you need to determine the model score to use as the cut point or threshold. Figure 12-5 shows a gains chart depicting this scenario.

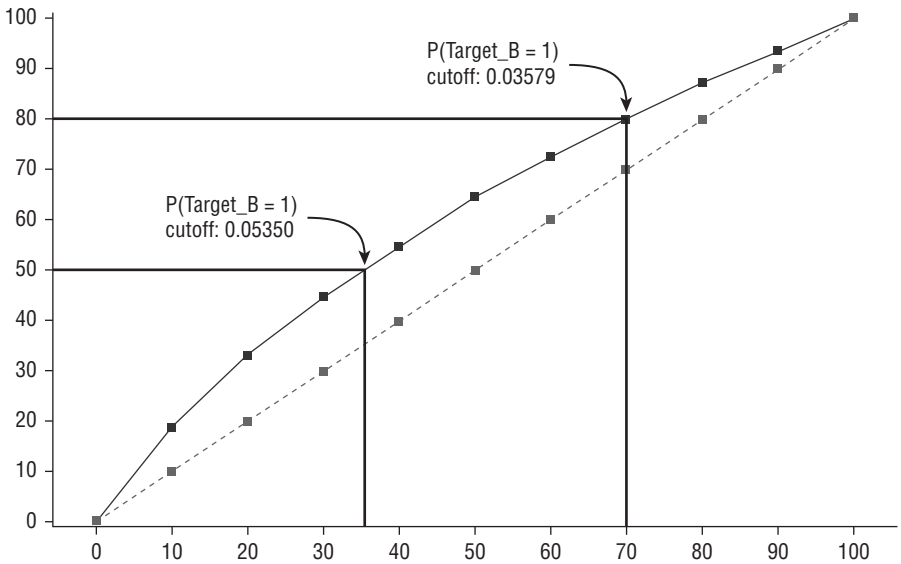


Figure 12-5: Gains chart used for creating a select population

Table 12-3 shows what the data looks like as this decision is made. The data is sorted by model score, from highest score to lowest score (descending).

The file depth in number of records and percentage of records in the validation data is shown in the table as well. If the criterion is to select a population such that you expect to find 80 percent of lapsed donor responders, you need to find the record where the 80 percent gain percentage appears. The model score associated with that record is the model score threshold. In this case, that score is 0.03579; whenever the model score exceeds 0.03579, the record is a select, otherwise it is a non-select.

Table 12-3: Model Scores Corresponding to Gain Percent and Lift

MODEL SCORE	GAIN PERCENT	LIFT	VALIDATION DATA FILE DEPTH	VALIDATION DATA FILE DEPTH PERCENTAGE
0.10661	0.00	0.0	1	0.002
0.10661	0.00	0.0	2	0.004
0.10661	0.00	0.0	3	0.006
...	...	...	...	...
0.03579	79.96	1.1416	33,413	70.039
0.03579	79.96	1.1416	33,414	70.042
0.03579	79.96	1.1416	33,415	70.044
0.03579	79.96	1.1415	33,416	70.046
0.03579	80.00	1.1421	33,417	70.048
0.03579	80.04	1.1426	33,418	70.050

If the threshold is to be set so that only 50 percent of the recoverable lapsed donors are found rather than 80 percent, the model score cut point is now 0.0535 as shown in Figure 12-5. Note that the depth of lapsed donors mailed to in this scenario is only about 35 percent of the population (the x-axis value for 50 percent gain).

If the criterion is to select all lapsed donors such that the expected model lift is greater than or equal to 1.5, Figure 12-6 shows the lift chart for the model. The model score corresponding to the lift value is equal to 1.5 0.06089. Finally, if the criterion is to select all lapsed donors such that the expected response rate of the model is greater than or equal to 7.5 percent, a model score cutoff value of 0.05881 is to be used, as shown in Figure 12-7.

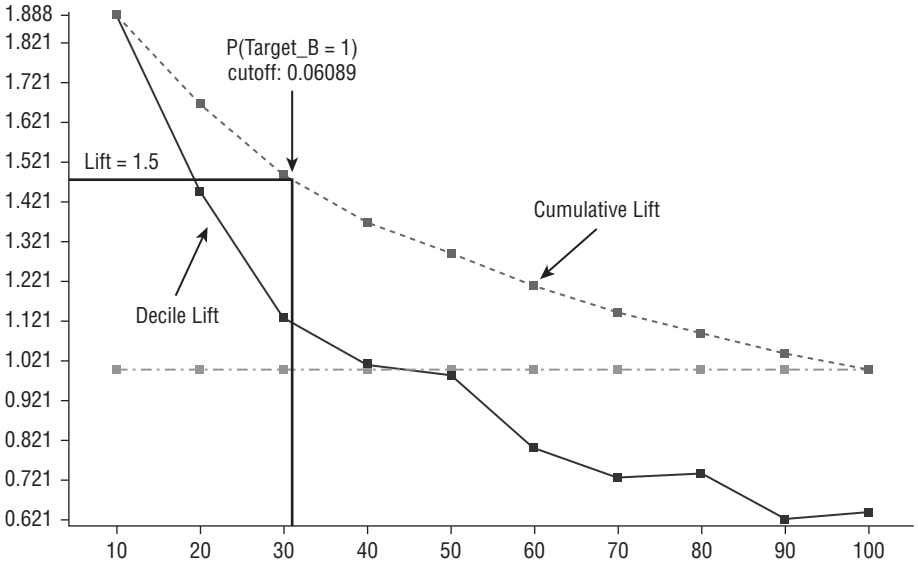


Figure 12-6: Lift equal to 1.5 as the metric for selecting the population to contact

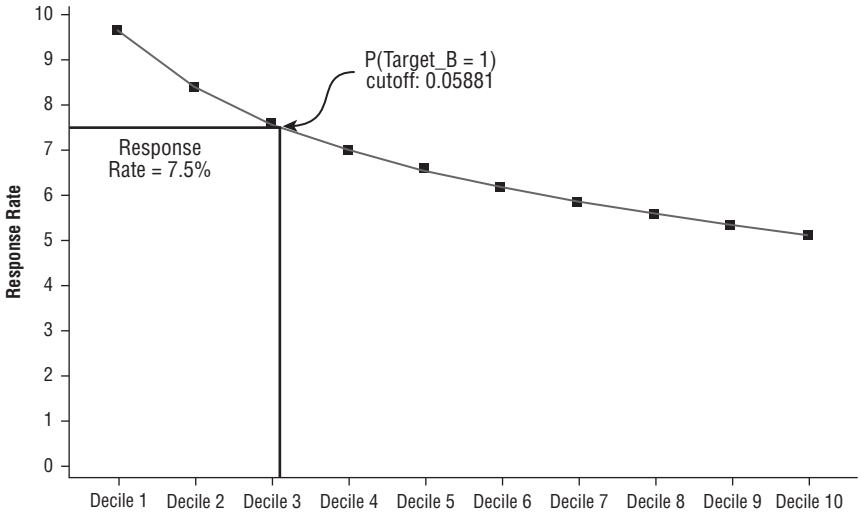


Figure 12-7: Response Rate equal to 7.5 percent as the metric for selecting the population to contact

When Should Models Be Rebuilt?

Even the most accurate and effective models don't stay effective indefinitely. Changes in behavior due to new trends, fads, incentives, or disincentives should be expected. Common reasons for models to lose effectiveness include:

- **Fraud models:** If patterns related to fraud found by models result in the identification of perpetrators, the pool of those using these patterns will decrease as a result of the models, forcing a change in tactics and behavior for those who wish to perpetrate fraud.
- **Customer acquisition models:** Some campaigns target products that have a limited supply or tap into a short-term preference or fad. Once individuals have purchased the product, or once the product has had its run, response rates will decrease.
- **Manufacturing defect identification:** Once defects have been corrected and problems leading to the defect have been corrected, fewer cases will result in high model scores.

Nevertheless, uncovering the reason for changes in behavior isn't needed to identify that the models need to be updated; one only needs to measure whether or not model accuracy has worsened. The most common way organizations measure model degradation is by examining average model performance over regular time intervals.

The rest of this section describes one approach to determine when models should be rebuilt based on model effectiveness. Our discussion will primarily refer to the KDD Cup 1998 data—the lapsed donor problem—but can be extended to any other type of classification or regression problem.

For example, you can measure the response rate for the population the model selected on a weekly basis, such as what is shown in Figure 12-8. In this figure, in the first four weeks the model was used, an average response rate of 7.5 percent was observed for the lapsed donors selected by the model. However, beginning in week 5, the rate began to decline to the point that after 12 weeks the rate had settled at 6 percent.

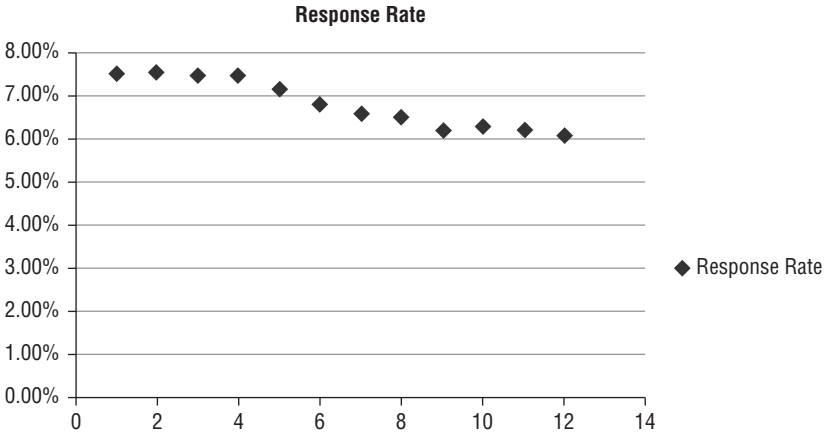


Figure 12-8: Decaying response rate from model

How much reduction in model performance is too much? Or, stated another way, is a reduction from 7.5 to 7.3 percent a *significant* reduction in performance, indicating the model needs to be rebuilt? The answer requires consideration of two additional pieces of information. First, recall that any measure of amounts or rates from a sample is an estimate with some uncertainty: The response rate 7.5 percent is actually 7.5 percent plus or minus some amount. The error depends on the size of the sample.

For two-valued categorical target variables (binomial or binary variables), a binomial calculator can identify the size of a population needed to identify a difference in response rates that is of interest to an organization operationally.

Without addressing the theoretic justification for the formula, the assumptions behind the formula, or the specific formula, you can compute the sample size needed if you know the proportion (response rate for the lapsed donor problem), the error that is acceptable or observed, and your desired confidence that the estimate is within the error bounds . Most online calculators begin with sample size, proportion, and confidence, and return the confidence interval, but this is reversed easily by solving for sample size from the error rather than solving for error from the sample size.

Table 12-4 shows some sample size values for the 7.5 percent response rate. The interpretation of the first row of the table in the context of model deployment of the lapsed donor model is as follows. Suppose you are expecting a 7.5 percent response rate from the campaign based on modeling data. If the campaign contacted approximately 29,000 individuals, then there is approximately an 80 percent confidence that the observed response rate will fall between 7.3 and 7.7 percent, as shown in the third row. So if we return to the question "Is the reduction from 7.5 to 7.3 percent a significant reduction in response rate?"

we have a complicated answer: According to the table, we expected a 7.3 percent response rate to be possible, but only with an 80 percent confidence, so the 7.3 response rate is on the edge of our expectations.

If, on the other hand, the response rate was 6.5 percent, as it was 8 weeks after the model was deployed (see Figure 12-8), the fourth row of Table 12-4 shows that for 28,862 records, this is outside the range 7.1 to 7.9 percent at the 99 percent confidence level, so it is very unlikely that a 6.5 percent response rate would have been observed by chance; at this point, once the response rate dips below 7.1 percent, the model should be rebuilt based on the statistical significance test.

Table 12-4: Binomial Significance

PERCENT RESPONSE RATE (PROPORTION)	+/− PERCENT ERROR	EXPECTED RESPONSE RATE RANGE (PERCENT)	PERCENT CONFIDENCE	SAMPLE SIZE NEEDED FOR RANGE AND CONFIDENCE
7.5	0.2	7.3–7.7	99	115,447
7.5	0.2	7.3–7.7	95	66,628
7.5	0.2	7.3–7.7	80	28,416
7.5	0.4	7.1–7.9	99	28,862
7.5	0.4	7.1–7.9	95	16,657
7.5	0.4	7.1–7.9	80	7,104
7.5	0.1	6.0–8.5	99	461,788
7.5	0.1	6.0–8.5	95	266,511
7.5	0.1	6.0–8.5	80	113,664

If the target variable is instead continuous, you could apply a difference of means test instead as measured by the *F*-statistic to provide an indication if the outcome of the response from the model is different from what is expected.

These kinds of tests—the binomial test and the F test—assume distributions that may or may not be observed in the data. They are useful barometers for predictive modelers to use to gauge if differences in rates or mean values are truly different, though modelers should be careful to not interpret them too strictly without a deeper understanding of statistics, the assumptions, and the meaning of the tests.

Model Assessment by Comparing Observed vs. Expected

So far, the observed results after the model is deployed have been compared with the expected rates or mean values based on what was observed during modeling,

and in particular, based on the validation data. However, model deployment does not necessarily apply to a representative sample. For example, the model deployment could be done for records from just one geographical region or a particular segment of the population being used for a custom deployment. In these cases, the observed response rates may be much lower or higher than the average value found in validation data, not because the models aren't working properly, but rather because the input data itself is different than the full population used in modeling, and therefore response rates for the particular subpopulation are expected to be different.

Fortunately, you know exactly what the model performance should be. Each record generates a score, and you already know how the scores relate to the metric being used: You identified this relationship during the Model Evaluation stage. You therefore know what response rate is expected from the population used for deployment. This is the better number to use as the baseline for identifying when the model is behaving differently than expected because it takes into account any population used for deployment.

Sampling Considerations for Rebuilding Models

When it is time to rebuild models, care should be taken to ensure the data used to rebuild the models is representative of the entire population the models are to be applied to, not just the recently treated population. Supervised learning requires a target variable—a measure of what happens when a treatment occurs, whether that treatment is a display ad, an e-mail, an audit, or some other action.

When a model is deployed and its scores are used to select a sub-population of records, only the sub-population that has been selected will generate responses; the untreated population, obviously, cannot and will not generate responses.

Consider again the KDD Cup 1998 data set. Assume that the select criterion chose all lapsed donors with an expected response rate greater than or equal to 7.5 percent. This translated into selecting all donors whose model score exceeded a cut point of 0.05881 (from Figure 12-7). Now assume that the non-profit organization has a population of 96,367 lapsed donors to score, and of these, 29,769 lapsed donors have scores that exceed the threshold score of 0.05881 and will therefore be selected to be contacted. After the contact has occurred, you will have 29,769 new outcomes: the response of the lapsed donors to the campaign. Each of these 29,769 lapsed donors will have either responded or not responded to the campaign, giving you a new target value for each. This new value can be used for future modeling in the same way TARGET_B was used in the original model.

However, the 66,598 lapsed donors who weren't contacted have no outcome associated with them; you don't know whether or not they would have responded had they received the contact.

The first five rows of Table 12-5 show a summary of the populations after applying the model, including the selects and non-selects if a threshold is used. Of course, as the table shows, our expectation is that the response rate of the non-selected population will be only 3.97 percent, just over half the rate of the population selected by the mode, but we can't verify this unless the 66,598 individuals are contacted. The last three rows of Table 12-5 show the mailing counts and expected response rate when we incorporate the strategy that adds a random subset of non-selects to the mailing.

Table 12-5: Expected count and response rates from the sampling strategy

MEASURE	SELECT	NON-SELECT	MAILING
# Lapsed donors	29,769	66,598	NA
Min score	0.05881	0.0208	NA
Max score	0.1066	0.05881	NA
Expected response rate	7.50%	3.97%	NA
Number of expected responders (if everyone mailed)	2,232	2,641	NA
Mail sample (selects and random non-selects)	29,769	2,556	32,325
Expected responders	2,232	101	2,333
Expected rate: Select + non-select	7.5%	3.97%	7.2%

On the surface, mailing to those who are unlikely to respond is counter-intuitive and an unnecessary cost. However, there is a different cost associated with forgoing this additional mailing cost when it is time to rebuild models. There are only a few possible behavioral differences that could cause the lower performance in model scores that lead to rebuilding the models. It could be that the entire population (select and non-select) responds at a lower rate due to a fundamental behavioral shift. However, it could also be that the entire population would have responded at approximately the same rate, but *who* responds has shifted. Some who were not likely to respond before have now become likely to respond and some who were likely to respond before have now become unlikely to respond.

However, you can't identify any of these effects unless you measure responses from the entire population; to understand the shifts in behavior you need to contact those in the non-select population. But isn't this the very reason you built models in the first place, so you wouldn't need to contact the entire population? The answer is "yes," but there is an alternative. If you only contact a *subset* of the non-select population that is large enough to measure the response of

non-responders but small enough that it doesn't influence the cost significantly for the organization, you can satisfy both objectives.

The size of the sample needed can be computed in the same way you computed the sample size. Table 12-6 shows the tradeoffs you can make to ensure the sample size is big enough to avoid clashing with the 7.1 percent response rate from the select population. To identify the difference between a cumulative response of 7.1 percent (the lower bound expected from the model according to Table 12-4 if you use the 99 percent confidence level) and 3.97 percent, you only need 266 records. However, this sample size is too small to characterize the non-selects well; only 10 responses would be expected from the non-select group from the sample size of 266. A better sample size is 2,000, using the rule-of-thumb from Chapter 4, which considers the curse of dimensionality.

If 2,556 non-selects are included in the population to contact, then a total of 32,325 (29,769 + 2,556) lapsed donors will be mailed. The expected response rate for the total population has now dropped from 7.5 percent to 7.2 percent, but you gain two important pieces of information:

- You can now compare the response rate from the select population to the sample of the non-select population and ensure there is a significant difference in response rates.
- You now have a significantly sized population of non-selects to include in the training data for new models when they have to be rebuilt.

Table 12-6: Sample Sizes for Non-Select Population

PERCENT EXPECTED RESPONSE RATE	PERCENT CONFIDENCE	PERCENT ERROR	MAXIMUM EXPECTED RESPONSE RATE (PERCENT)	NUMBER OF SAMPLES NEEDED
4	99	1	5	2,556
4	99	1.5	5.5	1,113
4	99	2	6	639
4	99	3.1	7.1	266

What Is Champion-Challenger?

The champion model is the model being deployed currently. We have already described why models need to be rebuilt and what data to use to rebuild models. However, rather than waiting until a model needs to be rebuilt before building a new one, the champion-challenger approach builds a challenger model as often as

new data is available to improve the existing model. Once the challenger model performs better than the champion model, the challenger model replaces the champion model and the process of building a new challenger model begins.

Building the challenger model is usually an automated background process rather than a manual, analyst-driven process. However, any time models are built automatically, care must be taken to ensure the new model is not only more accurate, but also appears to be stable and continues to make sense. In other words, the analyst should still ensure the new model is acceptable before deploying the challenger model.

Consider Figure 12-9. The champion model is the same one shown in Figure 12-8, which shows the declining response rate. Each new challenger model includes data used in building the initial models, but is augmented by the new data collected from the treatments from weeks 1 and 2 (the two left-most data points in the figure). Its performance is remaining stable, in part because it is incorporating new data the original model had not seen. The proportion of data to use from the initial modeling data compared to the new data depends on how much new data is available from weeks 1 and 2; if there is sufficient data to build stable models, only newer data can be used. This approach is good to identify new behavioral patterns that emerge over time. If insufficient data is available from weeks 1 and 2, the data can be augmented with older data.

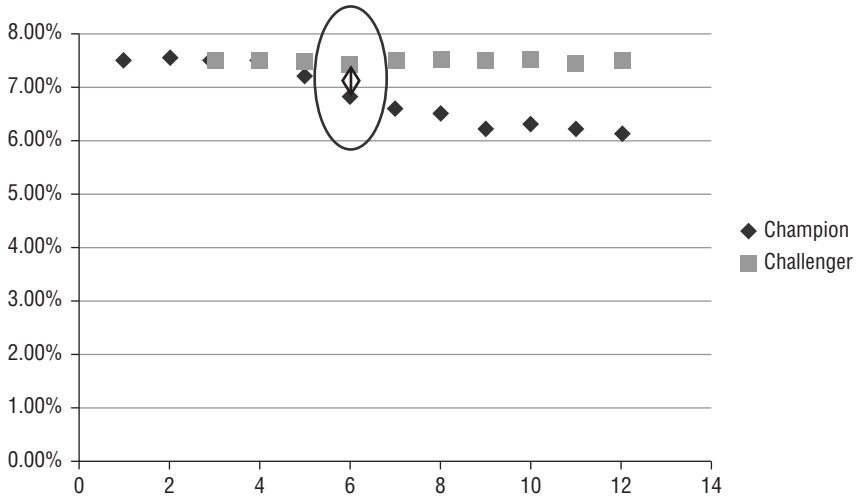


Figure 12-9: Champion vs. Challenger average response rate

Assessing the accuracy of the champion model is straightforward for the lapsed donor problem: Compute the response rate for the lapsed donors selected to treat. How is the challenger model assessed? After all, this model isn't being deployed.

The simplest method is to compare the accuracy of the challenger model on its own validation data with the accuracy of the champion model on its own validation data. The two values can then be tested for significance; when the challenger model accuracy on validation data is significantly better than the champion model accuracy on its validation data, replace the champion model with the challenger model.

However, if the characteristics of the model inputs are changing, comparisons between the two models are not based on the same data and therefore could be misleading.

One approach compares the two models based on identical data, taking advantage of the similarity in the population of records the models select. In this approach, let's compare the accuracy of the two models using only records that were selected by the champion model and would have been selected by the challenger model. The champion model selects were treated—they were contacted in the lapsed donor model example—so you have actual responses for these records. The model with the higher response rate indicates a better model.

Figure 12-10 shows this approach visually. The champion model and challenger model were scored on 25,500 records. The number of records the two models have in common is 20,000, and therefore the response rates on the actual treatment will be computed based on the 20,000 records the two models have in common.

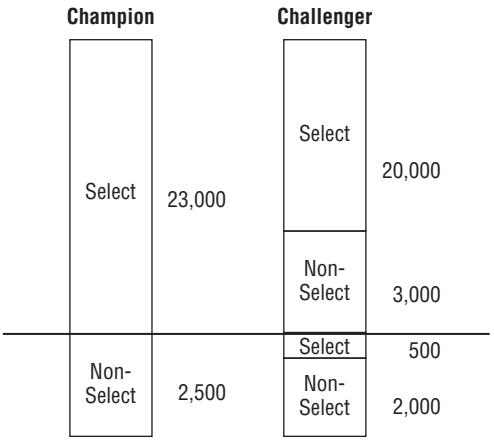


Figure 12-10: Champion-Challenger sampling

The determination of when the challenger model is better than the champion model, or more precisely, how much better the challenger model needs to be before it is considered definitively better can be made in the same way the determination was made whether or not a new model is needed: When the difference between the champion model accuracy and the challenger model

accuracy is unlikely to have occurred by chance as measured by the binomial significance test or an F test, the challenger model is considered better and will replace the champion model for deployment.

Summary

Deployment of predictive models is the most underappreciated stage of the CRISP-DM process. Planning for deployment begins during Business Understanding and should incorporate not only how to generate model scores, but also how to convert the scores to decisions, and how to incorporate the decisions into an operational system.

Greatly increased efficiency in generating model scores can be achieved by eliminating processing no longer needed to generate the model scores and by pushing as much of the data preparation steps up to the database as possible.

Finally, a good deployment system plan recognizes that models are not static: They were built from data that represented the history at a particular moment. But that moment will ultimately pass. Models should be monitored and ultimately be replaced by an improved model when needed.

CHAPTER

13

Case Studies

These case studies are included to describe real projects that use principles described in this book to create predictive models. Throughout any project, decisions are made at key points of the analysis that influence the quality of the final solution; this is the *art* of predictive modeling. Rarely does an analyst have time to consider all of the potential approaches to a solution, and therefore decisions must be made during Data Preparation, Modeling, and Model Evaluation.

The case studies should be used as motivational rather than as recipes for predictive modeling. The analyses in these case studies don't present perfect solutions, but they were successful. The survey analysis case study had the luxury of trying two approaches, each with pros and cons. The help desk case study succeeded because of the perseverance of the analysts after the first modeling approach failed. In both cases, the final solution used the science of predictive analytics plus creative twists that were unconventional, but productive.

Survey Analysis Case Study: Overview

This case study describes a survey analysis project for the YMCA, a cause-driven charity whose core mission is to improve the lives of its members and build communities. The YMCA achieves these objectives primarily through facilities and programs that promote the physical well-being of its members.

The YMCA expends considerable time and effort to understand how to help members achieve their fitness goals and build community. These analyses are grounded in sound social science and must be diagnostic and predictive so they are understandable by decision makers. Moreover, decisions are made at the branch level—the individual YMCA facility.

One source of data for achieving the analytics objectives is the annual member survey. The YMCA, as a national organization with more than 2,500 branches across the country, developed a member survey for use by its branches. Tens of thousands of these surveys are completed each year.

SEER Analytics (`http://www.seeranalytics.com`) was, and continues to be, the leader in analyzing YMCA surveys and creating actionable insights based on the analyses. I was a subcontractor to Seer Analytics for the work described in the case study.

Business Understanding: Defining the Problem

For the project, 32,811 surveys with responses were made available for the year 2001. Modeling data from the survey contained 48 multiple choice questions coded with values between 1 and 5, where 1 was the most positive response and 5 the most negative. Questions were primarily attitudinal, related to the member's experience with the Y, though four questions were demographic variables. There were two free-form text questions and two related fields that categorized the text into buckets, but these were not used in the analysis.

Throughout this case study, questions will be identified by a Q followed by the question number, so Q1 for question 1. Sometimes an explanation of the key idea of the question will be provided in parentheses, such as Q1 (satisfaction), indicating that the key idea in question 1 is member satisfaction.

Defining the Target Variable

Three questions in the survey were identified as key questions related to attitudes and behaviors in members that are indicative of how well the Y is meeting the needs and expectations of the members. These three questions were Q1, Q32, and Q48, with the full text of the questions shown in Table 13-1.

Table 13-1: Three Questions for Target Variables

QUESTION	FULL QUESTION	SUMMARY OF IDEA REPRESENTED IN QUESTION	PERCENT WITH RESPONSE = 1
Q1	Overall, how would you rate the [Branch Name] YMCA?	Satisfaction	31%
Q32	All things considered, do you think you will belong to this Club a year from now?	Intend to renew	46%
Q48	Would you recommend the Club to your friends?	Recommend to a friend	54%

The three target variables had high association with each other. Of the 31 percent of Q1 = 1 responders, 86 percent of them also had Q48 = 1, a lift of 1.6 over the 54 percent baseline for Q48 = 1. Conversely, of the 54 percent with Q48 = 1, 49 percent have Q1 = 1, a lift of 1.6 as well.

Because of this phenomenon and the desire to create a single model to characterize members with a highly positive opinion of their Y branch, we created a new derived target variable: a simple linear combination of the three questions, called the *Index of Excellence* (IOE). In order to have larger values of IOE considered better than smaller values and have its maximum value equal to 10, we reversed the scale by subtracting the sum from the maximum possible value (15), or

$$\text{IOE} = 10 \times [(15 - \text{Q1} - \text{Q32} - \text{Q48}) \div 12]$$

The three target variables are all skewed toward the lower end of their respective distributions, and therefore IOE, with its distribution reversed, was skewed toward the upper end of its range; the vast majority of values exceed 7, as can be seen in Figure 13-1.

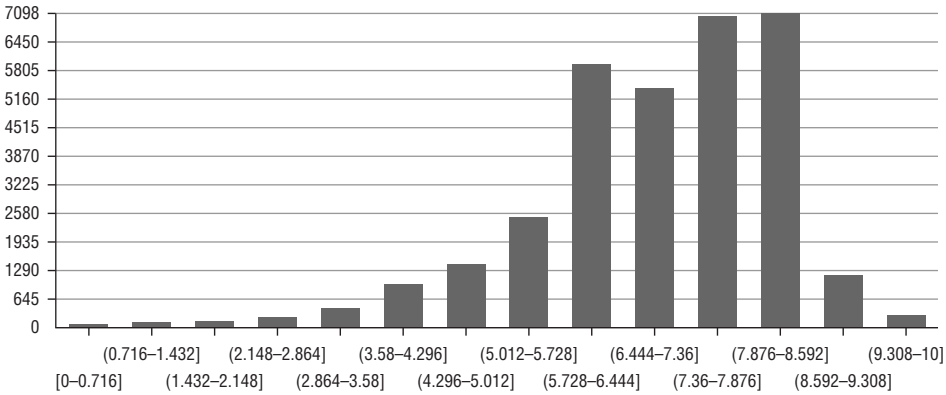


Figure 13-1: Index of Excellence distribution

Data Understanding

The candidate input questions are shown in Table 13-2, along with their category labels as determined by the analysts.

Table 13-2: Questions and Categories for Model Inputs and Outputs

CATEGORY	QUESTIONS
Staff	Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12
Building	Q13, Q14, Q15, Q16, Q17, Q42
Equipment	Q18, Q19, Q20, Q37, Q38, Q39, Q40, Q41
Programs	Q21, Q23, Q43
Value	Q22
Goals	Q44
Relationships	Q24, Q25, Q26, Q27
Other	Q28, Q29, Q30, Q31, Q45, Q46, Q47
Targets	Q1, Q32, Q48
Text	Q33, Q34, Q35, Q36
Membership	Q49, Q50, Q51
Demographics	Q52, Q53, Q54, Q55

First, two of the quirks and potential problems in the data are as follows:

- In the full data set, 232 surveys had no responses (0.7 percent of the total survey population); all the response values were zero. However, these

232 surveys had text comments, indicating the members still wanted to communicate a viewpoint. These surveys were included in the analysis (an oversight), though had little effect on the patterns found.

- One question that arose was whether there was a significant number of members who merely checked the same box for all the questions: all 1s, all 2s, all 3s, and so on. The good news was that the worst instance of this pattern was in 274 surveys that had all responses equal to 1 (only 0.8 percent of the population), more than the number of surveys with all 2s (58), all 3s (108). Therefore, the responses to the questions were believed to be an accurate reflection of the true member response.

Because neither of these problems cause significant numerical issues, no corrective action was taken.

Data Preparation

Very little data preparation was needed for the data set. The responses were coded with values 0 through 5, so there were no outliers. There were few NULL values. Any NULL values were recoded as 0.

The most significant decision about the data was determining how to use the questions in the algorithms. The data was ordinal and not truly continuous. However, for regression models, it was far more convenient to use the data as continuous because in this case, there is only one column in the data per question. If the data were assumed to be categorical, the questions would be exploded to dummy variables, leading to up to five or six columns per question if the zeros were included as dummy columns.

Missing Data Imputation

The next most significant issue was the problem with responses coded with 0—the code for no response. If the data were treated as categorical, these could be left as the value 0 and treated as just another level. However, if the variables are treated as continuous or ordinal, leaving these as 0 communicates to the algorithms that 0 is the best answer of all (smaller than the "top" response value of 0). These values therefore were similar in function to missing values and needed to be corrected in some way.

On the surface, mean imputation seemed to be a misleading way to recode these values; coding non-responses as 3 didn't necessarily convey the relationship between the non-responses and the target variables. Instead, the approach was to identify the relationship between the 0s and the target variable first, and then recode the 0s to values that did the least harm.

Figure 13-2 shows two plots. The plots show the relationship between the mean value of the target Q1 = 1 and Q2. Q2 = 5 was omitted because for this question and most others, the counts for the worst response code, 5, was very small and did not follow the trend of the other responses. Note that the relationship between the percentage of responses for Q1 = 1 decreases monotonically with Q2. The percentage of Q1 = 1 responses for non-responsive Q2 was 16.24 percent, shown at the far left of both plots.

The plot at the right shows a linear curve fit of the data points and the plot at the left an exponential curve fit of the percentage of Q1 = 1 vs. Q2, and the resulting formulas for the curve fits are shown on the plots. The equations were solved for "x" (Q2), which yielded Q2 = 2.2 for the exponential equation and Q2 = 2.8 for the linear fit. Replacing the 0s with either of these values would bias the models less than using the mean (1.8) or the median (2). The final formula for imputation places the 0s at a value in between these, 2.5 in the case of question Q2.

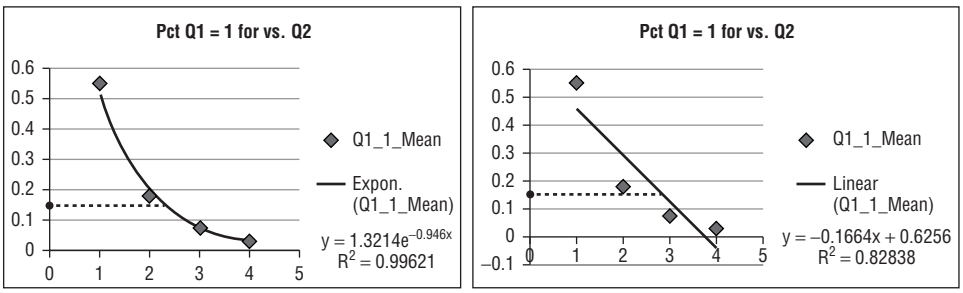


Figure 13-2: Missing value imputation

Was this cheating, to use the target variable in the imputation? The answer technically was "yes"; using the target variable in decisions for how to recode and transform variables is inherently dangerous, possibly leading to biased decisions that will increase accuracy on training data but lessen accuracy on new data. On the other hand, from a practical standpoint, the answer is clearly "no." Because the values used in imputation were nearly always 2.5, they were stable and not prone to overfitting data based on the imputation value.

This process of imputing missing values using a linear fit of a question to the target variable was repeated for every question. Very often, the value 2.5 was a good value to use like Q2, though this wasn't always the case.

Feature Creation and Selection through Factor Analysis

The questions in a survey were intended to capture attitudes of the responders. Often, multiple questions were asked to uncover an attitude or opinion in different ways, which led to a high correlation between questions. One approach often used by social scientists to uncover the underlying ideas represented by the

questions is Factor Analysis, a technique closely related to Principal Component Analysis, discussed in Chapter 6. Even though the algorithm has some difference with PCA, the use of Factor Analysis in this case study mirrors the use of PCA described in Chapter 6.

After applying Factor Analysis using default settings, we determined that the top ten factors were interesting and comprised enough of the variance to be a good cutoff. As is the usual practice, each of the factors was named according to the questions that loaded highest on the factors. The factor loadings for five of the factors are shown in Table 13-3. The highest loading questions on the factors appear in bold. The highest loading questions generally tend to be questions neighboring each other because of the way the survey was laid out, though it isn't always the case. In addition, a few questions loaded high on multiple factors, indicating a clean set of ideas identified by Factor Analysis.

Table 13-3: Factor Loadings for Five of Six Top Factors

FACTOR DESCRIPTION	STAFF CARES	FACILITIES CLEAN/SAFE	EQUIPMENT	REGISTRATION	FRIENDLY STAFF
Factor Number	**1**	**2**	**3**	**4**	**6**
Q2	0.295	0.238	0.115	**0.458**	0.380
Q3	0.217	0.143	0.093	**0.708**	0.077
Q4	0.298	0.174	0.106	**0.601**	0.266
Q5	0.442	0.198	0.087	0.173	**0.613**
Q6	0.417	0.254	0.142	0.318	**0.584**
Q7	0.406	0.277	0.167	0.252	**0.461**
Q8	**0.774**	0.058	0.041	0.093	0.113
Q9	**0.733**	0.175	0.108	0.145	0.260
Q10	**0.786**	0.139	0.079	0.110	0.218
Q11	**0.765**	0.120	0.101	0.132	0.015
Q12	**0.776**	0.090	0.049	0.087	0.014
Q13	0.145	**0.728**	0.174	0.112	0.110
Q14	0.191	**0.683**	0.163	0.151	0.124
Q15	0.102	**0.598**	0.141	0.090	0.070
Q16	0.100	0.370	0.133	0.082	0.035
Q17	0.128	**0.567**	0.229	0.102	0.080
Q18	0.148	0.449	**0.562**	0.116	0.114
Q19	0.129	0.315	**0.811**	0.101	0.103
Q20	0.171	0.250	**0.702**	0.086	0.078

Table 13-4 rearranges the information in Table 13-3, showing a summary of the factors, the top loading questions, and the percent variance explained. It is perhaps more clear from this table that questions did not overlap between factors.

Table 13-4: Summary of Factors, Questions, and Variance Explained

FACTOR	FACTOR DESCRIPTION	TOP LOADING QUESTIONS	CUMULATIVE PERCENT VARIANCE EXPLAINED
Factor 1	Staff cares	Q8, Q9, Q10, Q11, Q12	12.2
Factor 2	Facilities clean/safe	Q13, Q14, Q16	20.1
Factor 3	Equipment	Q18, Q19, Q20	25.3
Factor 4	Registration	Q3, Q4	29.7
Factor 5	Condition of locker rooms, gym, swimming pool	Q39, Q40, Q41	34.1
Factor 6	Friendly/competent staff	Q5, Q6, Q7	38.2
Factor 7	Financial assistance	Q28, Q29	42.2
Factor 8	Parking	Q16, Q42	46.0
Factor 9	Good place for families	Q26, Q30	49.1
Factor 10	Can relate to members / feel welcome	Q23, Q24, Q25	52.1

To help understand why the factors were labeled as they were, Tables 13-5, 13-6, 13-7, and 13-8 show the question numbers and corresponding descriptions of the questions that loaded highest for the top four factors.

Table 13-5: Top Loading Questions for Factor 1, Staff Cares

Q8	Know your name
Q9	Care about your well-being
Q10	Take the initiative to talk to members
Q11	Check on your progress & discuss it with you
Q12	Would notice if you stop coming

Table 13-6: Top Loading Questions for Factor 2, Facilities Clean/Safe

Q13	Overall cleanliness
Q14	Security and safety
Q16	Adequate parking

Table 13-7: Top Loading Questions for Factor 3, Equipment

Q18	Maintenance of equipment
Q19	Has the right equipment
Q20	Has enough equipment

Table 13-8: Top Loading Questions for Factor 4, Registration

Q3	Ease of program or class registration
Q4	Staff can answer questions about schedules, classes, etc.

Modeling

Sampling for building the predictive models was standard: 50 percent of the data was used for training, 50 percent for out-of-sample testing. The first models used a traditional approach of stepwise linear regression to predict IOE with a few key questions and the factor analysis projections as inputs. Some descriptions of Factor Analysis and Principal Component Analysis (like Wikipedia) describe these techniques as performing data reduction or dimensionality reduction. A more accurate description is that these techniques perform candidate input variable reduction. All 48 of the original questions are still needed to compute the factors or principal components, so all of the data is still needed even if the Factor Analysis or Principal Component Analysis reduces the questions down to ten factors like the ones shown in Table 13-4.

For this project, two questions were identified as key questions on their own: Q22 (Value for the money) and Q44 (How has the YMCA helped meet your fitness goals) based on the recommendations of domain experts and the strong predictive relationship of these questions in preliminary modeling. These were included in addition to the ten factors so that there were 12 candidate inputs. A stepwise linear regression variable selection approach was taken in building the model summarized in Table 13-9.

Table 13-9: Regression Model with Factors as Inputs

VARIABLE	VALUE	STD. ERROR	T VALUE	PR(>|T|)
(Intercept)	10.2566	0.0172	597.0967	0.0000
Q44	–0.5185	0.0074	–70.4438	0.0000
Q22	–0.4893	0.0068	–71.5139	0.0000

Continues

Table 13-9 (*continued*)

VARIABLE	VALUE	STD. ERROR	T VALUE	PR(>|T|)
Factor2	–0.2761	0.0055	–50.5849	0.0000
Factor1	–0.2397	0.0051	–47.0156	0.0000
Factor6	–0.2242	0.0056	–39.9239	0.0000
Factor9	–0.2158	0.0054	–40.0539	0.0000
Factor10	–0.1917	0.0057	–33.4452	0.0000
Factor3	–0.1512	0.0051	–29.472	0.0000
Factor4	–0.1068	0.0055	–19.2649	0.0000
Factor5	–0.0798	0.0054	–14.846	0.0000

All of the factors included in the model were significant predictors. Factors 7 and 8 were removed as a result of the stepwise procedure, as their reduction in accuracy did not justify their inclusion per the *Akaike information criterion* (AIC). However, the two key questions, Q44 and Q22, had a much larger influence on the model as evidenced by their coefficients and t values.

An alternative approach was tried as well. Rather than using the factors as inputs to the models, the question that loaded the highest on each factor was selected as the representative of the factor. This approach has an advantage over using the factors as inputs: It is more transparent.

The factors, while representing an idea, still require all of the inputs so the factors can be computed; each factor is a linear combination of all the survey questions. However, if you use just one representative question for each factor, the question that loaded the highest on the factor can be used instead of the entire factor. In this approach, rather than needing all the questions to run the model, only 12 questions (at most, if no variable selection took place) are candidate inputs. After further assessment of the data, a third key variable, Q25 (Feel welcome at the YMCA) was added, making 13 candidate inputs to the model. That model, with seven inputs found by a stepwise regression procedure, is summarized in Table 13-10. The four factors represented in the model are 1, 2, 3, and 6.

Table 13-10: Regression Model with Representative Questions as Inputs

INPUT	QUESTION	QUESTION DESCRIPTION	FACTOR
1	Q25	Feel Welcome	NA
2	Q44	Y Helps Meet Fitness Goals	NA
3	Q22	Value for the Dollar	NA
4	Q13	Facilities clean	Factor 2
5	Q18	Equipment Maintained	Factor 3

6	Q9	Staff Cares about Well-Being	Factor 1
7	Q6	Competent Staff	Factor 6

After a comparison of the two approaches, using individual questions as inputs rather than the factors, generated higher R-squared, and therefore was the model selected for use.

Model Interpretation

The regression model was not built so that the three target questions or the combined form of the three questions, IOE, could be predicted. They were built to identify member attitudes related to these target questions. The models identified the key questions (inputs) related to IOE nationally. However, the models needed to be able to inform individual branches how well they were achieving their own IOE, and in what areas they could improve their Y so that IOE could be increased.

To explain the models, Seer Analysis focused on the questions included as inputs to the models. If a YMCA branch was doing well with those questions, they were necessarily doing well on the IOE scale. The branches then could incorporate changes in their staff, programs, facilities, or equipment to improve the responses to the inputs of the model, thus increasing their IOE. The desire was to show these relationships in a concise and informative way through data visualization.

The key visual shown for each branch was like the one shown in Figure 13-3. The seven key questions found by the regression model were shown along the x axis, with the importance as found by the regression equation increasing as one goes to the right; the most important question nationally was at the far right. All of the questions were important though, not just the ones at the right. The same visualization could be created for each branch.

On the y axis, relative measures of the impact of these questions on IOE were shown. Several key ideas were incorporated in the visualization. First, the order of the balls on the x axis reflected the level of relative importance of the question in the model. Feel welcome was the most important question and Staff cares was the least important of the top seven questions.

Second, Figure 13-3 compares the effectiveness of a YMCA branch to its peers. The average value of the peer group the branch belonged to appeared as a horizontal line across the middle of the plot. Peer groups were branches whose members are from similar socio-economic status, level of education, and ethnic background. A ball whose center was above the line was performing better for that question than its peers, whereas if the ball center fell below the line, the branch was performing worse than its peers.

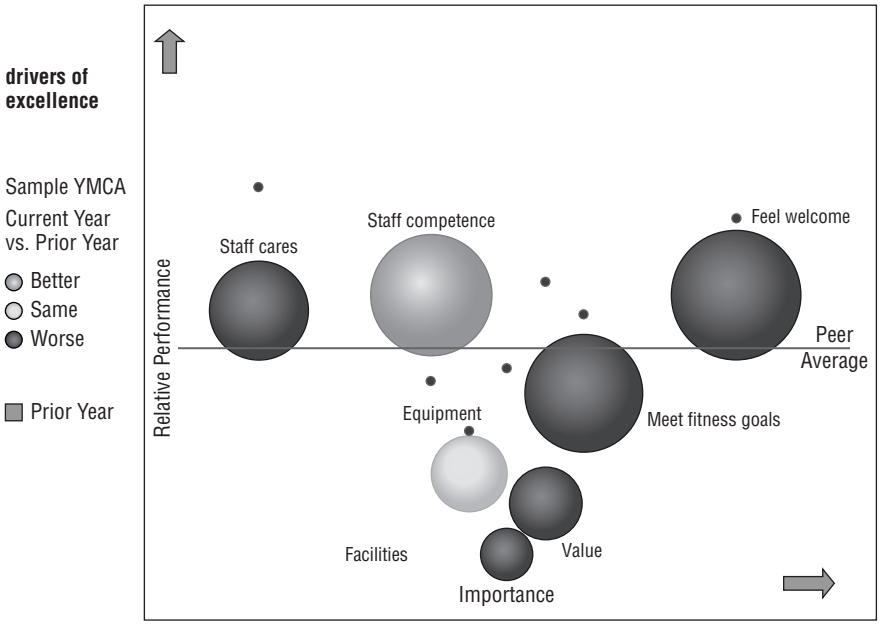


Figure 13-3: Visualization of Drivers of Excellence

Third, Figure 13-3 compares the effectiveness of the branch compared to the values of each factor for the prior year. If the ball was medium gray, the branch was improving compared to the year prior; if it was dark gray, the branch was doing worse; and if the ball was light gray, it was approximately the same as the year prior. To provide a measure of how much better or worse the branch was performing, a small round dot was placed on the plot to indicate the prior year value for each question.

Fourth, the size of the balls indicated the relative importance of the question to IOE for that particular branch; larger balls indicated higher importance, smaller balls lower importance. The order of the balls from left to right was kept the same for every branch, indicating the national trend relating the questions to IOE. The individual branch could therefore identify if it was behaving in ways similar to or different from the national trends.

For the branch shown in Figure 13-3, Staff competence improved compared to the prior year, but most of the other factors were worse than the prior year. Feel welcome had the most significant influence on IOE (just like the average influence nationally). The Facilities question is the smallest ball, and therefore, for this branch, has the least influence on IOE, although it was the fourth most influence nationally.

The second example, Figure 13-4, shows a very successful YMCA branch. All seven of the key factors show that this branch performs better than its peer group, and five of the seven questions show an improvement for this branch compared to its prior year measures. Six of the seven questions were worse than their peer group the year before (the dots are below the peer average line), indicating this branch took steps in the past year to improve the attributes. Why is this so important? If responses to the questions were improved, IOE would improve; this is what the predictive model demonstrated empirically from the data.

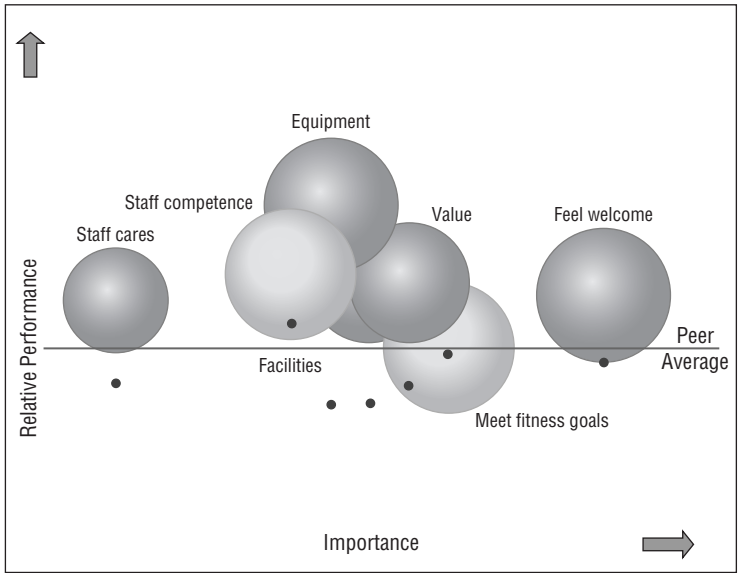


Figure 13-4: Drivers of Excellence, example 2

In the example shown in Figure 13-5, the branch performed at or below its peer group average for every factor, and its performance was flat or mildly improving from the prior year, indicating that the branch is making some improvements to close its gap with other branches. The biggest areas of deficit were related to the facilities and equipment, good candidates for improvement in the next year.

In the example shown in Figure 13-6, the branch was performing worse in six of the seven factors compared to its prior year surveys, and was below its peer group average in the current year. Clearly this branch was heading in the wrong direction and needed extensive improvements.

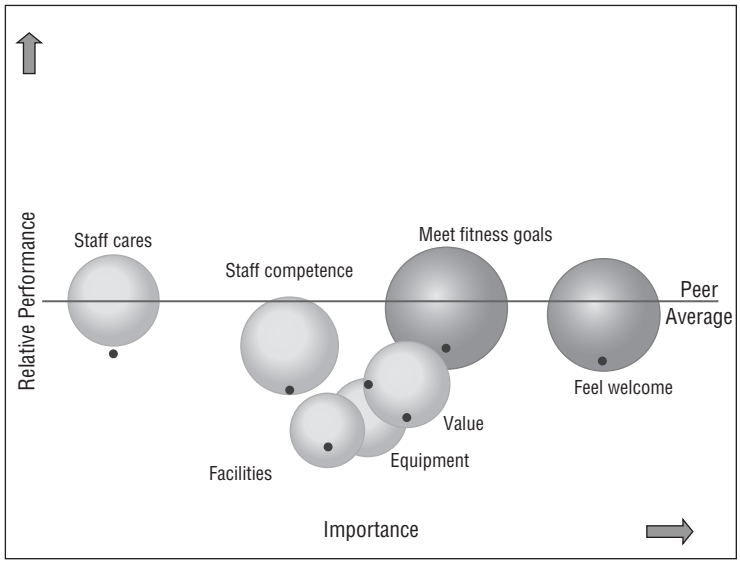


Figure 13-5: Drivers of Excellence, example 3

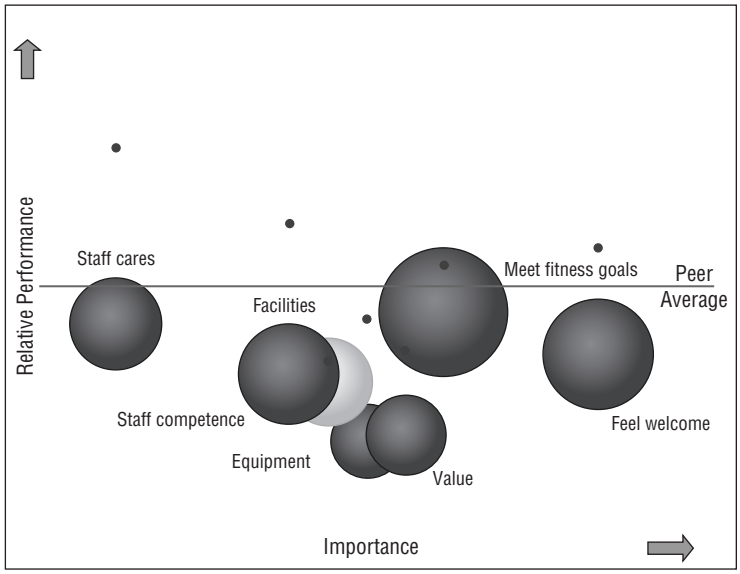


Figure 13-6: Drivers of Excellence, example 4

Finally, in Figure 13-7, the branch performance is mixed. On some dimensions, such as Staff competence, the branch did well: It was higher than its peers and doing better than it was in the prior year. The Staff cares question was also much

better than its peers. On the other hand, the Facilities and Value of the branch were well below the values of its peers and the branch was doing worse than the prior year. The factors Facilities and Value clearly needed improvement.

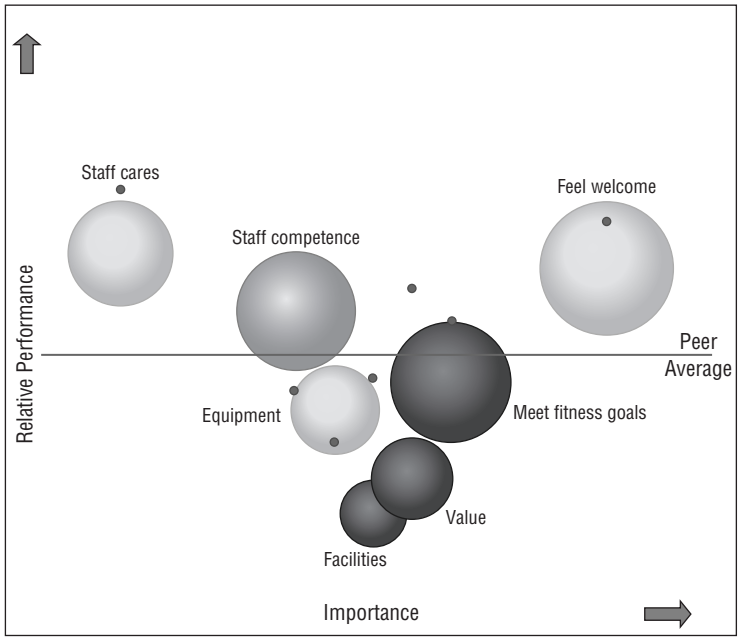


Figure 13-7: Drivers of Excellence, example 5

The idea behind the visualization, therefore, was to provide a report and an explanation of the report to every branch; each branch would only see its own report and could focus on factors where the branch was performing worse than its peers or worse than its performance in the prior year.

Deployment: "What-If" Analysis

One additional deployment strategy considered was incorporating a "what-if" analysis based on model predictions. The premise was that if the branch could change a significant percentage of members that checked the "2" box on a survey question to a "1," how would that change IOE? Figure 13-8 shows the results of some of the analysis. The entire simulation took place in a spreadsheet, where a random proportion of a single question changed in value from 2 to 1—10 percent, 20 percent, and 50 percent—and IOE was recalculated based on the new proportions of 1s and 2s. The percent change in IOE was recorded in the figure.

Recall that the regression models showed that "Feel Welcome" was the most important factor in predicting IOE. However, this factor did not have the highest

sensitivity to change from 2 to 1; that honor belonged to Value for the money. In other words, nationally, on average for all branches, the best way to improve IOE was to do something at the branch that caused a significant percentage of members to change their survey score for Value for the money. In second place was "Staff Cares." Therefore, getting 10 percent or more of the members to believe the staff cares about their well being would result in a 4.2 percent increase in IOE, thus increasing Satisfaction, Intend to renew, and Recommend to a friend.

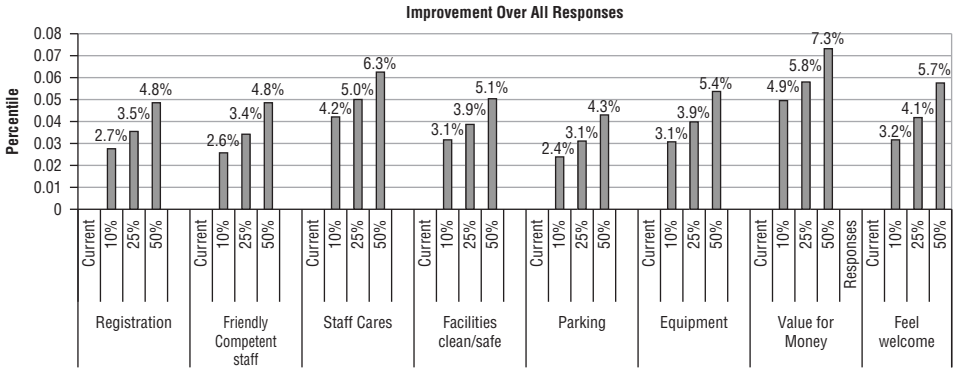


Figure 13-8: What-if scenarios for key questions

This simulation was never delivered to the YMCA but provides a way to use the models that is not directly included in the predictive modeling business objective. In fact, the predictive models didn't directly influence the simulation at all; the role of the regression models was to identify which questions to focus on in the simulation, and therefore was intended to complement the visualizations shown in Figures 13-3 to 13-7.

Revisit Models

Unfortunately, decision-makers found the visualization too complex; it wasn't clear to them exactly what the information meant for their individual branch. There are other predictive modeling approaches that are more accessible, however, including decision trees. Therefore, decision trees were created to uncover key questions in the survey related to the three target variables that would hopefully provide a more transparent interpretation of the survey.

Business Understanding

The business objective remained the same: Identify survey questions related to the three target questions so that individual YMCA branches can evaluate how they can improve member satisfaction, likelihood to renew, and recommendations to friends. However, this time, a key part of the business objectives is making the insights transparent.

Decision trees are often used to gain insights into data because rules are easier to interpret than mathematical equations, especially if the rules are simple. The original three target variables were modeled directly instead of modeling IOE, in keeping with the desire to make the models as easy to understand as possible.

Data Preparation

Because the decision tree algorithm used in the analysis could handle missing values, no data preparation was done to recode NULLs or 0s. However, for each question, dummy variables were created to indicate if the responder checked the "1" box or not; these were the only inputs used in the models.

Modeling and Model Interpretation

We built decision trees using a CART-styled algorithm and determined complexity using cross-validation, a standard CART algorithm practice. The algorithm identified surrogate splits in the tree which we used to help understand the variables that were the most important to predict to target variables. Models were built for each of the three target variables: Satisfaction, Intend to renew, and Recommend to a friend.

Satisfaction Model

The tree for target variable Q1 = 1 (Satisfaction) is shown in Figure 13-9. If the question response was 1, the path of the tree goes down the left side of the split. The number at the bottom of a branch labels the terminal node and indicates it is one of the most significant terminal nodes. For example, terminal node 1 follows the rule Q25 = 1 and Q13 = 1, whereas terminal node 2 follows the rule Q25 = 1 and Q13 ≠1 and Q22 = 1. Table 13-11 shows a list of variables included in the tree.

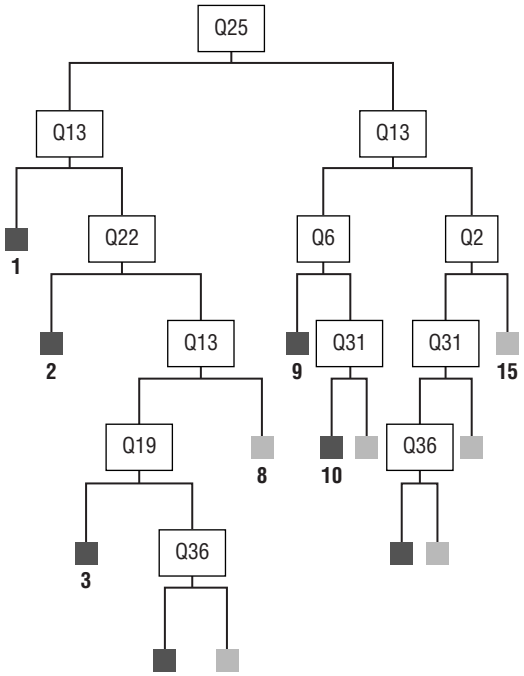


Figure 13-9: Member satisfaction tree

Table 13-11: Key Variables Included in the Satisfaction Tree

QUESTION	DESCRIPTION
Q2	Efficiency of front desk procedures
Q6	Staff competence
Q13	Overall cleanliness
Q19	Has the right equipment
Q22	Value for the money
Q25	You feel welcome at the Club.
Q31	Compared to other organizations in your community or companies you deal with, please rate your loyalty to the Club.

Characterizing the rules in English rather than in mathematical symbols is necessary to facilitate interpretation. Decision tree rules can be read as sequences of "and" conditions. The rules in the tree indicate the best way to predict the target variable according to the algorithm, though they aren't necessarily the only way to achieve comparable predictive accuracy; sometimes other combinations of splits in the tree can yield nearly identical accuracy.

Table 13-12 shows a summary report based on the tree in Figure 13-9. The key terminal nodes were defined as those terminal nodes with much higher than average satisfaction proportions and those with much lower than average satisfaction proportions. The rules defined by the branch of the tree were put into English to make it easier for decision-makers to understand what the rules were communicating about the member attitudes represented by the branch.

Table 13-12: Rule Descriptions for the Satisfaction Model

TERMINAL NODE	RULE
1	If strongly agree that facilities are clean and strongly agree that member feels welcome, then highly satisfied.
9	If strongly agree that facilities are clean, and strongly agree that staff is competent, even if don't strongly agree feel welcome, then highly satisfied.
2	If strongly agree that feel welcome and strongly agree Y is value for money, even if don't strongly agree facilities are clean, then highly satisfied.
3	If strongly agree that Y has the right equipment and strongly agree that feel welcome, and somewhat agree that facilities are clean, even though don't strongly feel Y is good value for the money, then highly satisfied.
10	If strongly agree that loyal to Y and strongly agree that facilities are clean, even though don't strongly agree that feel welcome nor strongly agree that staff is competent, then highly satisfied.
8	If don't strongly agree that facilities are clean and don't strongly agree that the Y is good value for the money, even though strongly agree that feel welcome, member isn't highly satisfied.
15	If don't strongly agree that staff is efficient and don't strongly agree that feel welcome, and don't strongly agree that the facilities are clean, then member isn't highly satisfied.

Finally, Table 13-13 shows several key terminal nodes from the satisfaction model, including two key statistics. The first key statistic is the proportion of the population with high satisfaction. The highest value is found in terminal node 1 (72.8 percent). The second key statistic is also important, however: the proportion of all highly satisfied members identified by the rule. The top terminal node by percent satisfaction is terminal node 1, but also important is that the rule finds nearly half of all highly satisfied members (49.1 percent). In fact, the top three rules, terminal nodes 1, 2, and 9, comprise over 70 percent of all highly satisfied members. The key questions included in these rules are summarized in Table 13-14, with a "yes" if the question has the value 1 in the

branch, "no" if the question has a value greater than 1 in the branch, and "NA" if the question is not in the branch.

Note as well that terminal node 15, the terminal node with the lowest satisfaction, is primarily the converse of the best rule: If the member doesn't agree that they feel welcome, the facilities are clean and the staff is efficient, only 6 percent of the members were highly satisfied with the branch; it makes one wonder why these 6 percent were still highly satisfied!

Table 13-13: Key Terminal Nodes in the Satisfaction Model

TERMINAL NODE	NUMBER OF SURVEYS IN NODE	PERCENT OF ALL SURVEYS FALLING INTO NODE	NUMBER OF HIGHLY SATISFIED IN TERMINAL NODE	PERCENT OF HIGHLY SATISFIED IN TERMINAL NODE	PERCENT OF ALL HIGHLY SATISFIED IN TERMINAL NODE
1	10,014	20.80	7,289	72.8%	49.1
9	1,739	3.60	904	52.0%	6.1
2	4,578	9.50	2,317	50.6%	15.5
3	1,014	2.11	471	46.5%	3.2
10	998	2.08	431	43.2%	2.9
8	1,364	2.80	141	10.3%	1.0
15	19,323	40.20	1,231	6.4%	8.3

The highest satisfaction comes from clean facilities and a feeling of being welcome at the Y. However, the branch can overcome a lack of cleanliness by demonstrating value in being a member of the Y (terminal node 2), and the branch can overcome a lack of feeling welcome with high staff competence. These "even if" insights from the decision tree provide nuance to the picture of why members are satisfied with their Y branch. These interaction effects cannot be learned or inferred from the regression models without introducing the interaction effects directly.

Table 13-14: Key Questions in Top Terminal Nodes

TERMINAL NODE	FEEL WELCOME	OVERALL CLEANLINESS	STAFF COMPETENCE	VALUE FOR MONEY
1	yes	yes	NA	NA
2	yes	no	NA	yes
9	no	yes	yes	NA

Recommend to a Friend Model

The Recommend to a friend model appears in Figure 13-10, which shows that the key questions are Feel welcome (Q25) and Loyal to the Y (Q31), with terminal node summary statistics shown in Table 13-15. The loyalty question was interesting because the other two models did not use this question at all, nor was it a competitor or surrogate split in the other models. The top rule—terminal node 1—found a population with 88.6 percent highly likely to recommend to a friend, comprising nearly half of all members who strongly recommend the Y. Once again, the top three terminal nodes represented more than 70 percent of the target variable population. These three terminal nodes (1, 2, and 4) included the questions related to Loyalty to the Y, Feel welcome, and Value for the money (Q31, Q25, and Q22). The descriptions of top rules for Recommend to a friend are shown in Table 13-16.

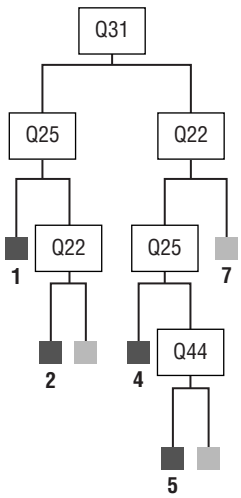


Figure 13-10: Recommend to a Friend decision tree

Table 13-15: Terminal Node Populations for the Recommend to a Friend Model

TERMINAL NODE	NUMBER OF SURVEYS IN NODE	PERCENT OF ALL SURVEYS FALLING INTO NODE	NUMBER OF HIGHLY RECOMMEND TO FRIEND IN TERMINAL NODE	PERCENT OF HIGHLY RECOMMEND TO FRIEND IN TERMINAL NODE	PERCENT OF ALL HIGHLY RECOMMEND TO FRIEND IN TERMINAL NODE
1	13678	28.46	12122	88.60	47.0
2	6637	13.80	4744	71.50	18.4

Continues

Table 13-15 (*continued*)

TERMINAL NODE	NUMBER OF SURVEYS IN NODE	PERCENT OF ALL SURVEYS FALLING INTO NODE	NUMBER OF HIGHLY RECOMMEND TO FRIEND IN TERMINAL NODE	PERCENT OF HIGHLY RECOMMEND TO FRIEND IN TERMINAL NODE	PERCENT OF ALL HIGHLY RECOMMEND TO FRIEND IN TERMINAL NODE
4	2628	5.50	1932	73.50	6.1
7	21865	45.50	5461	25.00	21.2
5	814	1.70	509	62.50	2.0

Table 13-16: Rule Descriptions for the Recommend to a Friend Model

TERMINAL NODE	RULE
1	If strongly agree that loyal to Y and strongly agree that feel welcome, then strongly agree that will recommend to a friend.
2	If strongly agree that loyal to Y and agree that Y is a good value for the money, even though don't strongly agree feel welcome, strongly agree will recommend to a friend.
4	If strongly agree that Y is a good value for the money and strongly agree that feel welcome, even though not strongly loyal to Y, strongly agree will recommend to a friend.
7	If don't strongly agree that loyal to Y and don't strongly agree that Y is value for the money, then will not highly recommend to a friend.
5	If strongly agree that Y is good value for the money, and strongly agree that Y helps meet fitness goals, even though not strongly loyal to the Y and don't strongly feel welcome, will highly recommend to a friend.

The interesting contrast of the Recommend to the Friend model compared to the satisfaction model is the inclusion of loyalty here; when a member feels a sense of loyalty to their branch, they are more likely to recommend it to a friend. If a member is loyal, if the member feels welcome or believes the Y is a good value for the dollar, they are likely to recommend it. But even without the loyalty, meeting fitness goals and good value for the dollar is enough for the member to recommend the Y to a friend.

Intend to Renew Model

The Intend to Renew model appears in Figure 13-11, which shows that the key questions were Feel welcome (Q25), Y helps to meet fitness goals (Q44), and Value for the money (Q22), with terminal node summary statistics shown in Table 13-17. Rules using the fitness goals question were interesting because the other

two models did not use this question in their top three terminal nodes, though it was in the fourth best terminal in the Recommend to a friend model, and in that same model was a surrogate for the Loyal to the Y split. As a reminder, surrogate splits identify a variable that splits the most like the winning split for a node in the tree.

The top rule—terminal node 1—found a population with 73.9 percent highly likely to intend to renew, comprising nearly half of all members who intended to renew. Once again, the top three terminal nodes represented more than 70 percent of the target variable population. The description of top rules for Intend to renew are shown in Table 13-18.

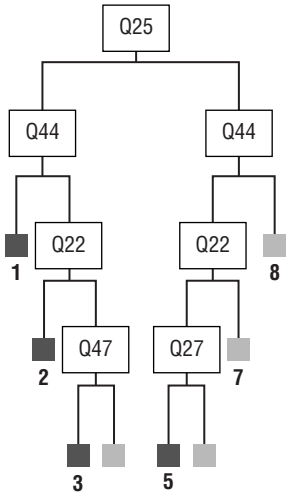


Figure 13-11: Intend to Renew decision tree

Table 13-17: Terminal Node Populations for the Intend to Renew Model

TERMINAL NODE	NUMBER OF SURVEYS IN NODE	PERCENT OF ALL SURVEYS FALLING INTO NODE	NUMBER OF HIGH INTEND TO RENEW IN TERMINAL NODE	PERCENT OF HIGH INTEND TO RENEW IN TERMINAL NODE	PERCENT OF ALL HIGH INTEND TO RENEW IN TERMINAL NODE
1	13397	27.90	9903	73.90	48.40
2	3051	6.30	1823	59.80	8.90
5	5704	11.90	3201	56.10	15.60
8	18547	38.60	3130	16.90	15.30
7	2178	4.50	578	26.50	2.80

Table 13-18: Rule Descriptions for the Intend to Renew Model

TERMINAL NODE	RULE
1	If strongly agree that feel welcome and strongly agree that Y helps meet fitness goals, then strongly agree that intend to renew.
2	If strongly agree Y is good value for the money and strongly agree that feel welcome, even if don't strongly agree that Y helps meet fitness goals, then strongly agree that intend to renew.
5	If strongly agree that feel sense of belonging, and agree that Y is value for the money, and strongly agree that Y helps meet fitness goals, even if don't feel welcome, then strongly agree intend to renew.
8	If don't strongly agree that feel welcome and don't strongly agree that Y helps meet fitness goals, then don't strongly agree that intend to renew.
7	If don't strongly agree that Y is good value for money and don't strongly agree that feel welcome, even if strongly agree Y helps meet fitness goals, don't strongly agree that intend to renew.

Fitness goals figured strongly in the Intend to renew models when coupled with feeling welcome (73.9 percent). Conversely, negating both of these reduced the Intent to renew to 16.9 percent, a ratio of more than four to one. Value for the money can help overcome fitness goals not being met, but only partially. Terminal nodes 2 and 5 had Intent to renew percentages of 59.8 and 56.1, respectively, well below the top terminal node.

Summary of Models

The Satisfaction model was more complex than the other two models, which implied that the reasons for satisfaction are more complex. Each of the models captured different characteristics of the members. The differences also highlight the importance of defining the target variable well. For this project, all three of the target variables provided insights into the attitudes of members.

Feel welcome was a top question for all three models and provided an important insight into the mindset of the members; the connection to the Y was more important than the conditions of the facility, parking, programs, or staff competence. Value for the money was also a key question in all three models, showing that member costs were significant contributors to the attitudes of members toward their branch.

However, each model also had one or two questions that differed from the other models, summarized in Table 13-19. These differences could be useful in helping decision-makers tune the changes to the target they are most interested in. For example, if the branch wishes to improve member satisfaction, in addition to making members feel welcome and revisiting the value of the Y,

they can make sure staff is trained well (Staff competence) and the facilities are kept clean.

Table 13-19: Key Questions That Differ between Target Variables.

MODEL	KEY QUESTION(S)
Satisfaction	Facility cleanliness, Staff competence
Recommend to a friend	Loyalty to the Y
Intend to renew	Y helps to meet fitness goals

Deployment

The models were not deployed nationwide; individual branches decided to what degree the models were used to influence what changes were made within their branches. After years of using models, a 32 percent improvement in satisfaction (Q1 = 1) was measured, which clearly indicates improvement at the branch level in meeting the expectations of members. Additionally, the Recommend to a friend (Q48 = 1) improved by 6 percent, easily a statistically significant improvement based on the number of surveys, though operationally not much higher than the original value.

Summary and Conclusions

Two predictive modeling approaches were described in this chapter, the first a mostly traditional approach to modeling, including the use of factor analysis and stepwise regression. The model visualization reports provided a rich, branch-level summary of the status of key questions on the survey that influence satisfaction. It ultimately was not used because of its complexity.

The second took a machine learning approach, building decision trees to predict the target variable. Simple rules were built that provided a more transparent view of which questions influence Satisfaction, Intend to renew, and Recommend to a friend, including how much interactions in key questions relate to these three target questions.

Neither modeling approach is right or wrong; they provide different ways to understand the surveys in complementary ways. A more sophisticated approach could even incorporate both to find main effects and interactions related to IOE and its components. Given more time and resources, more avenues of analysis could have been attempted, but in most projects, time and resources are limited, requiring the analysts to make decisions that may not be optimum, but hopefully are reasonable and will help the company improve decisions.

This case study also demonstrates the iterative nature of predictive modeling solutions. The data was plentiful and relatively clean, and the target variables were well defined. Nevertheless, translating the insights from the models was not straightforward and required considerable thought before communicating these insights to the decision makers. Success in most predictive modeling projects hinge on how well the information gleaned from the models—both predictions and interpretations—can ultimately be leveraged.

Since the completion of work described in this case study, Seer Analytics has progressed well beyond these models and has developed a framework that addresses the target variable from a different perspective; they readdressed the objectives of modeling to better match the business needs. Instead of predicting influencers for factors that drive satisfaction, they now define six key components of the member experience and create a pyramid of factors that are hierarchical. These six factors are, in order from bottom to top, Facility, Value, Service (of staff), Engagement (with staff and members), Health (meeting goals), and Involvement. This approach has resonated much better with YMCA branches and is in use today.

Help Desk Case Study

In the second case study, a combination of text mining and predictive modeling approaches were used to improve the efficiency of the help desk of a large U.S. corporation responsible for hardware services and repairs of devices it manufactured, sold, and supported.

The problem the company needed to address is this: Could the company use the description of problems transcribed from help desk calls to predict if a part would be needed to resolve the problem. After receiving a help desk phone call and after the call was processed, a support engineer was assigned to the ticket to try to resolve it. The engineer first tried to resolve the problem over the phone, and if that wasn't successful, the engineer went to the customer site to resolve the ticket.

In particular, the problem was efficiency. Knowing whether a part was needed for a repair or not before going to the customer site was helpful. Even more important was predicting which parts were most likely to be needed in the repair. These models were built for the modeling project but will not be described in this case study. A second set of models were built to predict the actual part that was needed to complete the repair, though that part of the modeling is not described here.

Concerns over revealing too much to competitors prevents me from revealing the client. Additionally, specifics about the project are not provided, such as actual performance numbers, precise definitions of derived variables, and

the identities of the most predictive variables in the models. The project was so successful that the details about the models and how much the models improved efficiency became a strategic corporate asset. Nevertheless, even without the details, the principles used to solve the problem can be applied broadly.

The accuracy of the model was paramount for successful deployment. The decision makers determined the minimum accuracy of the "parts needed" model for the model to be successful, a number that cannot be revealed here. The entire data set did not need to be classified at this rate. If even 20 percent of the tickets that needed a part to complete the repair could be identified correctly, the model could be successfully deployed. The remaining tickets would then be processed as they had always been processed.

Data Understanding: Defining the Data

The unit of analysis was a support ticket, so each row contained a unique ticket ID and columns contained descriptions of the ticket. More than 1,000,000 tickets were available for modeling. For each support ticket, the following variables were available as candidate inputs for models:

- Date and time of the call
- Business/individual name that generated the ticket
- Country of origin
- Device error codes for the hardware
- The reason for the call that generated the ticket: regularly scheduled maintenance call or a hardware problem
- Warranty information
- The problem description (text transcribed from the call)
- The outcome of the ticket: ticket closed, solution found, and so on
- The part(s) needed to close the ticket

The target variable had two parts: Did the ticket require a part to close, and which part or parts were needed to close the ticket. Only the PartsUsed target variable models are described in this case study.

Data Preparation

Extensive preparation was done on the data to examine all codes, correcting incorrect codes when they were identified as incorrect. The date of the call was transformed into a single column: day of week. The business name was not used in building models, but helped in interpreting them.

Problems with the Target Variable

The target variable definition was clear: Create a 1/0 dummy variable indicating if the ticket required a part to complete the repair. The target variable was labeled the PartsUsed flag. There were, however, ambiguities even with this definition. First, if a part was not necessary for the repair, although it helped the repair, it was still coded as a part being used.

In other cases, a part was used to fix a problem that was not the one specified on the ticket. If this situation was discovered, the PartsUsed flag was set to 0 because a part the engineer specified to close the ticket was not actually needed to fix the problem on the ticket, and the engineer should have opened a new ticket for the second problem. These ambiguities were difficult to discover, however, and it is presumed that these problems persisted in the data.

A third problem occurred when a fix was made using a part and the ticket was closed, only to reopen later when the problem reappeared. If the problem was subsequently solved using a different part, the PartsUsed flag would still be coded as a 1 (though a different part was used to fix the problem). However, if a part was not needed to fix the problem, the PartsUsed flag would have to be changed to a 0. If the ticket was coded properly, this would work itself out.

Feature Creation for Text

The company realized early on in the project that extracting features from the transcribed text was critical to the success of the models. Initially, text was extracted using SQL text matching rather than a text mining software package, and it was done quite well. After finding some success using SQL to extract text, they initiated a search for text mining software and technology to give it the ability to leverage the transcription of help desk calls.

Problems with the text were those problems common in text mining including misspellings, synonyms, and abbreviations. Fortunately, the client had an excellent database programmer who spent considerable time fixing these problems.

Table 13-20 shows a sample of the kinds of misspellings found in the data. The database programmer not only recoded these to a common spelling, but even without any knowledge of stemming, he essentially stemmed these words, capturing the concepts into one keyword. The keyword STICK from the table therefore included all of the variants listed in the table.

Table 13-20: Data Preparation for Help Desk Text

ID	WORD	KEYWORD	COUNT
1220	STCK	STICK	19
1221	STIC	STICK	16

1222	STICK	STICK	61
1223	STICKIN	STICK	15
1224	STICKING	STICK	1,141
1225	STICKS	STICK	179
1226	STICKY	STICK	176

As a second example, the keyword FAILURE included words FAILURE, FAIL, FAILURES, FAILED, and FA found in the text. These words were combinations of inflected words and abbreviations.

Domain experts and database programmers worked together to build a list of keywords deemed to be potentially useful in building the predictive models. After combining synonyms, stemming, and converting abbreviations to the appropriate keyword, the team identified more than 600 keywords and phrases.

Modeling

Several algorithms could have been built to achieve the business objectives, including neural networks, support vector machines, and decision trees, but two considerations pushed the team to select decision trees. First, the data was largely categorical and the number of candidate inputs was very large, two areas decision trees handle well. Second, the company needed to build models that were easy to interpret so support engineers could understand why a part was predicted to be needed for the repair. For these reasons, decision trees were used for modeling.

Hundreds of decision trees were built from the modeling data using different partitions of training and testing data records, different variable subsets of keywords, and different tree settings for priors and complexity penalties. The tree in Figure 13-12 is typical of the trees that were built. The column name for the PartsUsed target variable in the tree is Parts. Terminal nodes had the percentage of records needing a part (Class equal to Parts) ranging from a low of 1.5 percent in Terminal Node 2 to 50.5 percent in Terminal Node 8. The nodes in Figure 13-12 are color coded by percentage of records needing a part to facilitate seeing where the percentages are highest.

However, 50.5 percent fell far short of the minimum value required in the business objectives, even though the 50.5 percent Parts rate represented a lift of more than two over the baseline rate of 23.2 percent. This tree and the hundreds of others were clearly insufficient to achieve the goals of the model.

What was the problem? The trees were able to handle the large number of keywords but struggled to find combinations that produced high percentages of tickets with parts use. Decision trees sometimes struggle with sparse data

because of the greedy search strategy: Each keyword dummy variable was a sparse variable, populated in a small minority of tickets, and individual keywords didn't necessarily provide enough information on their own to create good splits.

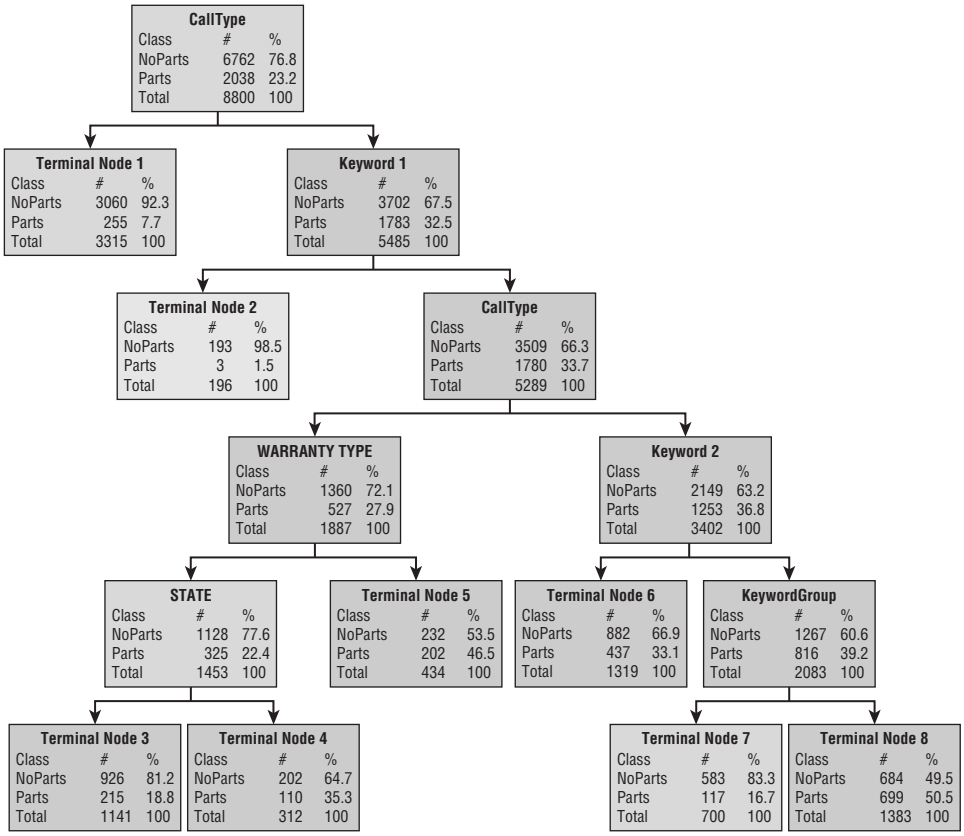


Figure 13-12: Typical parts prediction decision tree

Model ensembles, such as Random Forests, helped the accuracy some, but still not enough to achieve the business objective. Random Forests, in particular, represent a good strategy for this kind of problem because of the random variable subsets that are part of the algorithm, forcing the trees to find many ways to achieve high accuracy. But the modest improvement in accuracy also came at the expense of transparency of the rules, an important factor to communicate to the support engineer before going to the company site; even if the Random Forests solution was accurate enough, it is unlikely the company would have adopted it as the model to deploy.

Some trees were very interesting because they did not determine if parts were needed from the keywords, but rather from the business making the call,

the level of expertise of their internal support within the company making the support call, or specific devices needing repair. Trees, to some degree, were therefore measuring which companies had better internal support (if they had good support, parts were more likely to be needed because they fixed the easy problems themselves).

The best branches in the trees were actually ones that predicted a part *not* being needed. For example, in one tree, when both keywords "machine" and "down" existed in the ticket, parts were rarely needed for the repair. Rules like this one were valuable, but not the kind of rule that the company needed for successful deployment.

Revisit Business Understanding

The research team went back to the drawing board to try to find a different way to solve the problem. They asked good questions, such as "What additional information is known about problems at the time tickets are submitted?" and "How would an informed person think about the data to solve the problem?" The answers to these questions were in the historical context of the tickets themselves.

When a ticket came in, a support engineer who was trying to predict what the problem might be and whether a part was needed for a repair would refine their conclusion by eliminating options that were not likely, just as decision trees do. But the engineer would also group together problems naturally, using an "or" condition, an operation that trees do not do in a single split. But even these conditions did something subtle in the mind of the support engineer.

The keywords and other features represented a historical idea: the experience of the engineer. The experiences of the engineer also included temporal information; one combination of keywords may have been a problem for particular kinds of devices at one time, but two years prior to that time the problem didn't exist.

The analysts and decision makers then determined the following. Rather than using codes and keywords directly, they would use historic information about parts being needed when those codes and keywords appeared in the ticket as inputs. Let's say "paper jam" was a keyword in the data. Rather than using this as a dummy variable, use the percentage of times this pair needed a part in the past year.

Figure 13-13 is a visual representation of the thought process. Ticket 1 was resolved at some timestamp prior to "Now," where "Now" represents the date the data for modeling was created. The date of resolution for Ticket 1, called "Ticket 1 Timestamp" in the figure, means Ticket 1 has a target variable. The input variables for Ticket 1, PartsUsed percentages for keywords and codes in Ticket 1, were generated for the time period range represented by the curly bracket with the label "Time Period to Compute PartsUsed for Keywords in Ticket 1."

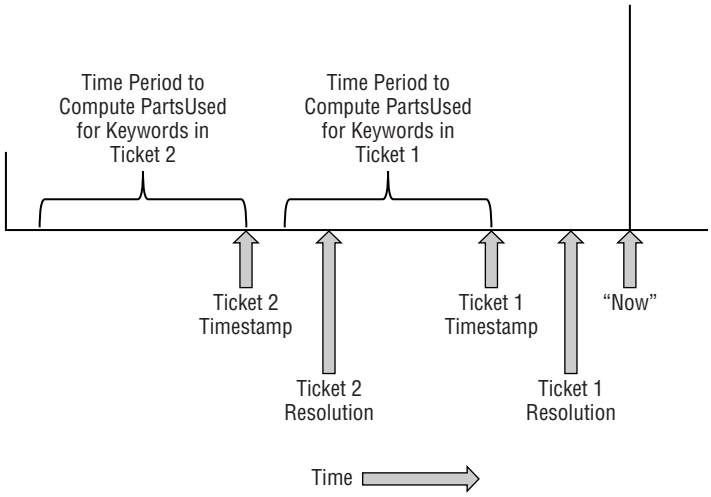


Figure 13-13: Temporal framework for new features

The same process was done for each ticket, including Ticket 2 shown in the figure, and all other tickets (more than one million).

What's the difference between this and just using the flag? What is the difference between this data and the data used in the first modeling pass? After all, doesn't the tree compute the average PartsUsed rate for each keyword in the data and show us those in the nodes and terminal nodes of the tree?

First, the new representation contains richer information: instead of a 1/0 dummy, you have a percentage. Second, there is additional temporal information in the new features missing from the dummy variables. When models were built from the dummy variables, all occurrences of the keywords and error codes were included in the tree regardless of when the ticket occurred. In the new representation, tickets occurring after the date for Ticket 1 ("Ticket 1 Resolution") are not included in the computation of the PartsUsed percentage. Therefore, the new variables take trends into account in ways the first data set couldn't; the first models included all tickets in the modeling data regardless of when the tickets occurred.

New features included not only historic PartsUsed, but also counts of how many tickets contained the keyword or code.

Modeling and Model Interpretation

Decision trees were used once again for building the predictive models because they could scale well. In addition, many decision trees could be built easily by modifying parameters, such as priors or misclassification costs and the set of inputs that could be included in the models.

Nevertheless, the trees still did not achieve the classification accuracy required by the business objectives if measured by Percent Correct Classification (PCC) or by using a confusion matrix. What was new was that there were finally at least some terminal nodes in most of the trees that did meet the business objective PartsUsed rate, even if only for a small percentage of the overall records.

When examining these terminal nodes, the analysts discovered that the rules describing the path to the terminal nodes differed from one another, as one would expect from decision trees. Moreover, these terminal nodes were not identifying the same populations: There were more than a few ways to predict a high percent of PartsUsed for different groups of tickets. Interestingly, some of the rules contained six or more conditions (the trees were six or more levels deep), but even in these situations, the rules made sense to domain experts.

At this point, an astute analyst may see that this kind of data is ideal for ensembles. Random Forests (RF), in particular, is a good match for generating trees from different input variables and achieving higher accuracy than individual trees can achieve. However, even though the RF models had higher accuracy than individual trees, the RF models had two problems. First, the full trees did not achieve overall accuracy that was high enough to deploy. Second, the individual branches of RF trees were (purposefully) overfit and unsuitable to use as a way to interpret why a part was needed.

Therefore, the rules that showed the most potential had to be plucked from thousands of trees. The most interesting rules were those that related to high percentages of PartsUsed, but the rules matching very low PartsUsed percentages were also interesting as cases that could be solved without parts being needed and therefore were good candidates for phone support.

An alternative to picking the best rules from decision trees is to use association rules. Advantages of association rules are that they find (exhaustively) all combinations of rules rather than just the best paths found by decision trees. A disadvantage is that the inputs must all be categorical, so the PartsUsed percentages would have to be binned to create the categorical representation for every keyword and code.

Deployment

More than 10,000 decision trees were built, creating more than 20,000 terminal nodes with a high percentage of PartsUsed to complete the repair. Each terminal represents a rule—a series of "and" conditions found by the decision tree. These rules were collected and sorted by the predicted percent PartsUsed for the terminal node. The sorted list of rules then was treated as a sequence of rules to fire, from highest PartsUsed likelihood to lowest, though even the lowest contained a high percentage of PartsUsed.

The algorithm for applying rules in this way can be thought of in this way:

1. Sort the rules by percent PartsUsed on training data in descending order. Each rule contains not only the PartsUsed percentage, but also the variables and splits (the rules) used to generate the PartsUsed percentage.
2. Choose one or more tickets to apply the rules to.
3. Run the first ticket through the rules, stopping when a rule matches (or "fires") with the highest PartsUsed percentage.
4. Repeat for all tickets in the set.

Table 13-21 contains a sample list of rules. As a reminder, these numbers are not real and are used for illustration purposes only; they should not be used to infer the PartsUsed percentages found and used by the company. The PartsUsed percentage found during training is shown in the table, but the more important number is the PartsUsed Percentage in Sequence, a number that can be significantly different than the Training PartsUsed Percentage.

Table 13-21: Sample List of Rules to Fire

RULE ID	RULE SEQUENCE NUMBER	NUMBER TICKETS MATCHING RULE	TRAINING PARTSUSED PERCENTAGE	PARTSUSED PERCENTAGE IN SEQUENCE	CUMULATIVE PARTSUSED PERCENTAGE
278	1	11809	93.6	93.6	93.56
2255	2	7215	93.5	91.7	92.85
1693	3	1927	93.5	93.0	92.85
2258	4	16	93.5	70.5	92.84
1337	5	20	93.5	89.3	92.83
993	6	2727	93.5	84.1	91.82
977	7	2	93.5	94.0	91.82
2134	8	2516	93.5	92.8	91.91
1783	9	4670	93.5	93.1	92.09
984	10	6	93.5	94.0	92.09

For example, consider Rule ID 2258. This rule matched only 16 tickets because the thousands of tickets it matched during training already matched one of the prior rules in the sequence: 278, 2255, or 1693. Moreover, the best tickets from rule 2258 were gone: The remaining 16 tickets had only a 70.5 percent PartsUsed rate. This rule should therefore be pruned from the list. The same applies to rules 1337, 977, and 984.

For the company in this case study, these rules were particularly attractive because they could be converted to SQL so easily, making deployment simple.

Summary and Conclusions

Several aspects of this case study were critical to the success of the project. First, deriving new, innovative features to use for modeling made the difference between failure and success. Sometimes one hears that more data will overcome algorithms and domain expertise. In this case, this was clearly false. It wouldn't matter how many tickets were included in the models. If the new features weren't created, the performance would still be below the requirements. Subject matter experts were essential to defining the new features.

Second, using decision trees allows the modeling to proceed quickly because trees handle wide data so easily and efficiently. Moreover, adjusting learning parameters for the trees produced thousands of trees and interesting terminal nodes. Third, the decision trees were not used blindly and as black boxes. When they were considered feature creators themselves, finding interesting business rules, the fact that no tree solved the entire problem was irrelevant. Finally, using the decision trees to find business rules meant the list of rules was intuitive and easy to communicate to the support engineers.

The models went into production and were very successful for the company, and so much so that the modeling was expanded to try to include more tickets in the models and to produce models that matched more tickets with high PartsUsed scores. Cost savings for the company as a result of the models were significant.

Index

D

L

M

N

T

W

X Y Z