Hui Lin and Ming Li

Introduction to Data Science

Contents

List of Tables

List of Figures

Preface

During the first couple years of our career as data scientists, we were bewildered by all kinds of data science hype. There is a lack of definition of many basic terminologies such as "Big Data" and "Data Science." How big is big? If someone ran into you asked what data science was all about, what would you tell them? What is the difference between the sexy role "Data Scientist" and the traditional "Data Analyst"? How suddenly came all kinds of machine algorithms? All those struck us as confusing and vague as real-world data scientists! But we always felt that there was something real there. After applying data science for many years, we explored it more and had a much better idea about data science. And this book is our endeavor to make data science to a more legitimate field.

Goal of the Book

This is an introductory book to data science with a specific focus on the application. Data Science is a cross-disciplinary subject involving hands-on experience and business problem-solving exposures. The majority of existing introduction books on data science are about the modeling techniques and the implementation of models using R or Python. However, they fail to introduce data science in a context of the industrial environment. Moreover, a crucial part, the art of data science in practice, is often missing. This book intends to fill the gap.

Some key features of this book are as follows:

- It is comprehensive. It covers not only technical skills but also soft skills and big data environment in the industry.

- It is hands-on. We provide the data and repeatable R and Python code. You can repeat the analysis in the book using the data and code provided. We also suggest you perform the analyses with your data whenever possible. You can only learn data science by doing it!

- It is based on context. We put methods in the context of industrial data science questions.

- Where appropriate, we point you to more advanced materials on models to dive deeper

Who This Book Is For

Non-mathematical readers will appreciate the emphasis on problem-solving with real data across a wide variety of applications and the reproducibility of the companion R and python code.

Readers should know basic statistical ideas, such as correlation and linear regression analysis. While the text is biased against complex equations, a mathematical background is needed for advanced topics.

What This Book Covers

Based on industry experience, this book outlines the real world scenario and points out pitfalls data science practitioners should avoid. It also covers big data cloud platform and the art of data science such as soft skills. We use R as the main tool and provide code for both R and Python.

Conventions

Acknowledgements

About the Authors

Hui Lin is leading and building data science department at Netlify. Before Netlify, she was a Data Scientist at DuPont. She was a leader in the company of applying advanced data science to enhance Marketing and Sales Effectiveness. She provided data science leadership for a broad range of predictive analytics and market research analysis from 2013 to 2018. She is the cofounder of Central Iowa R User Group, blogger of scientistcafe.com and 2018 Program Chair of ASA Statistics in Marketing Section. She enjoys making analytics accessible to a broad audience and teaches tutorials and workshops for practitioners on data science (`https://course2019.netlify.com/`). She holds MS and Ph.D in statistics from Iowa State University.

Ming Li is currently a Senior Data Scientist at Amazon and an Adjunct Faculty of Department of Marketing and Business Analytics in Texas A&M University - Commerce. He is the Chair of Quality & Productivity Section of ASA for 2017. He was a Data Scientist at Walmart and a Statistical Leader at General Electric Global Research Center. He obtained his Ph.D. in Statistics from Iowa State University at 2010. With deep statistics background and a few years' experience in data science, he has trained and mentored numerous junior data scientist with different background such as statistician, programmer, software developer, database administrator and business analyst. He is also an Instructor of Amazon's internal Machine Learning University and was one of the key founding member of Walmart's Analytics Rotational Program which bridges the skill gaps between new hires and productive data scientists.

1

Introduction

Interest in data science is at an all-time high and has exploded in popularity in the last couple of years. Data scientists today are from various backgrounds. If someone ran into you ask what data science is all about, what would you tell them? It is not an easy question to answer. Data science is one of the areas that everyone is talking about, but no one can define.

Media has been hyping about "Data Science" "Big Data" and "Artificial Intelligence" over the past few years. I like this amusing statement from the internet:

"When you're fundraising, it's AI. When you're hiring, it's ML. When you're implementing, it's logistic regression."

For outsiders, data science is whatever magic that can get useful information out of data. Everyone should have heard about big data. Data science trainees now need the skills to cope with such big data sets. What are those skills? You may hear about: Hadoop, a system using Map/Reduce to process large data sets distributed across a cluster of computers or about Spark, a system build atop Hadoop for speeding up the same by loading huge datasets into shared memory(RAM) across clusters. The new skills are for dealing with organizational artifacts of large-scale cluster computing but not for better solving the real problem. A lot of data means more tinkering with computers. After all, it isn't the size of the data that's important. It's what you do with it. Your first reaction to all of this might be some combination of skepticism and con-

fusion. We want to address this up front that: we had that exact reaction.

To declutter, let's start from a brief history of data science. If you hit up the Google Trends website which shows search keyword information over time and check the term "data science," you will find the history of data science goes back a little further than 2004. From the way media describes it, you may feel machine learning algorithms were just invented last month, and there was never "big" data before Google. That is not true. There are new and exciting developments of data science, but many of the techniques we are using are based on decades of work by statisticians, computer scientists, mathematicians and scientists of all types.

In the early 19th century when Legendre and Gauss came up the least squares method for linear regression, only physicists would use it to fit linear regression. Now, non-technical people can fit linear regressions using excel. In 1936 Fisher came up with linear discriminant analysis. In the 1940s, we had another widely used model – logistic regression. In the 1970s, Nelder and Wedderburn formulated "generalized linear model (GLM)" which:

"generalized linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value." [from Wikipedia]

By the end of the 1970s, there was a range of analytical models and most of them were linear because computers were not powerful enough to fit non-linear model until the 1980s.

In 1984 Breiman et al(et. al, 1984). introduced the classification and regression tree (CART) which is one of the oldest and most utilized classification and regression techniques. After that Ross Quinlan came up with more tree algorithms such as ID3, C4.5, and C5.0. In the 1990s, ensemble techniques (methods that combine

many models' predictions) began to appear. Bagging is a general approach that uses bootstrapping in conjunction with regression or classification model to construct an ensemble. Based on the ensemble idea, Breiman came up with random forest in 2001(Breiman, 2001a). In the same year, Leo Breiman published a paper "Statistical Modeling: The Two Cultures[1]" (Breiman, 2001b) where he pointed out two cultures in the use of statistical modeling to get information from data:

(1) Data is from a given stochastic data model

(2) Data mechanism is unknown and people approach the data using algorithmic model

Most of the classic statistical models are the first type. Black box models, such as random forest, GMB, and today's buzz work deep learning are algorithmic modeling. As Breiman pointed out, those models can be used both on large complex data as a more accurate and informative alternative to data modeling on smaller data sets. Those algorithms have developed rapidly, however, in fields outside statistics. That is one of the most important reasons that statisticians are not the mainstream of today's data science, both in theory and practice. Hence Python is catching up R as the most commonly used language in data science. It is due to the data scientists background rather than the language itself. Since 2000, the approaches to get information out of data have been shifting from traditional statistical models to a more diverse toolbox named machine learning.

What is the driving force behind the shifting trend? John Tukey identified four forces driving data analysis (there was no "data science" back to 1962):

1. The formal theories of math and statistics
2. Acceleration of developments in computers and display devices

---

[1]http://www2.math.uu.se/~thulin/mm/breiman.pdf

3. The challenge, in many fields, of more and ever larger bodies of data
4. The emphasis on quantification in an ever wider variety of disciplines

Tukey's 1962 list is surprisingly modern. Let's inspect those points in today's context. People usually develop theories way before they find the applications. In the past 50 years, statisticians, mathematician, and computer scientists have been laying the theoretical groundwork for constructing "data science" today. The development of computers enables us to apply the algorithmic models (they can be very computationally expensive) and deliver results in a friendly and intuitive way. The striking transition to the internet of things generates vast amounts of commercial data. Industries have also sensed the value of exploiting that data. Data science seems certain to be a major preoccupation of commercial life in coming decades. All the four forces John identified exist today and have been driving data science.

Benefiting from the increasing availability of digitized information, and the possibility to distribute that through the internet, the toolbox and application have been expanding fast. Today, people apply data science in a plethora of areas including business, health, biology, social science, politics, etc. Now data science is everywhere. But what is today's data science?

1.1   Blind men and an elephant

There is a widely diffused Chinese parable (depending on where you are from, you may think it is a Japanese parable) which is about a group of blind men conceptualizing what the elephant is like by touching it:

"...In the case of the first person, whose hand landed on the

trunk, said: 'This being is like a thick snake'. For another one whose hand reached its ear, it seemed like a kind of fan. As for another person, whose hand was upon its leg, said, the elephant is a pillar like a tree-trunk. The blind man who placed his hand upon its side said, 'elephant is a wall'. Another who felt its tail described it as a rope. The last felt its tusk, stating the elephant is that which is hard, smooth and like a spear." wikipedia[2]

Data science is the elephant. With the data science hype picking up stream, many professionals changed their titles to be "Data Scientist" without any of the necessary qualifications. Today's data scientists have vastly different backgrounds, yet each one conceptualizes what the elephant is based on his/her own professional training and application area. And to make matters worse, most of us are not even fully aware of our own conceptualizations, much less the uniqueness of the experience from which they are derived. Here is a list of somewhat whimsical definitions for a "data scientist":

- "A data scientist is a data analyst who lives in California"
- "A data scientist is someone who is better at statistics than any software engineer and better at software engineering than any statistician."
- "A data scientist is a statistician who lives in San Francisco."
- "Data Science is statistics on a Mac."

"We don't see things as they are, we see them as we are. [by Anais Nin]"

It is annoying but true. So the answer to the question "what is data science?" depends on who you are talking to. Who you may

---

[2] https://en.wikipedia.org/wiki/Blind_men_and_an_elephant

be talking to then? Data science has three main skill tracks: engineering, analysis, and modeling (and yes, the order matters!).

Here are some representative skills in each track. Different tracks and combinations of tracks will define different roles in data science. [3]

1.1.1 Data science role/skill tracks

When people talk about all the machine learning and AI algorithms, they often over look the critical data engineering part that makes everything else possible. Data engineering is the unseen iceberg under the water surface. Think your company need a data scientist? You are wrong if you haven't hired a data engineer yet. You need to have the ability to get data before making sense of it. If you only deal with small datasets, you may be able to get by with entering some numbers into a spreadsheet. As the data increasing in size, data engineering becomes a sophisticated discipline in its own right.

- **Engineering: the process of making everything else possible**

Data engineering mainly involves in building the data pipeline infrastructure. In the (not that) old day, when data is stored on local servers, computers or other devices, building the data infrastructure can be a humongous IT project which involves not only the software but also the hardware that used to store the data and perform ETL process. As the development of cloud service, data storage and computing on the cloud becomes the new norm. Data engineering today at its core is software engineering. Ensuring maintainability through modular, well-commented code and version control is fundamental.

---

[3]This is based on Industry recommendations for academic data science programs: https://github.com/brohrer/academic_advisory[4] with modifications. It is a collection of thoughts of different data scientist across industries about what a data scientist does, and what differentiates an exceptional data scientist.

(1) Data environment

Design and set up the environment to support data science workflow. It may include setting up data storage in the cloud, Kafka platform, Hadoop and Spark cluster etc. Each company has its unique data condition and needs. So the environment will be different depending on size of the data, update frequency, complexity of analytics, compatibility with the backend infrastructure and (of course) budget.

(2) Data management

Automated data collection is a common task which includes parsing the logs (depending on the stage of the company and the type of industry you are in), web scraping, API queries, and interrogating data streams. Determine and construct data schema to support analytical and modeling need. Use tools, processes, guidelines to ensure data is correct, standardized and documented.

(3) Production

If you want to integrate the model or analysis into the production system, then you have to automate all data handling steps. It involves the whole pipeline from data access to preprocessing, modeling and final deployment. It is necessary to make the system work smoothly with all existing stacks. So it requires to monitor the system through some sort of robust measures, such as rigorous error handling, fault tolerance, and graceful degradation to make sure the system is running smoothly and the users are happy.

- **Analysis – the process of turning raw information into insights in a fast way**

(1) Domain knowledge

Domain knowledge is the understanding of the organization or industry where you apply data science. You can't make sense of data without context, such as what are the important metric for

this kind of business, what are the business questions, what type of data they have and what the data represents, how to translate a business need to a data problem, what has been tried before and with what results, what are the accuracy-cost-time trade-offs, how can things fail, what other factors are not accounted, what are the reasonable assumptions and what are faulty. In the end, domain knowledge helps you to deliver the results in an audience-friendly way.

(2)   Exploratory analysis

This type of analysis is about exploration and discovery. Rigor conclusion is not a concern which means the goal is to get insights driven by correlation not causation. The later one requires statistical skills and hence more expensive. Instead this role will help your team look at as much data as possible so that the decision-makers can get a sense of what's worth further pursuing. It often involves different ways to slice and aggregate data. An important thing to note here is that you should be careful not to get conclusion beyond the data. You don't need to write gorgeous, robust code to perform well in this role.

(3)   Story telling

Storytelling with data is key to deliver the insights and drive better decision making. It is the art of telling people what the numbers actually signify. It usually requires data summarization, aggregation and visualization. It is important to answer the following questions before you begin down the path of creating a data story: * Who are your audience? * What do you want your audience to know or do? * How can you use data to help make your point?

- **Modeling – the process of diving deeper into the data to discover the pattern we don't easily see**

Even fancy machine learning model is the first thing comes to mind when people think about data science, unfortunately, in industry,

it occupies the smallest part of data scientist's time. Nevertheless, it is a powerful set of tools.

(1)   Supervised learning

In supervised learning, each observation of the predictor measurement(s) corresponds to a response measurement. There are two flavors of supervised learning: regression and classification. In regression, the response is a real number such as the total net sales in 2017, or the yield of corn next year. The goal is to approximate the response measurement as much as possible. In classification, the response is a class label, such as dichotomous response such as yes/no. The response can also have more than two categories, such as four segments of customers. A supervised learning model is a function that maps some input variables with corresponding parameters to a response y. Modeling tuning is to adjust the value of parameters to make the mapping fit the given response. In other words, it is to minimize the discrepancy between given responses and the model output. When the response y is a real value, it is intuitive to define discrepancy as the squared difference between model output and given the response. When y is categorical, there are other ways to measure the difference, such as AUC or information gain.

(2)   Unsupervised learning

In unsupervised learning, there is no response variable. For a long time, the machine learning community overlooked unsupervised learning except for one called clustering. Moreover, many researchers thought that clustering was the only form of unsupervised learning. One reason is that it is hard to define the goal of unsupervised learning explicitly. Unsupervised learning can be used to do the following: * Identify a good internal representation or pattern of the input that is useful for subsequent supervised or reinforcement learning, such as finding clusters. * It is a dimension reduction tool that is to provide compact, low dimensional representations of the input, such as factor analysis. * Provide

a reduced number of uncorrelated learned features from original variables, such as principal component regression.

(3)　Customized model development

In most of the cases, you just need to use the out of the box algorithms to solve the problem. But in some situations, there isn't enough data to use machine learning model, or the question doesn't fit neatly in the specifications of existing tools, or the model needs to incorporate some prior domain knowledge . You may need to develop new models to accommodate the subtleties of the problem at hand. For example, people use bayesian models to include domain knowledge as prior distribution.

What others?

There are some common skills to have regardless the role people have in data science.

- **Data Preprocessing: the process nobody wants to go through yet nobody can avoid**

No matter what role you hold in data science team, you will have to do some data cleaning which tend not to be the favorite part of anyone's job. Data preprocessing is the process of converting raw data into clean data that is proper for use.

(1)　Data preprocessing for data engineer

Getting data together from different sources and dumping them to a Data Lake, a dumping ground of amorphous data, is far from the data schema analyst and scientist would use. A data lake is a storage repository that stores a vast amount of raw data in its native format, including XML, JSON, CSV, Parquet, etc. It is a data cesspool rather than data lake. It is data engineer's job to get a clean schema out of the data lake by transforming and formatting the data. Some common problems to resolve are:

- Enforce new tables' schema to be the desired one

- Repair broken records in newly inserted data
- Aggregate the data to form the tables with a proper granularity

(2) Data preprocessing for data analyst and scientist

Not just for data engineer, it also occupies a large fraction of data analyst and scientist's working hours too. A facility and a willingness to do these tasks are a prerequisite for a strong data scientist. If you are lucky as a data scientist, you may end up spending 50% of your time doing this. If you are like most of us, you will spend over 80% of your working hours wrangling data.

The data you get can still be very rough even it is from a nice and clean database that engineers set up. Dates and times are notorious for having many representations and time zone ambiguity. You may also get market survey responds from your clients in an excel file where the table title could be multi-line, or the format does not meet the requirements, such as using 50% to represent the percentage rather than 0.5. So in many cases, you need to set the data to be the right format before moving on to analysis.

Even the data is in the right format, there are other issues to solve before or during analysis. For example, variables can have missing values. Knowledge about the data collection process and what it will be used for is necessary to decide a way to handle the missing. Also, different models have different requirements on the data. For example, some model may require the variables are of consistent scale; some may be susceptible to outliers or collinearity, some may not be able to handle categorical variables and so on. The modeler has to preprocess the data to make it proper for the specific model.

Most of the people in data science today focus on one of the tracks. A small number of people are experts of two tracks. People that are proficient in all three? They are unicorns!

1.2   What should data science do?

1.2.1   Let's dream big

Here is my two points for the question:

- Make human better human by alleviating bounded rationality and minimize politics/emotion (rather than make machine more like human)
- Strive for the "democratization" of data as legally possible: empower everyone in the organization to acquire, process, and leverage data in a timely and efficient fashion

I know it is vague. Behold, I am going to explain more.

It's easy to pretend that you are data driven. But if you get into the mindset to collect and measure everything you can, and think about what the data you've collected means, you'll be ahead of most of the organizations that claim to be data driven. If you know the difference between "data driven" and "data confirmed", you'll be sailing at the right direction. What on earth is the difference?

Imagine that you are buying something online and you need to decide whether or not to trust the product without seeing it physically. You see the average rating is 4.1 out of 5.0. Is this a good score? It depends on your subconscious decision. If you really need the thing, you may happily cheer "It is more than 4.0!". If you are still not sure whether you need it, you can't help to check the few low rating reviews and tell yourself "look at those 1-star reviews". Sounds familiar? Psychologists call it confirmation bias.

Confirmation bias is the tendency to search for, interpret, favor, and recall information in a way that confirms one's preexisting beliefs or hypotheses [Wikipedia]

So if you use data to feel better (confirm) decisions/assumptions that are already made before you analyze the data, that is "data confirmed". A clear sign of confirmation bias is when you go back to tinker the definition of your metic because the current result is not impressive. However, this bias is not always easy to see. It is not only misleading but also expensive. Because it could take data science team days of toil to boil everything down to that magic number and put the result on the report. Data scientists are not totally immune to the bias either. Good news is that there is antidote to confirmation bias.

Antidote 1: Do the brainstorming of data definition and set the goal in advance and resist temptation to move them later. In other words, the decision makers have to set decision criteria and the boundary up front in your data science project.

Antidote 2: Data democratization. Keep in mind that data isn't just for the professionals or a small group of people in the organization that are "key decision makers". Everyone should be able to get access to and look at the data (as much as legally possible). In that way, there will be more eyes on the decision.

The way data science can help is to provide a sound data framework and necessary training for the organization to access data with least amount of pain. Also be clear about the data definition and documentation. Data science holds the responsibility for data stewardship in the organization with high integrity. (there is data science for social good which is data science's responsibility for outside the organization but we are not going to discuss that here)

That is still very abstract, I hear you. Now, Let's be more specific...

1.2.2 What kind of questions can data science solve?

1.2.2.1 Prerequisites

Data science is not a panacea and there are problems data science can't help. It is best to make a judgment as early in the analytical cycle as possible. Tell your clients honestly and clearly when you

think data analytics can't give the answer they want. What kind of questions can data science solve?

1.   Your question needs to be specific enough

Look at two examples:

- Question 1: How can I increase product sales?
- Question 2: Is the new promotional tool introduced at the beginning of this year boosting the annual sales of P1197 in Iowa and Wisconsin? (P1197 is an impressive corn seed product from DuPont Pioneer)

It is easy to see the difference between the two questions. Question 1 is a grammatically correct question, but it is proper for data analysis to answer. Why? It is too general. What is the response variable here? Product sales? Which product? Is it annual sales or monthly sales? What are the candidate predictors? You nearly can't get any useful information from the questions. In contrast, question 2 is much more specific. From the analysis point of view, the response variable is clearly "annual sales of P1197 in Iowa and Wisconsin". Even we don't know all the predictors, but the variable of interest is "the new promotional tool introduced early this year." We want to study the impact of the promotion of sales. You can start from there and move on to figure out other variables need to include in the model by further communication.

As a data scientist, you may start with something general and unspecific like question 1 and eventually get to question 2. Effective communication and in-depth domain knowledge about the business problem are essential to convert a general business question into a solvable analytical problem. Domain knowledge helps data scientist communicate with the language the other people can understand and obtain the required information.

However, defining the question and variables involved won't guarantee that you can answer it. For example, I encountered this situation with a well-defined supply chain problem. My client asked

me to estimate the stock needed for a product in a particular area. Why can't this question be answered? I tried fitting a Multivariate Adaptive Regression Spline (MARS) model and thought I found a reasonable solution. But it turned out later that the data my client gave me was inaccurate. In this case, only estimates rather than actual values of past supply figures were available and there was no way to get accurate data. The lesson lends itself to the next point.

2. You need to have sound and relevant data

One cannot make a silk purse out of a sow's ear. Data scientists need data, sound and relevant data. The supply problem is a case in point. There was relevant data, but not sound. All the later analytics based on that data was a building on sand. Of course, data nearly almost have noise, but it has to be in a certain range. Generally speaking, the accuracy requirement for the independent variables of interest and response variable is higher than others. In question 2, it is data related to the "new promotion" and "sales of P1197".

The data has to be helpful for the question. If you want to predict which product consumers are most likely to buy in the next three months, you need to have historical purchasing data: the last buying time, the amount of invoice, coupons and so on. Information about customers' credit card number, ID number, the email address is not going to help.

Often the quality of the data is more important than the quantity, but the quantity cannot be overlooked. In the premise of guaranteeing quality, usually the more data, the better. If you have a specific and reasonable question, also sound and relevant data, then congratulations, you can start playing data science!

1.2.2.2 Problem type

Many of the data science books classify the various models from a technical point of view. Such as supervised vs. unsupervised

models, linear vs. nonlinear models, parametric models vs. non-parametric models, and so on. Here we will continue on "problem-oriented" track. We first introduce different groups of real problems and then present which models can be used to answer the corresponding category of questions.

1. Description

The basic analytic problem is to summarize and explore a data set with descriptive statistics (mean, standard deviation, and so forth) and visualization methods. It is the simplest problem and yet the most crucial and common one. You will need to describe and explore the dataset before moving on to more complex analysis. In the problem such as customer segmentation, after you cluster the sample, the next step is to figure out the profile of each class by comparing the descriptive statistics of the various variables. Questions of this kind are:

- How does the annual income distribute?
- Are there outliers?
- What are the mean active days of different accounts?

Data description is often used to check data, find the appropriate data preprocessing method, and demonstrate the model results.

2. Comparison

The first common problem is to compare different groups. Such as: Is A better in some way than B? Or more comparisons: Is there any difference among A, B, and C in a particular aspect? Here are some examples:

- Are males more inclined to buy our products than females?
- Are there any differences in customer satisfaction in different business districts?
- Do soybean carrying a particular gene have higher oil content?

For those problems, it is usually to start exploring from the summary statistics and visualization by groups. After a preliminary visualization, you can test the differences between treatment and control group statistically. The commonly used statistical tests are chi-square test, t-test, and ANOVA. There are also methods using Bayesian methods. In biology industry, such as new drug development, crop breeding, mixed effect models are the dominant technique.

3. Clustering

Clustering is a widespread problem, which is usually related to classification. Clustering answers questions like:

- Which customers have similar product preference?
- Which printer performs a similar pattern to the broken ones?
- How many different themes are there in the corpus?

Note that clustering is unsupervised learning. The most common clustering algorithms include K-Means and Hierarchical Clustering.

4. Classification

Usually, a labeled sample set is used as a training set to train the classifier. Then the classifier is used to predict the category of a future sample. Here are some example questions:

- Who is more likely to buy our product?
- Is the borrower going to pay back?
- Is it spam?

There are hundreds of classifiers. In practice, we do not need to try all the models but several models that perform well generally.

5. Regression

In general, regression deals with the problem of "how much is it?" and return a numerical answer. In some cases, it is necessary to coerce the model results to be 0, or round the result to the nearest integer. It is the most common problem.

- What will be the temperature tomorrow?
- What is the projected net income for the next season?
- How much inventory should we have?

1.3 Structure data science team

During the past decade, a huge amount of data has become available and readily accessible for analysis in many companies across different business sectors. The size, complexity, and speed of increment of data suddenly beyond the traditional scope of statistical analysis or BI reporting. To leverage the big data, do you need an internal data science team to be a core competency, or can you outsource it? The answer depends on the problems you want to solve using data. If the problems are critical to the business, you can't afford to outsource it. Also, each company has its own business context and hence needs new kinds of data or or use the results in novel ways. Being a data driven organization requires cross organization commitments to identify what data each department needs to collect, establish the infrastructure and process for collecting and maintaining that data, and the way to deliver analytical results. Unfortunately, it is unlikely that an off-the-shelf solution will be flexible enough to adapt to the specific business context. So most of the companies establish their own data science team.

Where should data science team fit? Much has been written about different ways data science function fit in the organization. In general, data science team is organized in three ways.

(1) A standalone team

Data science is an autonomous unit that is parallel to the other organizations (such as engineering, product etc.) and the head of data science reports directly to senior leadership, ideally to the CEO or at least to someone who understands data strategy and is willing to invest to give it what it needs. The advantages of this type of data organization are:

- Data science team has autonomy and is well positioned to tackle whatever problems it deems important to the whole company.
- It is advantageous for people in data science team to share knowledge and grow professionally.
- It provides a clear career path for data science professionals and shows the company treats data as a first-class asset. So it tends to attract and retain top talent people.

The biggest concern of this type of organization is the risk of marginalization. Data science only has value if data drives action which requires collaboration among data scientists, engineers, product managers and other business stakeholders across the organization. If you have a standalone data science team, it is critical to choose a data science leader who is knowledgable about the applications of data science in different areas and also has strong interdiscipline communication skills. The head of data science needs to build strong collaboration with other departments.

Also, as companies grow, each department prefers to be self-sufficient and tries to hire data own analytical personal under different titles even when they can get support from the data science team. This is why it is unlikely for an already mature company to have a standalone data science team. If you start your data science team in the early stage as a startup, it is important that the CEO sets a clear vision from the beginning and sends out strong message to the whole company about accessing data support.

  (2)   An embedded model

There is still a head of data science but his/her role is mostly a

hiring manager and coach and he/she may report to a senior manager in IT department. The data science team brings in talented people and farms them out to the rest of the company. In other words, it gives up autonomy to ensure utility. The advantages are:

- Data science is closer to its applications.
- There is still a data science group so it is easy to share knowledge.
- It has high flexibility to allocate data science resource to the rest of the company.

However, there are also concerns.

- It brings difficulty to the management since the lead of the designated team is not responsible for data science professionals' growth and happiness while the data science managers are not directly vested in their work.
- Data scientists are second-class citizens everywhere and it is hard to attract and retain top talent.

(3) Integrated team

There is no data science team. Each team hires its own data science people. For example, there may be a marketing analytics group consisting of data engineer, data analyst and data scientists. The team leader is a marketing manager who has an analytical mind and deep business knowledge. The advantages are obvious.

- Data science resource aligns with the organization very well
- Data science professionals are first-class members and valued in their own team. The manager is responsible for data science professionals' growth and happiness.
- The insights from data are easily put into actions.

It works well in the short term for both the company and the data science hires. However, there are also many concerns.

- It sacrifices the professional growth of data science hires since they work in silos and specialize in specific application. It is also difficult to share knowledge across different applied areas.
- It is harder to move people around since they are highly associated with a specific function in the organization.
- There is no career path for data science people and so it is difficult to retain talent.

There is not an universal answer for the best way to organize data science team. It depends on the answer of many other questions. How important do you think the data science team is for your company? What is the stage of your company when you start to build data science team? Are you a startup or a relatively mature company? Data science has its own skillset, workflow, tooling, integration process and culture. If it is critical to your organization, it is the best not to bury it under any part of the organization. Otherwise data science will inevitably only serve the need for specific branch of the organization and it also impedes data democratization across the organization. How valuable it is to use data to tell the truth, how dangerous it is to use data to affirm existing opinions. No matter which way you choose, be aware of both sides of the coin. If you are looking for a data science position, it is important to know where the data science team fits in the company.

2

Soft Skills for Data Scientists

2.1 Comparison between Statistician and Data Scientist

Statistics as a scientific area can be traced back to 1749 and statistician as a career has been around for hundreds of years with well-established theory and application. Data Scientist becomes an attractive career for only a few years along with the fact that data size and variety beyond the traditional statistician's toolbox and the fast-growing of computation power. Statistician and data scientist have a lot of in common, but there are also significant differences.

Statistician	Data Scientist
Data is given (i.e. in text files)	Need to get data (i.e. for database)
Focus on modeling	Focus on results
Bring data to model	Bring model to data
Further away from production system	Embedded in production system
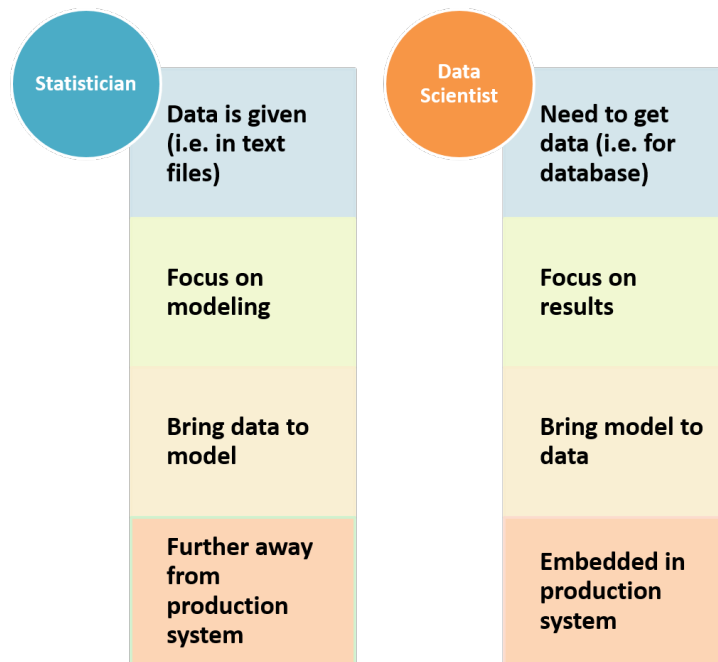
Both statistician and data scientist work closely with data. For the traditional statistician, the data is usually well-formatted text files with numbers and labels. The size of the data usually can be fitted in a PC's memory. Comparing to statisticians, data scientists need to deal with more varieties of data:

- well-formatted data stored in a database system with size much larger than a PC's memory or hard-disk;
- huge amount of verbatim text, voice, image, and video;
- real-time streaming data and other types of records.

One unique power of statistics is to make statistical inference based on a small set of data. Statisticians spend most of their time developing models and don't need to put too much effort on data cleaning. Today, data is relatively abundant, and modeling is only part of the overall effort, often a small part. Due to the active development of some open source communities, fitting models is not too far from button pushing. Data scientists instead spend lot of time preprocessing and wrangling the data before feeding them to the model.

Different from statisticians, data scientists often focus on delivering actionable results and sometimes need to fit model on the cloud. The data can be too large to read in laptop. From the entire problem-solving cycle, statisticians are usually not well integrated with the production system where data is obtained in real time; while data scientists are more embedded in the production system and closer to the data generation procedures.

2.2   Beyond Data and Analytics

Data scientists usually have a good sense of data and analytics, but data scientist project is more than that. A data science project may involve people with different roles, especially in a big company:

- a business owner or leader to identify business value;
- program manager to ensure the data science project fits into the overall technical program development and coordinate all parties to set periodical tasks so that the project meets the preset milestones and results;
- data owner and computation resource and infrastructure owner from the IT department;
- dedicated team to make sure the data and model are under model governance and privacy guidelines;
- a team to implement, maintain and refresh the model;
- multiple rounds of discussion of resource allocation among groups (i.e., who pay for the data science project).

Effective communication and in-depth domain knowledge about the business problem are essential requirements for a successful data scientist. A data scientist may interact with people at various levels from senior leaders who set the corporate strategies to front-line employees who do the daily work. A data scientist needs to have the capability to view the problem from 10,000 feet above the ground, as well as down to the detail to the very bottom. To convert a business question into a data problem, a data scientist needs to communicate using the language the other people can understand and obtain the required information.

In the entire process of data science project defining, planning, executing and implementing, every step involves the data scientist to ensure people correctly define the business problem and reasonably evaluate the business value and success. Corporates are investing heavily in data science and machine learning with a very high expectation of return.

However, it is easy to set unrealistic goal and wrongly estimate the business impact. The data scientist lead should navigate the discussions to make sure the goal can be backed by data and analytics. Many data science projects over promise and are too optimistic on the timeline. These projects eventually fail by not delivering the preset business impact within the timeline. As data scientists, we need to identify these issues early in the stage and communicate

with the entire team to make sure the project has a realistic deliverable and timeline. The data scientist team also need to work closely with data owners to identify relevant internal and external data source and evaluate the quality of the data; as well as working closely with the infrastructure team to understand the computation resources (i.e. hardware and software) available for the data science project.

2.3   Three Pillars of Knowledge

(1)   Analytics knowledge and tool sets

A successful data scientist needs to have a strong technical background in data mining, statistics and machine learning. The in-depth understanding of modeling with the insight about data enable a data scientist to convert a business problem to a data science problem.

(2)   Domain knowledge and collaboration

A successful data scientist needs some domain knowledge to understand the business problem. For any data science project, the data scientist need to collaborate with other team members and effective communication and leadership skills are critical, especially when you are the only data person in the room and you need to decide with uncertainty.

(3)   (Big) data management and (new) IT skills

The last pillar is about computation environment and model implementation in a big data platform. This used to be the most difficult one for a data scientist with statistics background (i.e. lack computer science or programming skills). The good news is that with the rise of cloud computation big data platform, it is easier for a statistician to overcome this barrier.

• Understand and prepare data
• Statistical methods and problem solving
• Machine learning and data mining experience

Analytics
knowledge
and tool sets

(Big) Data
management
and (new) IT
skills

Domain
Knowledge &
Collaboration

• Unstructured data
• Big data infrastructure
• Database and data retrieval
• Software: Hadoop, SQL, R, Python …
• "Cloud" Solution

• Teamwork
• Problem definition
• Communication skill
• Strategic planning & execution

2.4 Data Science Project Cycle

A data science project has various stages. Many textbooks and blogs focus on one or two specific stages and it is rare to see the end-to-end life cycle of data science projects. In fact, to get a good grasp of the end-to-end cycle requires many years of experience of doing real-world data science. We will share our opinions on that in this section. Seeing a holistic picture of the whole cycle helps you to better prepare for real-world applications.

2.4.1 Types of Data Science Projects

People often use data science project to describe any project that uses data to solve a business problem, including traditional business analytics or data visualization. Here we limit our discussion of data science projects that involve data and some statistical or machine learning models. The business problem itself gives us the flavor of the project, data is the raw ingredient to start with, and

the model makes the dish. Different types of data science projects can be determined by the types of data used and the final model development and implementation.

2.4.1.1 Offline and online Data

There are offline and online data. Offline data are historical archived data stored in databases or data warehouses. With the development of data storage, the cost to store a large amount of data is cheap and offline data are very rich in general (for example website may track and store each individual user's mouse position, click and typing information while the user is visiting the website). Offline data is usually stored in a distributed system and it can be extracted in batch as raw materials to create features that can be used in model training. Online data are real-time information that can be feed to models to make automatic actions. Real-time information can changes frequently such as the keywords a customer is searching for. Capturing and using real-time online data requires the integration of machine learning to the production infrastructure. It used to be a steep learning curve for data scientists, but the cloud infrastructure makes it much easier.

2.4.1.2 Offline training and offline application

This type of data science project is for a specific business problem which needs to be solved once or multiple times. But the dynamic nature of the business problem requires substantial work every time. One example of such a project is "whether a new workflow is going to improve efficiency." In this situation, we often use offline internal and external data, build models, and deliver the final results as a report to answer the specific business question. It is similar to the traditional business intelligence project but with more focus on data and model. Sometimes the data size and model complexity are beyond the capacity of a single computer. So you need to use distributed storage and computation. Since the model is based on the historical data and the output is a report, there is no need for real-time execution. Usually, there is no run-time constraint on the machine learning model unless the model is run-

ning beyond a reasonable time frame such as a few hours or a few days. We can call this type of data science project "offline training, offline application" project.

2.4.1.3 Offline training and online application

Another type of data science project is to use offline data for training and apply the trained model to real-time online data in the production environment. One example of such a project is "using historical data to train a personalized advertisement model, and then provides real-time ad recommendation when customers visit the website." The model is trained based on offline data, and then use a customer's online real-time data as features to run the model in real time to provide an automatic action. The model training is very similar to the "offline training, offline application" project, but as the trained model will be put to production, there are specific requirements such as features used in the offline training have to be available online in real time, and the online run-time of the model has to be short enough without impacting user experience. In most cases, data science projects in this category create continuous and scalable business value. We will use this type of data science project to describe the project cycle.

2.4.1.4 Online training and online application

For some business problems, it is so dynamic that even yesterday's data is out of date. For such cases, we can use online data to train the model and then applying it in real time. We call this type of data science project "online training, online application." This type of data science project requires high automation and low latency.

2.4.2 At the Planning Stage

To ensure a successful data science project, a data-driven and fact-based planning stage is essential. With the recent big data and data science hype, there is a high demand for data science projects to create business value across different business sectors. Often times,

these data science project proposals are initiated by the leaders of an organization. This top-down style data science projects usually have high visibility with certain human and computation resources pre-allocated. However, it is crucial to understand the business problem first and align the goal across different teams including

(1) the business team which may include members from the business operation team, business analyst, insight and reporting team;

(2) technology team which may include members from database and data warehouse team, data engineering team, infrastructure team, core machine learning team, and software development team; (3) project management team which may include program management team and product management team depending on the scope of the data science project.

To start the conversation, we can ask the following questions to everyone in the team:

- What are the pain points in current business operation?
- What data are available and how is the quality and quantity of the data?
- What might be the most significant impacts of a data science project?
- Are there any negative impact to other teams?
- What computation resources are available for model training and model execution?
- Can we define key metrics to compare and quantity business value?
- Are there any data security, privacy and legal concerns?
- What are the desired milestones, check points and timeline?
- Is the final application online or offline?
- Are the data online or offline?

It is likely to have a series of intense meetings and heated discussions to frame the project to a reasonable scope. After the planning

stage, we should be able to define a set of key metrics related to the project, identify some offline and online data sources, request needed computation resources, draft tentative timeline and milestones, and form a team of data scientist, data engineer, software developer, project manager and members from business operation. Data scientists should play a major role in these discussions. If data scientist is not leading the data science project formulation, it is very likely the entire project will not reach the timeline and milestones.

2.4.3   At the Modeling Stage

Even though at the planning stage we already set some strategy, milestone, and timeline, data science projects are dynamic in nature and there could be uncertainties along the road. As a data scientist, clearly communicate any newly encountered difficulties during the modeling stage to the entire team is essential to keep the data science project progress. With the available data source identified at the planning stage, data clearing, data wrangling, and exploratory data analysis are great starting points toward modeling. Meanwhile, abstracting the business problem to be a set of statistical and machine learning problems is an iterative process. It is rare that business problems can be solved by using just one statistical or machine learning model. The ability to use a sequence of methods to decompose the business problem is one of the key responsibility for a senior data scientist. The process requires iterative rounds of discussions with the business team and data engineering team based on the new learning from each iteration. Each iteration includes both data related and model related part.

2.4.3.1   Data related

Data cleaning, data preprocessing and feature engineering are closely related procedures. The goal of these data-related procedures is to create usable variables or features for statistical and machine learning models. One important aspect of data related procedures is to make sure the data source we are using is a good

representation of the situation where the final trained model will be applied. The exact same representation is rarely possible, and reasonable approximation is totally fine to start with. A data scientist has to be clear on the assumptions and communicate with the entire team the limitations of biased data and quantify its impact on the application. In data related part, sometimes the available data is not that relevant to the business problem we want to solve, and we have to collect more data.

2.4.3.2 Model related

There are different types of statistical and machine learning models, such as supervised learning, unsupervised learning, and causal inference. For each type, there are various algorithms, libraries, or packages readily available. To solve a business problem, you sometimes need to piece together a few methods at the model exploring and developing stage. This stage also includes model training, validation, and testing to make sure the model works well in the production environment; i.e., it is not overfitting and can be generalized. The model selection follows Occam's razor, which is to choose the simplest among a set of compatible models. Before you try complicated models, it is a good practice to get some benchmark by additional business rules, common sense decision, or standard models (such as random forest for classification).

2.4.4 At the Production Stage

For offline application data science projects, the end product is often a detailed report with model result and output. However, for online application projects, a trained model is just halfway from the finish line. The offline data is stored and processed in a totally different environment from the online production environment. Building the online data pipeline and implementing machine learning models in a production environment requires lots of additional work. Even though recent advance in cloud infrastructure lowers the barrier dramatically, it still takes effort to implement an

offline model in the online production system. Before you promote the model to production, there are two more steps to go:

1. shadow mode
2. A/B testing

A **shadow mode** is like an observation period when the data pipeline and machine learning models run as it is fully functional, but we only record the model output without any actions. Some people call it proof of concept (POC). During POC, people frequently check the data pipeline and model and detect bugs such as a timeout or missing features, version conflict (for example python 2 v.s. python 3), data type mismatch, etc.

Once the online model passes the shadow mode, **A/B testing** is the next stage. During A/B testing, all the incoming observations are randomly separated into two groups: control and treatment. The control group is going to skip the machine learning model, while the treatment group is going through the machine learning model. After that, people monitor a list of pre-defined key metrics during a specific time period to compare the control and treatment groups. The differences in these key metrics determine whether the machine learning model provides business value or not. Real applications can be complicated. For example, there can be multiple treatment groups, or hundreds, even thousands of A/B testing running by different teams at any given time.

Once the A/B testing shows that the model provides significant business value, then you can put it into full production. It is ideal that the model runs as expected and continues to provide scalable values. However, the business can change and a machine learning model works now can break tomorrow, and features available now may not be available tomorrow. You need a monitoring system to automatically notify us when one or multiple features change. When the model performance degrades below a pre-defined a level, you need to fine-tune the parameters and thresholds, re-train the model with more recent data, add or remove features to improve model performance. Eventually, any model will fail or retire at some time.

2.4.5 Summary

Data science end-to-end project cycle is a complicated process which requires close collaboration among many teams. Data scientist, maybe the only scientist in the team, has to lead the planning discussion and model development based on data available and clearly communicate key assumptions and uncertainties with the entire team. A data science project may fail at any stage, and a clear end-to-end cycle view of the project helps avoid some mistakes.

2.5 Common Mistakes in Data Science

Data science project can go wrong at different stages in many ways. Most textbooks and online blogs focus on technical mistakes about machine learning model, algorithm or theory, such as including outliers and overfitting. It is important to avoid these technical mistakes. However, there are common systematic mistakes across data science projects that are rarely discussed in textbooks. To summarize these mistakes, people need to do real-world data science projects. In this section, we describe these common mistakes in detail so that readers can proactively identify and avoid these systematic mistakes in their own data science projects.

2.5.1 Problem Formulation Stage

The most challenging part of a data science project is problem formulation. Data science project stems from pain points of the business. The draft version of the goal of the project is relatively vague without much quantification or is the gut feeling of the leadership team. Often there are multiple teams involved in the initial project formulation stage and they have different views. It is easy to have malalignment across different teams such as resource allocation, milestone deliverable, and timeline. At the prob-

lem formulation stage, data science team members with technical background sometimes are not even invited to the initial discussion. It sounds ridiculous, but sadly true that a lot of resources are spent on **solving the wrong problem,** the number one systematic common mistake in data science. Formulating a business problem into the right data science project requires an in-depth understanding of the business context, data availability and quality, computation infrastructure, and methodology to leverage the data to quantify business value.

We have seen people over promise about business value all the time, another common mistake that is going to fail the project at the beginning. With the hype of big data and machine learning, leaders across industries often have unrealistic high expectation on data science. It is especially true during enterprise transformation when there is a strong push to adopt new technology to get value out of the data. The unrealistic expectations are based on assumptions that are way off chart without checking the data availability, data quality, computation resource, and current best practices in the field. Even there is some exploratory analysis by the data science team at the problem formulation stage, project leaders sometimes ignore their data-driven voice.

These two systematic mistakes undermine the organization's data science strategy. The higher the expectation, the bigger the disappointment when the project cannot deliver business value. Data and business context are essential to formulate the business problem and set reachable business value. It helps to avoid the mistakes by having a strong data science leader with a broad technical background and let data scientist coordinate and drive the problem formulation and set realistic goals based data and business context.

2.5.2   Problem Planning Stage

Now suppose the data science project is formulated correctly with a reasonable expectation on the business value. The next step is to plan the project such as allocating resources, setting up milestones and timeline, and defining deliverable. In most cases, there are

project managers to coordinate different teams that are involved in the project and use agile project management tools similar to those in software development. Unfortunately, the project management team may not have experience with data science projects and hence fail to account for the uncertainties at the planning stage. The fundamental difference between data science projects and other projects lead to another common mistake: **too optimistic about the timeline**. For example, data exploratory and preparation may take 60% to 80% of the total time for a given data science project, but people often don't realize that.

When there are lots of data already collected across the organization, people assume you have enough data for everything. It leads to the mistake: t**oo optimistic about data availability and quality**. What you need is not "big data", but data that can help you solve the problem. The data available may be low quality and you need to put substantial efforts to clean the data before you can use it. There are "unexpected" efforts to bring the right and relevant data for a specific data science project. To ensure smooth delivery of data science projects, you need to account for the "unexpected" work at the planning stage. We all know data pre-processing and feature engineering is usually the most time-consuming part of a data science project. However, people outside data science are not aware of it and we need to educate other team members and the leadership team.

### 2.5.3	Modeling Stage

Finally, you start to look at the data and fit some models. One common mistake at this stage is unrepresentative data. The model trained using historical data may not generalize to the future. There is always a problem of biased or unrepresentative data. As a data scientist, we need to use data that are closer to the situation where the model is going to apply and quantify the impact of model output in production. Another mistake at this stage is overfitting and obsession for complicated models. Now we can easily get hundreds or even thousands of features and the machine learning models are getting more complicated. People can use open source

libraries to try all kinds of models. People are sometimes obsessed with complicated models instead of using the simplest among a set of compatible models.

The data used to build the models is always biased or unrepresentative to some extent, simpler models are better to generalize and it has a higher chance to provide consistent business value once the model passes the test and is finally implemented in the production environment. It is possible that you can not use the existing data and methods to solve the business problem. You can try to collect more data, do feature engineering, or create your own models. However, if there is a fundamental gap between data and the business problem, the data scientist has to make the tough decision to unplug the project. On the other hand, data science projects usually have high visibility and may be initiated by senior leadership. Even the data science team provide enough evidence that they can't deliver the expected business value, people may not want to stop the project which leads another common mistake at modeling stage: **take too long to fail**. The earlier we can stop a failing project, the better. Because we can put valuable resources to other promising projects. It damages the data science strategy and everyone will be hurt by a long data science project that is doomed to fail.

2.5.4 Production Stage

Now suppose you have found a model that works great for the training and testing data. If it is an online application, you are halfway. The next is to put the model in production, which sounds like alien work for a data scientist. It is true that the data engineering team can help with model production. However, as a data scientist, you need to know the potential mistakes at this stage. One big mistake is **missing A/B testing** and assuming that the model performance at model training/testing stays the same in the production environment. Unfortunately, the model trained and evaluated using historical data nearly never performs the same in the production environment. The data used in the offline training maybe significant different from online data and the business

context may have changed. If possible, machine learning models in production should always go through A/B testing to evaluate performance.

In the model training stage, people usually focus on model performance, such as accuracy without paying too much attention to the model execution time. When a model runs online in real time, the total run time for each instance (i.e., model latency) should not impact the customer's user experience. Nobody wants to wait for even one second to see the results after click the "Search" button. In the production stage, feature availability is crucial to run a real-time model. Engineering resources are essential for model production. However, in traditional companies, it is common that a data science project **fail to scale in real time applications** due to lack of computation capacity, engineering resources, or non-tech culture and environment.

As the business problem evolve rapidly, the data and model in the production environment need to change accordingly or the performance of the model deteriorates over time. The online production environment is more complicated than model training and testing, for example, you pull online features from different resources, and some features may be missing at a specific time; the model may run into time out zone, and there are tons of different software and data exceptions that may happen. We need regular checkup during the entire life of the model cycle from implementation to retirement. Unfortunately, people often don't set the monitoring system for data science projects, and it is another common mistake: **missing necessary online checkup**. It is essential to set a monitoring dashboard and automatic alarms, create model tuning, re-training, and retirement plans.

2.5.5 Summary

The data science project is a combination of art and engineering. A data science project may fail in different ways. However, if we put data and business context at the center of the project, get familiar with the data science project cycle and proactively identify and

avoid these potential mistakes, the data science project can provide significant business value. Here is the summary of the mistakes:

- Solving the wrong problem
- Over promise on business value
- Too optimistic about the timeline
- Too optimistic about data availability and quality
- Unrepresentative data
- Overfitting and obsession for complicated models
- Take too long to fail
- Missing A/B testing
- Fail to scale in real-time applications
- Missing necessary online checkup

3

Introduction to The Data

Before tackling analytics problem, we start by introducing data to be analyzed in later chapters.

3.1 Customer Data for Clothing Company

Our first data set represents customers of a clothing company who sells products in physical stores and online. This data is typical of what one might get from a company's marketing data base (the data base will have more data than the one we show here). This data includes 1000 customers:

1. Demography

   - `age`: age of the respondent
   - `gender`: male/female
   - `house`: 0/1 variable indicating if the customer owns a house or not

2. Sales in the past year

   - `store_exp`: expense in store
   - `online_exp`: expense online
   - `store_trans`: times of store purchase
   - `online_trans`: times of online purchase

3. Survey on product preference

It is common for companies to survey their customers and draw

insights to guide future marketing activities. The survey is as below:

How strongly do you agree or disagree with the following statements:

1. Strong disagree
2. Disagree
3. Neither agree nor disagree
4. Agree
5. Strongly agree

- Q1. I like to buy clothes from different brands
- Q2. I buy almost all my clothes from some of my favorite brands
- Q3. I like to buy premium brands
- Q4. Quality is the most important factor in my purchasing decision
- Q5. Style is the most important factor in my purchasing decision
- Q6. I prefer to buy clothes in store
- Q7. I prefer to buy clothes online
- Q8. Price is important
- Q9. I like to try different styles
- Q10. I like to make decision myself and don't need too much of others' suggestions

There are 4 segments of customers:

1. Price
2. Conspicuous
3. Quality
4. Style

Let's check it:

```
str(sim.dat,vec.len=3)
```

```
## 'data.frame':    1000 obs. of  19 variables:
## $ age         : int  57 63 59 60 51 59 57 57 ...
## $ gender      : Factor w/ 2 levels "Female","Male": 1 1 2 2 2 2 2 2 ...
## $ income      : num  120963 122008 114202 113616 ...
## $ house       : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 ...
## $ store_exp   : num  529 478 491 348 ...
## $ online_exp  : num  304 110 279 142 ...
## $ store_trans : int  2 4 7 10 4 4 5 11 ...
## $ online_trans: int  2 2 2 2 4 5 3 5 ...
## $ Q1          : int  4 4 5 5 4 4 4 5 ...
## $ Q2          : int  2 1 2 2 1 2 1 2 ...
## $ Q3          : int  1 1 1 1 1 1 1 1 ...
## $ Q4          : int  2 2 2 3 3 2 2 3 ...
## $ Q5          : int  1 1 1 1 1 1 1 1 ...
## $ Q6          : int  4 4 4 4 4 4 4 4 ...
## $ Q7          : int  1 1 1 1 1 1 1 1 ...
## $ Q8          : int  4 4 4 4 4 4 4 4 ...
## $ Q9          : int  2 1 1 2 2 1 1 2 ...
## $ Q10         : int  4 4 4 4 4 4 4 4 ...
## $ segment     : Factor w/ 4 levels "Conspicuous",..: 2 2 2 2 2 2 2 2 ...
```

Refer to Appendix for the simulation code.

3.2 Customer Satisfaction Survey Data from Airline Company

This data set is from a customer satisfaction survey for three airline companies. There are N=1000 respondents and 15 questions. The market researcher asked respondents to recall the experience with different airline companies and assign a score (1-9) to each airline company for all the 15 questions. The higher the score, the more satisfied the customer to the specific item. The 15 questions are of 4 types (the variable names are in the parentheses):

- How satisfied are you with our_____?

1. Ticketing

   - Ease of making reservation Easy_Reservation
   - Availability of preferred seats Preferred_Seats
   - Variety of flight options Flight_Options
   - Ticket prices Ticket_Prices

2. Aircraft

   - Seat comfort Seat_Comfort
   - Roominess of seat area Seat_Roominess
   - Availability of Overhead Overhead_Storage
   - Cleanliness of aircraft Clean_Aircraft

3. Service

   - Courtesy of flight attendant Courtesy
   - Friendliness Friendliness
   - Helpfulness Helpfulness
   - Food and drinks Service

4. General

   - Overall satisfaction Satisfaction
   - Purchase again Fly_Again
   - Willingness to recommend Recommend

Now check the data frame we have:

```
str(rating,vec.len=3)
```

```
## 'data.frame':    3000 obs. of  17 variables:
## $ Easy_Reservation: int  6 5 6 5 4 5 6 4 ...
## $ Preferred_Seats : int  5 7 6 6 5 6 6 6 ...
## $ Flight_Options  : int  4 7 5 5 3 4 6 3 ...
## $ Ticket_Prices   : int  5 6 6 5 6 5 5 5 ...
## $ Seat_Comfort    : int  5 6 7 7 6 6 6 4 ...
## $ Seat_Roominess  : int  7 8 6 8 7 8 6 5 ...
## $ Overhead_Storage: int  5 5 7 6 5 4 4 4 ...
```

```
##   $ Clean_Aircraft  : int   7 6 7 7 7 7 6 4 ...
##   $ Courtesy        : int   5 6 6 4 2 5 5 4 ...
##   $ Friendliness    : int   4 6 6 6 3 4 5 5 ...
##   $ Helpfulness     : int   6 5 6 4 4 5 5 4 ...
##   $ Service         : int   6 5 6 5 3 5 5 5 ...
##   $ Satisfaction    : int   6 7 7 5 4 6 5 5 ...
##   $ Fly_Again       : int   6 6 6 7 4 5 3 4 ...
##   $ Recommend       : int   3 6 5 5 4 5 6 5 ...
##   $ ID              : int   1 2 3 4 5 6 7 8 ...
## $ Airline        : Factor w/ 3 levels "AirlineCo.1",..: 1 1 1 1 1 1 1 1 ...
```

Refer to Appendix for the simulation code.

3.3 Swine Disease Breakout Data

The swine disease data includes simulated 120 survey questions from 800 farms. There are three choices for each question. The outbreak status for the ith farm is generated from a $Bernoulli(1, p_i)$ distribution with p_i being a function of the question answers:

$$ln(\frac{p_i}{1 - p_i}) = \beta_0 + \Sigma_{g=1}^G \mathbf{x_{i,g}^T}\ \mathbf{g}$$

where β_0 is the intercept, $\mathbf{x_{i,g}}$ is a three-dimensional indication vector for question answer and (β_g) is the parameter vector corresponding to the gth predictor. Three types of questions are considered regarding their effects on the outcome. The first forty survey questions are important questions such that the coefficients of the three answers to these questions are all different:

$$\mathbf{g} = (1, 0, -1) \times \gamma,\ g = 1, \dots, 40$$

The second forty survey questions are also important questions but only one answer has a coefficient that is different from the other two answers:

$$\mathbf{g} = (1, 0, 0) \times \gamma, \ g = 41, ..., 80$$

The last forty survey questions are also unimportant questions such that all three answers have the same coefficients:

$$\mathbf{g} = (0, 0, 0) \times \gamma, \ g = 81, ..., 120$$

The baseline coefficient β_0 is set to be $-\frac{40}{3}\gamma$ so that on average a farm have 50% of chance to have an outbreak. The parameter γ in the above simulation is set to control the strength of the questions' effect on the outcome. In this simulation study, we consider the situations where $\gamma = 0.1, 0.25, 0.5, 1, 2$. So the parameter settings are:

$$\mathbf{T} = \left(\frac{40}{3}, \underbrace{1, 0, -1}_{question \ 1}, ..., \underbrace{1, 0, -1}_{question \ 40}, \underbrace{1, 0, 0}_{question \ 41}, ..., \underbrace{1, 0, 0}_{question \ 80}, \underbrace{0, 0, 0}_{question \ 81}, ..., \underbrace{0, 0, 0}_{question \ 120} \right) * \gamma$$

For each value of γ, 20 data sets are simulated. The bigger γ is, the larger the corresponding parameter. We provided the data sets with $\gamma = 2$. Let's check the data:

```
disease_dat <- read.csv("http://bit.ly/2KXb1Qi")
# only show the last 7 columns here
head(subset(disease_dat,select=c("Q118.A","Q118.B","Q119.A",
                                 "Q119.B","Q120.A","Q120.B","y")))
```

```
##   Q118.A Q118.B Q119.A Q119.B Q120.A Q120.B y
## 1      1      0      0      0      0      1 1
## 2      0      1      0      1      0      0 1
## 3      1      0      0      0      1      0 1
## 4      1      0      0      0      0      1 1
## 5      1      0      0      0      1      0 0
## 6      1      0      0      1      1      0 1
```

Here `y` indicates the outbreak situation of the farms. `y=1` means there is an outbreak in 5 years after the survey. The rest columns indicate survey responses. For example `Q120.A = 1` means the respondent chose `A` in Q120. We consider `C` as the baseline.

Refer to Appendix for the simulation code.

3.4   MNIST Dataset

The MNIST dataset is a popular dataset for image classification machine learning model tutorials. It is conveniently included in the Keras library and ready to be loaded with build-in functions for analysis. The WIKI page of MNIST provides a detailed description of the dataset: `https://en.wikipedia.org/wiki/MNIST_database`. It contains 70,000 images of handwritten digits from American Census Bureau employees and American high school students. There are 60,000 training images and 10,000 testing images. Each image has a resolution of 28x28, and the numerical pixel values are in greyscale. Each image is represented by a 28x28 matrix with each element of the matrix an integer between 0 and 255. The label of each image is the intended digit of the handwritten image between 0 and 9. We cover the detailed steps to explore the MNIST dataset in the R and Python notebooks. A sample of the dataset is illustrated in the figure below:

3.5   IMDB Dataset

The IMDB dataset (`http://ai.stanford.edu/~amaas/data/sentiment/`) is a popular dataset for text and language-related machine learning tutorials. It is also conveniently included in the Keras library, and there are a few build-in functions in Keras for data loading and pre-processing. It contains 50,000 movie reviews (25,000 in training
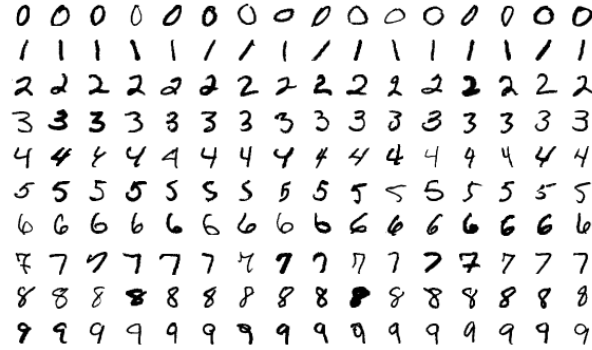
FIGURE 3.1: Sample of MNIST dataset (https://en.wikipedia.org/wiki/File:MnistExamples.png)

and 25,000 in testing) from IMDB, as well as each movie review's binary sentiment: positive or negative. The raw data contains the text of each movie review, and it has to be pre-processed before being fitted with any machine learning models. By using Keras's built-in functions, we can easily get the processed dataset (i.e., a numerical data frame) for machine learning algorithms. Keras' build-in functions perform the following tasks to convert the raw review text into a data frame:

1. Convert text data into numerical data. Machine learning models cannot work with raw text data directly, and we have to convert text into numbers. There are many different ways for the conversion and Keras' build-in function uses each word's rank of frequency in the entire training dataset to replace the raw text in both the training and testing dataset. For example, the 10th most frequent word is replaced by integer 10. There are a few additional setups for this process, including:

   a. Skip top frequent words. We usually skip a few top frequent words as they are mainly stopwords like "the" "and" or "a," which usually do not provide much information. There is a parameter in the build-in function to specify how many top words to

skip.

b. Set the maximum number of unique words. The entire vocabulary of the unique words in the training dataset may be large, and many of them have very low frequencies such as just appearing once in the entire training dataset. To keep the size of the vocabulary, we can also set up the maximum number of the unique words using Keras' built-in function such that any words with least frequencies will be replaced with a special index such as "2".

2. Padding or truncation to keep all the reviews to be the same length. For most machine learning models, the algorithms expect to see the same number of features (i.e., same number of input columns in the data frame). There is a parameter in the Keras build-in function to set the maximum number of words in each review (i.e., max_length). For reviews that have less than max_legth words, we pad them with "0". For reviews that have more than max_length words, we truncate them.

After the above pre-processing, each review is represented by one row in the data frame. There is one column for the binary positive/negative sentiment, and max_length columns input features converted from the raw review text. In the corresponding R and Python notebooks, we will go over the details of the data pre-processing using Keras' built-in functions.

4

Big Data Cloud Platform

Data has been statisticians and analysts' friend for hundreds of years. Tabulated data are the most common format that we use daily. People used to store data on papers, tapes, diskettes, or hard drives. Only recently, with the development of computer hardware and software, the volume, variety, and speed of the data exceed the capacity of a traditional statistician or analyst. So using data becomes a science that focuses on the question: how can we store, access, process, analyze the massive amount of data and provide actionable insights? In the past few years, by utilizing commodity hardware and open-source software, people created a big data ecosystem for data storage, data retrieval, and parallel computation. Hadoop and Spark have become a popular platform that enables data scientists, statisticians, and analysts to access the data and to build models. Programming skills in the big data platform have been an obstacle for a traditional statistician or analyst to become a successful data scientist. However, cloud computing reduces the difficulty significantly. The user interface of the data platform is much more friendly today, and people push mush of the technical details to the background. Today's cloud systems also enable quick implementation of the production environment. Now data science emphasizes more on the data itself, models and algorithms on top of the data, rather than the platform, infrastructure and low-level programming such as Java.

4.1   Power of Cluster of Computers

We are familiar with our laptop/desktop computers which have three main components to do data computation: (1) Hard disk, (2) Memory, and (3) CPU.

The data and codes stored in the hard disk have specific features such as slow to read and write, and large capacity of around a few TB in today's market. Memory is fast to read and write but with small capacity in the order of a few dozens of GB in today's market. CPU is where all the computation happens.
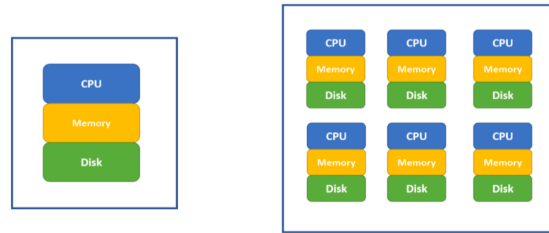
FIGURE 4.1: Single computer (left) and a cluster of computers (right)

For statistical software such as R, the amount of data it can process is limited by the computer's memory. The memory of computers before 2000 is less than 1 GB. The memory capacity grows way slower than the amount of the data. Now it is common that we need to analyze data far beyond the capacity of a single computer's memory, especially in an enterprise environment. Meanwhile, as the data size increases, to solve the same problem (such as regressions), the computation time is growing faster than linear. Using a cluster of computers become a common way to solve a big data problem. In Figure 4.1 (right), a cluster of computers can be viewed as one powerful machine with memory, hard disk and CPU equivalent to the sum of individual computers. It is common to have hundreds or even thousands of nodes for a cluster.

In the past, users need to write code (such as MPI) to distribute data and do parallel computing. Fortunately, with the recent new development, the cloud environment for big data analysis is more user-friendly. As data is often beyond the size of the hard disk, the dataset itself is stored across different nodes (i.e., the Hadoop system). When doing analysis, the data is distributed across different nodes, and algorithms are parallel to leverage corresponding nodes' CPUs to compute (i.e., the Spark system).

4.2 Evolution of Cluster Computing

Using computer clusters to solve general-purpose data and analytics problems needs a lot of effort if we have to specifically control every element and steps such as data storage, memory allocation, and parallel computation. Fortunately, high tech companies and open source communities have developed the entire ecosystem based on Hadoop and Spark. Users need only to know high-level scripting languages such as Python and R to leverage computer clusters' distributed storage, memory and parallel computation power.

4.2.1 Hadoop

The very first problem internet companies face is that a lot of data has been collected and how to better store these data for future analysis. Google developed its own file system to provide efficient, reliable access to data using large clusters of commodity hardware. The open-source version is known as Hadoop Distributed File System (HDFS). Both systems use Map-Reduce to allocate computation across computation nodes on top of the file system. Hadoop is written in Java and writing map-reduce job using Java is a direct way to interact with Hadoop which is not familiar to many in the data and analytics community. To help better use the Hadoop system, an SQL-like data warehouse system called

Hive, and a scripting language for analytics interface called Pig were introduced for people with analytics background to interact with Hadoop system. Within Hive, we can create user-defined functions through R or Python to leverage the distributed and parallel computing infrastructure. Map-reduce on top of HDFS is the main concept of the Hadoop ecosystem. Each map-reduce operation requires retrieving data from hard disk, then performing the computation, and storing the result onto the disk again. So, jobs on top of Hadoop require a lot of disk operation which may slow down the entire computation process.

4.2.2 Spark

Spark works on top of a distributed file system including HDFS with better data and analytics efficiency by leveraging in-memory operations. Spark is more tailored for data processing and analytics and the need to interact with Hadoop directly is greatly reduced. The spark system includes an SQL-like framework called Spark SQL and a parallel machine learning library called MLib. Fortunately for many in the analytics community, Spark also supports R and Python. We can interact with data stored in a distributed file system using parallel computing across nodes easily with R and Python through the Spark API and do not need to worry about lower-level details of distributed computing. We will introduce how to use an R notebook to drive Spark computations.

4.3 Introduction of Cloud Environment

Even though Spark provides a solution for big data analytics, the maintenance of the computing cluster and Spark system requires a dedicated team. Historically for each organization the IT departments own the hardware and the regular maintenance. It usually takes months for a new environment to be built and the cost is high. Luckily, the time to deployment and cost are dramatically

down due to the cloud computation trend. Now we can create a Spark computing cluster in the cloud in a few minutes with the desired configuration and the user only pay when the cluster is up. Cloud computing environments enable smaller organizations to adopt big data analytics.

There are many cloud computing environments such as Amazon's AWS and Microsoft Azure which provide a complete list of functions for heavy-duty enterprise applications. For example, Netflix runs its business entirely on AWS without owning any data centers. For beginners, however, Databricks provides an easy to use cloud system for learning purposes. Databricks is a company founded by the creators of Apache Spark and it provides a user-friendly web-based notebook environment that can create a Spark cluster on the fly to run R/Python/Scala/SQL scripts. We will use Databricks' free community edition to run demos in this book. Please note, to help readers to get familiar with the Databricks cloud system, the content of this section is partially adopted from the following web pages:

- `https://docs.databricks.com/user-guide/faq/sparklyr.html`
- `http://spark.rstudio.com/index.html`

4.3.1 Open Account and Create a Cluster

Anyone can apply for a free Databrick account through `https://databricks.com/try-databricks` and please make sure to choose the "**COMMUNITY EDITION**" which does not require credit card information and will always be free. Once the community edition account is open and activated. Users can create a cluster computation environment with Spark. The computing cluster associated with community edition account is relatively small, but it is good enough for running all the examples in this book. The main user interface to the computing environment is notebook: a collection of cells that contains formatted text or codes. When a new notebook is created, user will need to choose the default programming language type (i.e. Python, R, Scala, or SQL) and

every cells in the notebook will assume the default programming language. However, user can easily override the default selection of programming language by adding `%sql`, `%python`, `%r` or `%scala` at the first line of each cell to indicate the programming language in that cell. Allowing running different cells with different programming language in the same notebook enable user to have the flexibility to choose the best tools for each task. User can also define a cell to be markdown cell by adding `%md` at the first line of the cell. A markdown cell does not performance computation and it is just a cell to show formatted text. Well separated cells with computation, graph and formatted text enable user to create easy to maintain reproducible reports. The link to a video showing how to open Databricks account, how to create a cluster, and how to create notebooks is included in the book's website.

4.3.2   R Notebook

For this book, we will use R notebook for examples and demos and the corresponding Python notebook will be available online too. For an R notebook, it contains multiple cells, and, by default, the content within each cell are R scripts. Usually, each cell is a well-managed segment of a few lines of codes that accomplish a specific task. For example, Figure 4.2 shows the default cell for an R notebook. We can type in R scripts and comments same as we are using R console. By default, only the result from the last line will be shown following the cell. However, you can use `print()` function to output results for any lines. If we move the mouse to the middle of the lower edge of the cell below the results, a "+" symbol will show up and click on the symbol will insert a new cell below. When we click any area within a cell, it will make it editable and you will see a few icons on the top right corn of the cell where we can run the cell, as well as add a cell below or above, copy the cell, cut the cell etc. One quick way to run the cell is `Shift+Enter` when the cell is chosen. User will become familiar with the notebook environment quickly.
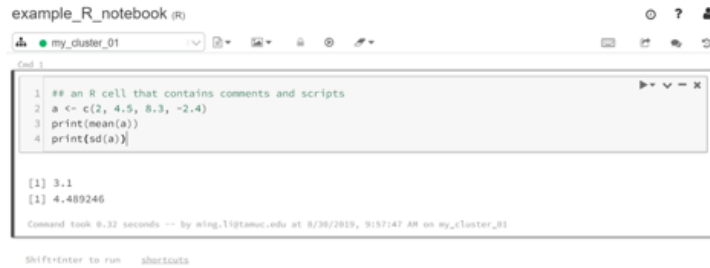
FIGURE 4.2: Example of R Notebook

4.3.3   Markdown Cells

For an R notebook, every cell by default will contain R scripts.
But if we put %md, %sql or %python at the first line of a cell, that
cell becomes Markdown cell, SQL script cell, and Python script
cell accordingly. For example, Figure 4.3 shows a markdown cell
with scripts and the actual appearance when exits editing mode.
Markdown cell provides a straightforward way to descript what
each cell is doing as well as what the entire notebook is about. It
is a better way than a simple comment within the code.

FIGURE 4.3: Example of Markdown Cell

4.4 Leverage Spark Using R Notebook

R is a powerful tool for data analysis given the data can be fit
into memory. Because of the memory bounded dataset limit, R it-
self cannot be used directly for big data analysis where the data is
likely stored in Hadoop and Spark system. By leverage the `sparklyr`
package created by RStudio, we can use Databricks' R notebook
to analyze data stored in the Spark system. As the data are stored
across different nodes, Spark enables parallel computation using
the collection of memory and CPU across all nodes. The fundamen-
tal data element in the Spark system is called Spark DataFrames
(SDF). In this section, we will illustrate how to use Databricks' R
notebook for big data analysis on top of the Spark environment
through `sparklyr` package.

Install pacakge

First, we need to install `sparklyr` package which enables the connec-
tion between master or local node to Spark cluster environments.
As it will install more than 10 dependencies, it may take more
than 10 minutes to finish. Be patient while it is installing! Once
the installation finishes, load the `sparklyr` package as illustrated
by the following code:

```r
# Install sparklyr
if (!require("sparklyr")) {
install.packages("sparklyr")
}
# Load sparklyr package
library(sparklyr)
```

Create a Spark Connection

Once the library is loaded, we need to create a Spark Connection to
link the computing node (i.e. local node) running the R notebook
to the Spark environment. Here we use the `"databricks"` option for

parameter `method` which is specific for databricks' cloud system. In other enterprise environments, please consult your administrator for details. The Spark Connection (i.e. `sc`) is the pipe to connect R notebook in the local node with the Spark Cluster. We can think of the R notebook is running on a local node that has its memory and CPU; the Spark system has a cluster of connected computation nodes, and the Spark Connection creates a mechanism to connect both systems. The Spark Connection can be established with:

```r
# create a sparklyr connection
sc <- spark_connect(method = "databricks")
```

To simplify the learning process, let us use a very familiar small dataset: the iris dataset. It is part of the `dplyr` library and we can load that library to use the `iris` data frame. Now the `iris` dataset is still on the local node where the R notebook is running on. And we can check the first a few lines of the `iris` dataset using the code below:

```r
library(dplyr)
head(iris)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1           5.1         3.5          1.4         0.2
## 2           4.9         3.0          1.4         0.2
## 3           4.7         3.2          1.3         0.2
## 4           4.6         3.1          1.5         0.2
## 5           5.0         3.6          1.4         0.2
## 6           5.4         3.9          1.7         0.4
##    Species
## 1  setosa
## 2  setosa
## 3  setosa
## 4  setosa
```

```
## 5  setosa
## 6  setosa
```

IMPORTANT - Copy Data to Spark Environment

In real applications, the data set may be massive and cannot fit in a single hard disk and most likely such data are already stored in the Spark system. If the data is already in Hadoop/Spark ecosystem in the form of SDF, we can create a local R object to link to the SDF by the `tbl()` function where `my_sdf` is the SDF in the Spark system, and `my_sdf_tbl` is the R local object that referring to `my_sdf`:

```
my_sdf_tbl <- tbl(sc = sc, my_sdf)
```

As we just created a brand new Spark computing environment, there is no SDF in the system yet. We will need to copy a local dataset to the Spark environment. As we have already created the Spark Connection `sc`, it is easy to copy data to spark system using `sdf_copy_to()` function as below:

```
iris_tbl <- sdf_copy_to(sc = sc, x = iris, overwrite = T)
```

The above one-line code copies iris dataset from the local node to Spark cluster environment. "`sc`" is the Spark Connection we just created; "**x**" is the data frame that we want to copy; "`overwrite`" is the option whether we want to overwrite the target object if the same name SDF exists in the Spark environment. Finally, `sdf_copy_to()` function will return an R object representing the copied SDF (i.e. creating a "pointer" to the SDF such that we can refer `iris_tbl` in the R notebook to operate `iris` SDF). Now `irir_tbl` in the local R environment can be used to refer to the `iris` SDF in the Spark system.

To check whether the `iris` data was copied to the Spark environment successfully or not, we can use `src_tbls()` function to the Spark Connection (`sc`):

```
## code to return all the data frames associated with sc
src_tbls(sc)
```

Analyzing the Data

Now we have successfully copied the `iris` dataset to the Spark environment as a SDF. This means that `iris_tbl` is an R object representing the `iris` SDF and we can use `iris_tbl` in R to refer the `iris` dataset in the Spark system (i.e. the `iris` SDF). With the `sparklyr` packages, we can nearly all the functions in `dplyr` to Spark DataFrame directly through `iris_tbl`, same as we are applying `dplyr` functions to a local R data frame in our laptop. For example, we can use the `%>%` operator to pass `iris_tbl` to the `count()` function:

```
iris_tbl %>% count
```

or using the `head()` function to return the first few rows in `iris_tbl`:

```
head(iris_tbl)
```

or applying more advanced data manipulation directly to `iris_tbl`:

```
iris_tbl %>%
  mutate(Sepal_Width = round(Sepal_Width * 2) / 2) %>%
  group_by(Species, Sepal_Width) %>%
  summarize(count = n(), Sepal_Length = mean(Sepal_Length),
            stdev = sd(Sepal_Length))
```

Collect Results Back to Master Node

Even though we can run nearly all of the `dplyr` functions on SDF,

we cannot apply functions from other packages directly to SDF (such as `ggplot()`). For functions that can only work on local R data frames, we must copy the SDF back to the local node as an R data frame. To copy SDF back to the local node, we use the `collect()` function. The following code using `collect()` will collect the results of a few operations and assign the collected data to `iris_summary`, a local R data frame:

```
iris_summary <- iris_tbl %>%
  mutate(Sepal_Width = round(Sepal_Width * 2) / 2) %>%
  group_by(Species, Sepal_Width) %>%
  summarize(count = n(),
            Sepal_Length = mean(Sepal_Length),
            stdev = sd(Sepal_Length)) %>%
  collect()
```

Now, `iris_summary` is a local R object to the R notebook and we can use any R packages and functions to it. In the following code, we will apply `ggplot()` to it, exactly the same as a stand along R console:

```
library(ggplot2)
ggplot(iris_summary, aes(Sepal_Width, Sepal_Length, color = Species)) +
  geom_line(size = 1.2) +
  geom_errorbar(aes(ymin = Sepal_Length - stdev,
                    ymax = Sepal_Length + stdev), width = 0.05) +
  geom_text(aes(label = count), vjust = -0.2, hjust = 1.2, color = "black") +
  theme(legend.position="top")
```

In most cases, the heavy-duty data preprocessing and aggregation is done in Spark using functions in ·dplyr·. Once the data is aggregated, the size is usually dramatically reduced and such reduced data can be collected to an R local object for downstream analysis.

Fit Regression to SDF

One of the advantages of the Spark system is the parallel machine learning algorithm. There are many statistical and machine learning algorithms developed to run in parallel across many CPUs with data across many memory units for SDF. In this example, we have already uploaded the `iris` data to the Spark system, and the data in the SDF can be referred through `iris_tbl` as in the last section. The linear regression algorithm implemented in the Spark system can be called through `ml_linear_regression()` function. The syntax to call the function is to define the local R object that representing the SDF (i.e. `iris_tbl` (local R object) for `iris` (SDF)), response variable (i.e. the y variable in linear regression in the SDF) and features (i.e. the x variables in linear regression in the SDF). Now, we can easily fit a linear regression for large dataset far beyond the memory limit of one single computer, and it is truly scalable and only constrained by the resource of the Spark cluster. Below is an illustration of how to fit a linear regression to SDF using R notebook:

```
fit1 <- ml_linear_regression(x = iris_tbl, response = "Sepal_Length",
        features = c("Sepal_Width", "Petal_Length", "Petal_Width"))
summary(fit1)
```

In the above code, `x` is the R object pointing to the SDF; `response` is y-variable, `features` are the collection of explanatory variables. For this function, both the data and computation are in the Spark cluster which leverages multiple CPUs, distributed memories and parallel computing.

Fit a K-means Cluster

Through the `sparklyr` package, we can use an R notebook to access many Spark Machine Learning Library (MLlib) algorithms such as Linear Regression, Logistic Regression, Survival Regression, Generalized Linear Regression, Decision Trees, Random Forests, Gradient-Boosted Trees, Principal Components Analysis, Naive-

Bayes, K-Means Clustering, and a few other methods. Below codes fit a k-means cluster algorithm:

```
## Now fit a k-means clustering using iris_tbl data
## with only two out of four features in iris_tbl
fit2 <- ml_kmeans(x = iris_tbl, centers = 3,
                  features = c("Petal_Length", "Petal_Width"))
# print our model fit
print(fit2)
```

$<<<<<<<$ HEAD After fitting the k-means model, we can apply the model to predict other datasets through `sdf_predict()` function. Following code applies the model to `iris_tbl` again to predict the cluster and collect the results as a local R object (i.e. `prediction`) using `collect()` function:

```
prediction = collect(sdf_predict(fit2, iris_tbl))
```

As `prediction` is a local R object, we can apply any R functions from any libraries to it. For example:

```
prediction %>%
  ggplot(aes(Petal_Length, Petal_Width)) +
  geom_point(aes(Petal_Width, Petal_Length,
                 col = factor(prediction + 1)),
             size = 2, alpha = 0.5) +
  geom_point(data = fit2$centers, aes(Petal_Width, Petal_Length),
             col = scales::muted(c("red", "green", "blue")),
             pch = 'x', size = 12) +
  scale_color_discrete(name = "Predicted Cluster",
                       labels = paste("Cluster", 1:3)) +
  labs(x = "Petal Length",
       y = "Petal Width",
```

```
title = "K-Means Clustering",
subtitle = "Use Spark ML to predict cluster
membership with the iris dataset")
```

So far, we have illustrated

1. the relationship between a local node (i.e. where R note-book is running) and Spark Clusters (i..e where data are stored and computation are done);
2. how to copy a local data frame to a Spark DataFrames (please note if your data is already in Spark environment, there is no need to copy and we only need to build the connection. This is likely to be the case for enterprise environment);
3. how to manipulate Spark DataFrames for data cleaning and preprocessing through `dplyr` functions with the instal-lation of `sparklyr` package;
4. how to fit statistical and machine learning models to Spark DataFrame in a truly parallel manner;
5. how to collect information from Spark DataFrames back to a local R object (i.e. local R data frame) for future analysis.

These procedures cover the basics of big data analysis that a data scientist needs to know as a beginner. We have an R notebook on the book website that contains the contents of this chapter. We also have a Python notebook on the book website.

5

Data Pre-processing

Many data analysis related books focus on models, algorithms and statistical inferences. However, in practice, raw data is usually not directly used for modeling. Data preprocessing is the process of converting raw data into clean data that is proper for modeling. A model fails for various reasons. One is that the modeler doesn't correctly preprocess data before modeling. Data preprocessing can significantly impact model results, such as imputing missing value and handling with outliers. So data preprocessing is a very critical part.
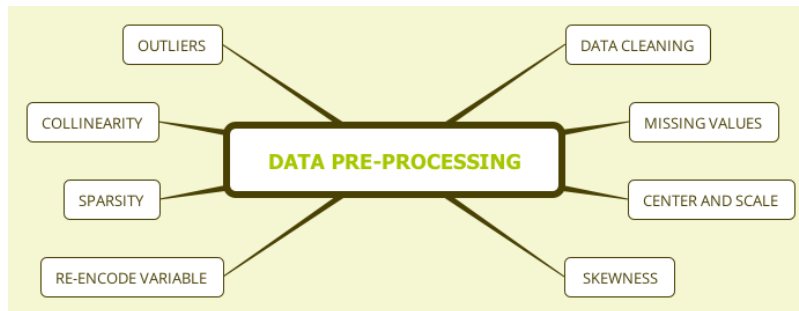

FIGURE 5.1: Data Pre-processing Outline

In real life, depending on the stage of data cleanup, data has the following types:

1. Raw data
2. Technically correct data
3. Data that is proper for the model
4. Summarized data
5. Data with fixed format

The raw data is the first-hand data that analysts pull from the database, market survey responds from your clients, the experimental results collected by the R & D department, and so on. These data may be very rough, and R sometimes can't read them directly. The table title could be multi-line, or the format does not meet the requirements:

- Use 50% to represent the percentage rather than 0.5, so R will read it as a character;
- The missing value of the sales is represented by "-" instead of space so that R will treat the variable as character or factor type;
- The data is in a slideshow document, or the spreadsheet is not ".csv" but ".xlsx"
- ...

Most of the time, you need to clean the data so that R can import them. Some data format requires a specific package. Technically correct data is the data, after preliminary cleaning or format conversion, that R (or another tool you use) can successfully import it.

Assume we have loaded the data into R with reasonable column names, variable format and so on. That does not mean the data is entirely correct. There may be some observations that do not make sense, such as age is negative, the discount percentage is greater than 1, or data is missing. Depending on the situation, there may be a variety of problems with the data. It is necessary to clean the data before modeling. Moreover, different models have different requirements on the data. For example, some model may require the variables are of consistent scale; some may be susceptible to outliers or collinearity, some may not be able to handle categorical variables and so on. The modeler has to preprocess the data to make it proper for the specific model.

Sometimes we need to aggregate the data. For example, add up the daily sales to get annual sales of a product at different locations. In customer segmentation, it is common practice to build a profile for each segment. It requires calculating some statistics

such as average age, average income, age standard deviation, etc. Data aggregation is also necessary for presentation, or for data visualization.

The final table results for clients need to be in a nicer format than what used in the analysis. Usually, data analysts will take the results from data scientists and adjust the format, such as labels, cell color, highlight. It is important for a data scientist to make sure the results look consistent which makes the next step easier for data analysts.

It is highly recommended to store each step of the data and the R code, making the whole process as repeatable as possible. The R markdown reproducible report will be extremely helpful for that. If the data changes, it is easy to rerun the process. In the remainder of this chapter, we will show the most common data preprocessing methods.

Load the R packages first:

```r
# install packages from CRAN
p_needed <- c('imputeMissings','caret','e1071','psych','car','corrplot')
packages <- rownames(installed.packages())
p_to_install <- p_needed[!(p_needed %in% packages)]
if (length(p_to_install) > 0) {
    install.packages(p_to_install)
}


lapply(p_needed, require, character.only = TRUE)
```

5.1 Data Cleaning

After you load the data, the first thing is to check how many variables are there, the type of variables, the distributions, and data errors. Let's read and check the data:

```
sim.dat <- read.csv("http://bit.ly/2P5gTw4")
summary(sim.dat)
```

```
      age            gender         income          house         store_exp
 Min.   : 16.0   Female:554   Min.   : 41776   No :432     Min.   : -500
 1st Qu.: 25.0   Male  :446   1st Qu.: 85832   Yes:568     1st Qu.:  205
 Median : 36.0                Median : 93869               Median :  329
 Mean   : 38.8                Mean   :113543               Mean   : 1357
 3rd Qu.: 53.0                3rd Qu.:124572               3rd Qu.:  597
 Max.   :300.0                Max.   :319704               Max.   :50000
                              NA's   :184
   online_exp     store_trans     online_trans       Q1             Q2
 Min.   :  69   Min.   : 1.00   Min.   : 1.0    Min.   :1.0    Min.   :1.00
 1st Qu.: 420   1st Qu.: 3.00   1st Qu.: 6.0    1st Qu.:2.0    1st Qu.:1.00
 Median :1942   Median : 4.00   Median :14.0    Median :3.0    Median :1.00
 Mean   :2120   Mean   : 5.35   Mean   :13.6    Mean   :3.1    Mean   :1.82
 3rd Qu.:2441   3rd Qu.: 7.00   3rd Qu.:20.0    3rd Qu.:4.0    3rd Qu.:2.00
 Max.   :9479   Max.   :20.00   Max.   :36.0    Max.   :5.0    Max.   :5.00


       Q3             Q4             Q5             Q6             Q7
 Min.   :1.00   Min.   :1.00   Min.   :1.00   Min.   :1.00   Min.   :1.00
 1st Qu.:1.00   1st Qu.:2.00   1st Qu.:1.75   1st Qu.:1.00   1st Qu.:2.50
 Median :1.00   Median :3.00   Median :4.00   Median :2.00   Median :4.00
 Mean   :1.99   Mean   :2.76   Mean   :2.94   Mean   :2.45   Mean   :3.43
 3rd Qu.:3.00   3rd Qu.:4.00   3rd Qu.:4.00   3rd Qu.:4.00   3rd Qu.:4.00
 Max.   :5.00   Max.   :5.00   Max.   :5.00   Max.   :5.00   Max.   :5.00


       Q8             Q9             Q10              segment
 Min.   :1.0    Min.   :1.00   Min.   :1.00    Conspicuous:200
 1st Qu.:1.0    1st Qu.:2.00   1st Qu.:1.00    Price      :250
 Median :2.0    Median :4.00   Median :2.00    Quality    :200
 Mean   :2.4    Mean   :3.08   Mean   :2.32    Style      :350
 3rd Qu.:3.0    3rd Qu.:4.00   3rd Qu.:3.00
 Max.   :5.0    Max.   :5.00   Max.   :5.00
```

Are there any problems? Questionnaire response Q1-Q10 seem

reasonable, the minimum is 1 and maximum is 5. Recall that the questionnaire score is 1-5. The number of store transactions (`store_trans`) and online transactions (`online_trans`) make sense too. Things to pay attention are:

- There are some missing values.
- There are outliers for store expenses (`store_exp`). The maximum value is 50000. Who would spend $50000 a year buying clothes? Is it an imputation error?
- There is a negative value (-500) in `store_exp` which is not logical.
- Someone is 300 years old.

How to deal with that? Depending on the real situation, if the sample size is large enough, it does not hurt to delete those problematic samples. Here we have 1000 observations. Since marketing survey is usually expensive, it is better to set these values as missing and impute them instead of deleting the rows.

```
# set problematic values as missings
sim.dat$age[which(sim.dat$age > 100)] <- NA
sim.dat$store_exp[which(sim.dat$store_exp < 0)] <- NA
# see the results
summary(subset(sim.dat, select = c("age", "store_exp")))
```

```
      age            store_exp
 Min.   :16.00   Min.   :  155.8
 1st Qu.:25.00   1st Qu.:  205.1
 Median :36.00   Median :  329.8
 Mean   :38.58   Mean   : 1358.7
 3rd Qu.:53.00   3rd Qu.:  597.4
 Max.   :69.00   Max.   :50000.0
 NA's   :1       NA's   :1
```

Now let's deal with the missing values in the data.

5.2 Missing Values

You can write a whole book about missing value. This section will only show some of the most commonly used methods without getting too deep into the topic. Chapter 7 of the book by De Waal, Pannekoek and Scholtus (de Waal et al., 2011) makes a concise overview of some of the existing imputation methods. The choice of specific method depends on the actual situation. There is no best way.

One question to ask before imputation: Is there any auxiliary information? Being aware of any auxiliary information is critical. For example, if the system set customer who did not purchase as missing, then the real purchasing amount should be 0. Is missing a random occurrence? If so, it may be reasonable to impute with mean or median. If not, is there a potential mechanism for the missing data? For example, older people are more reluctant to disclose their ages in the questionnaire, so that the absence of age is not completely random. In this case, the missing values need to be estimated using the relationship between age and other independent variables. For example, use variables such as whether they have children, income, and other survey questions to build a model to predict age.

Also, the purpose of modeling is important for selecting imputation methods. If the goal is to interpret the parameter estimate or statistical inference, then it is important to study the missing mechanism carefully and to estimate the missing values using non-missing information as much as possible. If the goal is to predict, people usually will not study the absence mechanism rigorously (but sometimes the mechanism is obvious). If the absence mechanism is not clear, treat it as missing at random and use mean, median, or k-nearest neighbor to impute. Since statistical inference is sensitive to missing values, researchers from survey statistics have conducted in-depth studies of various imputation schemes which focus on valid statistical inference. The problem of missing values in the prediction model is different from that in the traditional

survey. Therefore, there are not many papers on missing value imputation in the prediction model. Those who want to study further can refer to Saar-Tsechansky and Provost's comparison of different imputation methods (M and F, 007b) and De Waal, Pannekoek and Scholtus' book (de Waal et al., 2011).

5.2.1   Impute missing values with median/mode

In the case of missing at random, a common method is to impute with the mean (continuous variable) or median (categorical variables). You can use `impute()` function in `imputeMissings` package.

```
# save the result as another object
demo_imp <- impute(sim.dat, method = "median/mode")
# check the first 5 columns, there is no missing values in other columns
summary(demo_imp[, 1:5])
```

```
      age            gender          income         house       store_exp
 Min.   :16.00   Female:554   Min.   : 41776   No :432   Min.   :  155.8
 1st Qu.:25.00   Male  :446   1st Qu.: 87896   Yes:568   1st Qu.:  205.1
 Median :36.00                Median : 93869             Median :  329.8
 Mean   :38.58                Mean   :109923             Mean   : 1357.7
 3rd Qu.:53.00                3rd Qu.:119456             3rd Qu.:  597.3
 Max.   :69.00                Max.   :319704             Max.   :50000.0
```

After imputation, `demo_imp` has no missing value. This method is straightforward and widely used. The disadvantage is that it does not take into account the relationship between the variables. When there is a significant proportion of missing, it will distort the data. In this case, it is better to consider the relationship between variables and study the missing mechanism. In the example here, the missing variables are numeric. If the missing variable is a categorical/factor variable, the `impute()` function will impute with the mode.

You can also use `preProcess()` in package `caret`, but it is only for

numeric variables, and can not impute categorical variables. Since missing values here are numeric, we can use the `preProcess()` function. The result is the same as the `impute()` function. `PreProcess()` is a powerful function that can link to a variety of data preprocessing methods. We will use the function later for other data preprocessing.

```
imp <- preProcess(sim.dat, method = "medianImpute")
demo_imp2 <- predict(imp, sim.dat)
summary(demo_imp2[, 1:5])
```

```
      age              gender         income        house         store_exp
 Min.   :16.00    Female:554    Min.   : 41776    No :432    Min.   :  155.8
 1st Qu.:25.00    Male  :446    1st Qu.: 87896    Yes:568    1st Qu.:  205.1
 Median :36.00                  Median : 93869               Median :  329.8
 Mean   :38.58                  Mean   :109923               Mean   : 1357.7
 3rd Qu.:53.00                  3rd Qu.:119456               3rd Qu.:  597.3
 Max.   :69.00                  Max.   :319704               Max.   :50000.0
```

5.2.2 K-nearest neighbors

K-nearest neighbor (KNN) will find the k closest samples (Euclidian distance) in the training set and impute the mean of those "neighbors."

Use `preProcess()` to conduct KNN:

```
imp <- preProcess(sim.dat, method = "knnImpute", k = 5)
# need to use predict() to get KNN result
demo_imp <- predict(imp, sim.dat)
# only show the first three elements
lapply(sim.dat, class)[1:3]
```

```
      age              gender          income
```

```
Min.    :-1.5910972    Female:554    Min.    :-1.43989
1st Qu.:-0.9568733    Male  :446    1st Qu.:-0.53732
Median :-0.1817107                  Median :-0.37606
Mean    : 0.0000156                 Mean    : 0.02389
3rd Qu.: 1.0162678                  3rd Qu.: 0.21540
Max.    : 2.1437770                 Max.    : 4.13627
```

The `preProcess()` in the first line will automatically ignore non-numeric columns.

Comparing the KNN result with the previous median imputation, the two are very different. This is because when you tell the `preProcess()` function to use KNN (the option `method =" knnImpute"`), it will automatically standardize the data. Another way is to use Bagging tree (in the next section). Note that KNN can not impute samples with the entire row missing. The reason is straightforward. Since the algorithm uses the average of its neighbors if none of them has a value, what does it apply to calculate the mean?

Let's append a new row with all values missing to the original data frame to get a new object called `temp`. Then apply KNN to `temp` and see what happens:

```
temp <- rbind(sim.dat, rep(NA, ncol(sim.dat)))
imp <- preProcess(sim.dat, method = "knnImpute", k = 5)
demo_imp <- predict(imp, temp)
```

```
Error in FUN(newX[, i], ...) :
  cannot impute when all predictors are missing in the new data point
```

There is an error saying "`cannot impute when all predictors are missing in the new data point`". It is easy to fix by finding and removing the problematic row(s):

```
idx <- apply(temp, 1, function(x) sum(is.na(x)))
as.vector(which(idx == ncol(temp)))
```

It shows that row 1001 is problematic. You can go ahead to delete it.

5.2.3   Bagging Tree

Bagging (Bootstrap aggregating) was originally proposed by Leo Breiman. It is one of the earliest ensemble methods (L, 966a). When used in missing value imputation, it will use the remaining variables as predictors to train a bagging tree and then use the tree to predict the missing values. Although theoretically, the method is powerful, the computation is much more intense than KNN. In practice, there is a trade-off between computation time and the effect. If a median or mean meet the modeling needs, even bagging tree may improve the accuracy a little, but the upgrade is so marginal that it does not deserve the extra time. The bagging tree itself is a model for regression and classification. Here we use `preProcess()` to impute `sim.dat`:

```
imp <- preProcess(sim.dat, method = "bagImpute")
demo_imp <- predict(imp, sim.dat)
summary(demo_imp[, 1:5])
```

```
       age            gender           income         house        store_exp
 Min.   :16.00   Female:554   Min.   : 41776   No :432   Min.   :  155.8
 1st Qu.:25.00   Male  :446   1st Qu.: 86762   Yes:568   1st Qu.:  205.1
 Median :36.00                Median : 94739             Median :  329.0
 Mean   :38.58                Mean   :114665             Mean   : 1357.7
 3rd Qu.:53.00                3rd Qu.:123726             3rd Qu.:  597.3
 Max.   :69.00                Max.   :319704             Max.   :50000.0
```

5.3 Centering and Scaling

It is the most straightforward data transformation. It centers and scales a variable to mean 0 and standard deviation 1. It ensures that the criterion for finding linear combinations of the predictors is based on how much variation they explain and therefore improves the numerical stability. Models involving finding linear combinations of the predictors to explain response/predictors variation need data centering and scaling, such as PCA (Jolliffe, 2002), PLS (Geladi P, 1986) and EFA (Mulaik, 2009). You can quickly write code yourself to conduct this transformation.

Let's standardize the variable `income` from `sim.dat`:

```r
income <- sim.dat$income
# calculate the mean of income
mux <- mean(income, na.rm = T)
# calculate the standard deviation of income
sdx <- sd(income, na.rm = T)
# centering
tr1 <- income - mux
# scaling
tr2 <- tr1/sdx
```

Or the function `preProcess()` can apply this transformation to a set of predictors.

```r
sdat <- subset(sim.dat, select = c("age", "income"))
# set the 'method' option
trans <- preProcess(sdat, method = c("center", "scale"))
# use predict() function to get the final result
transformed <- predict(trans, sdat)
```

Now the two variables are in the same scale. You can check the result using `summary(transformed)`. Note that there are missing values.

5.4 Resolve Skewness

Skewness[1] is defined to be the third standardized central moment. The formula for the sample skewness statistics is:

$$skewness = \frac{\sum(x_i - \bar{x})^3}{(n-1)v^{3/2}}$$

$$v = \frac{\sum(x_i - \bar{x})^2}{(n-1)}$$

Skewness=0 means that the destribution is symmetric, i.e. the probability of falling on either side of the distribution's mean is equal.

```r
# need skewness() function from e1071 package
set.seed(1000)
par(mfrow = c(1, 2), oma = c(2, 2, 2, 2))
# random sample 1000 chi-square distribution with df=2
# right skew
x1 <- rchisq(1000, 2, ncp = 0)
# get left skew variable x2 from x1
x2 <- max(x1) - x1
plot(density(x2), main = paste("left skew, skewnwss =",
    round(skewness(x2), 2)), xlab = "X2")
plot(density(x1), main = paste("right skew, skewness =",
    round(skewness(x1), 2)), xlab = "X1")
```

---

[1] https://en.wikipedia.org/wiki/Skewness

left skew, skewnwss = -1.8   **right skew, skewness = 1.8**
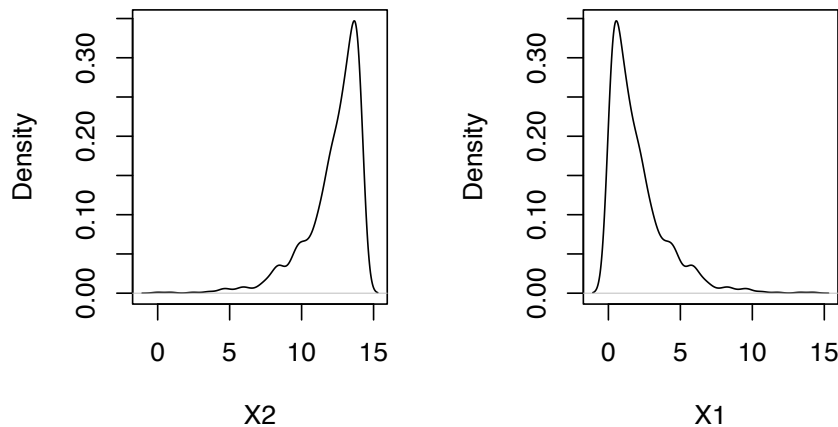
There are different ways may help to remove skewness such as log, square root or inverse. However, it is often difficult to determine from plots which transformation is most appropriate for correcting skewness. The Box-Cox procedure automatically identified a transformation from the family of power transformations that are indexed by a parameter λ(Box G, 1964).

$$x^* = \begin{cases} \frac{x^\lambda - 1}{\lambda} & if \ \lambda \neq 0 \\ log(x) & if \ \lambda = 0 \end{cases}$$

It is easy to see that this family includes log transformation ($\lambda = 0$), square transformation ($\lambda = 2$), square root ($\lambda = 0.5$), inverse ($\lambda = -1$) and others in-between. We can still use function `preProcess()` in package `caret` to apply this transformation by chaning the `method` argument.

```
describe(sim.dat)
```

	vars	n	mean	sd	median	trimmed	mad ...
age	1	1000	38.84	16.42	36	37.69	16.31
gender*	2	1000	1.45	0.50	1	1.43	0.00
income	3	816	113543.07	49842.29	93869	104841.94	28989.47

```
house*          4 1000     1.57     0.50      2     1.58      0.00

store_exp       5 1000  1356.85  2774.40    329   839.92    196.45

online_exp      6 1000  2120.18  1731.22   1942  1874.51   1015.21

store_trans     7 1000     5.35     3.70      4     4.89      2.97

online_trans    8 1000    13.55     7.96     14    13.42     10.38

...
```

It is easy to see the skewed variables. If `mean` and `trimmed` differ a lot, there is very likely outliers. By default, `trimmed` reports mean by dropping the top and bottom 10%. It can be adjusted by setting argument `trim=`. It is clear that `store_exp` has outliers.

As an example, we will apply Box-Cox transformation on `store_trans` and `online_trans`:

```
# select the two columns and save them as dat_bc
dat_bc <- subset(sim.dat, select = c("store_trans", "online_trans"))
(trans <- preProcess(dat_bc, method = c("BoxCox")))
```

```
## Created from 1000 samples and 2 variables
##
## Pre-processing:
##   - Box-Cox transformation (2)
##   - ignored (0)
##
## Lambda estimates for Box-Cox transformation:
## 0.1, 0.7
```

The last line of the output shows the estimates of λ for each variable. As before, use `predict()` to get the transformed result:

```
transformed <- predict(trans, dat_bc)
par(mfrow = c(1, 2), oma = c(2, 2, 2, 2))
hist(dat_bc$store_trans, main = "Before Transformation",
```

```
    xlab = "store_trans")
hist(transformed$store_trans, main = "After Transformation",
    xlab = "store_trans")
```

Before Transformation   After Transformation
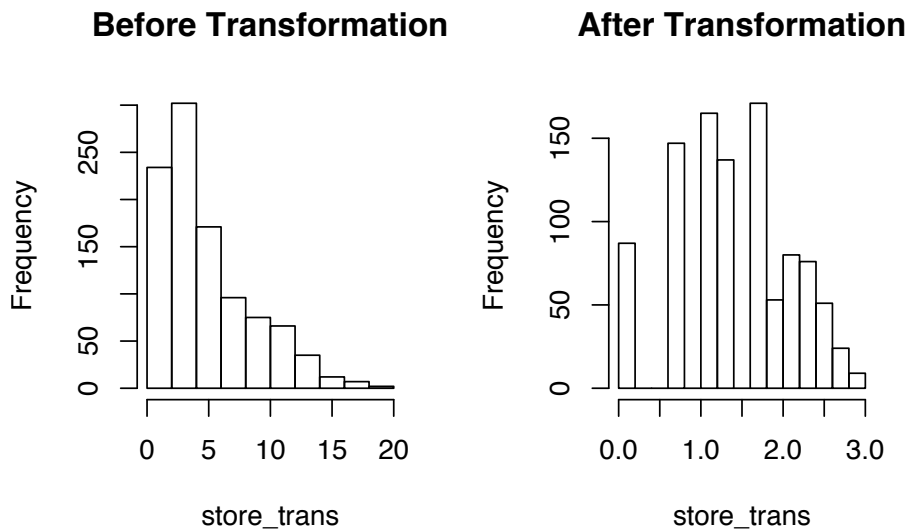
Before the transformation, the store_trans is skewed right. Box-CoxTrans () can also conduct Box-Cox transform. But note that BoxCoxTrans () can only be applied to a single variable, and it is not possible to transform difference columns in a data frame at the same time.

```
(trans <- BoxCoxTrans(dat_bc$store_trans))
```

```
## Box-Cox Transformation
##
## 1000 data points used to estimate Lambda
##
## Input data summary:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.00    4.00    5.35    7.00   20.00
##
```

```
## Largest/Smallest: 20
## Sample Skewness: 1.11
##
## Estimated Lambda: 0.1
## With fudge factor, Lambda = 0 will be used for transformations
```

```
transformed <- predict(trans, dat_bc$store_trans)
skewness(transformed)
```

```
## [1] -0.2155
```

The estimate of λ is the same as before (0.1). The skewness of the original observation is 1.1, and -0.2 after transformation. Although it is not strictly 0, it is greatly improved.

5.5 Resolve Outliers

Even under certain assumptions we can statistically define outliers, it can be hard to define in some situations. Box plot, histogram and some other basic visualizations can be used to initially check whether there are outliers. For example, we can visualize numerical non-survey variables in sim.dat:

```
# select numerical non-survey data
sdat <- subset(sim.dat, select = c("age", "income", "store_exp",
    "online_exp", "store_trans", "online_trans"))
# use scatterplotMatrix() function from car package
par(oma = c(2, 2, 1, 2))
scatterplotMatrix(sdat, diagonal = "boxplot", smoother = FALSE)
```
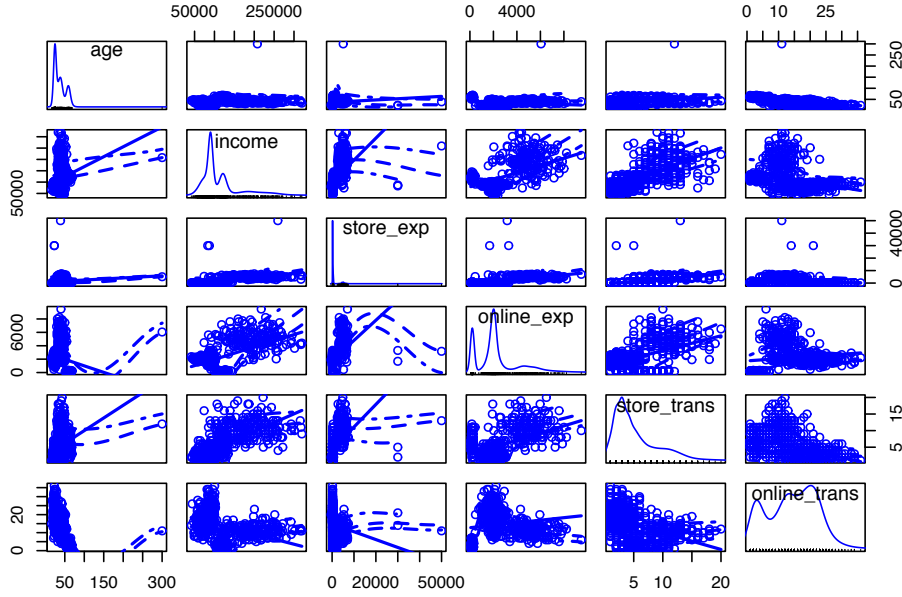
It is also easy to observe the pair relationship from the plot. `age` is negatively correlated with `online_trans` but positively correlated with `store_trans`. It seems that older people tend to purchase from the local store. The amount of expense is positively correlated with income. Scatterplot matrix like this can reveal lots of information before modeling.

In addition to visualization, there are some statistical methods to define outliers, such as the commonly used Z-score. The Z-score for variable **Y** is defined as:

$$Z_i = \frac{Y_i - \bar{Y}}{s}$$

where $\bar{Y}$ and s are mean and standard deviation for Y. Z-score is a measurement of the distance between each observation and the mean. This method may be misleading, especially when the sample size is small. Iglewicz and Hoaglin proposed to use the modified Z-score to determine the outlier(Iglewicz and Hoaglin, 1993)

$$M_i = \frac{0.6745(Y_i - \bar{Y})}{MAD}$$

Where MAD is the median of a series of $|Y_i - \bar{Y}|$, called the median of the absolute dispersion. Iglewicz and Hoaglin suggest that the points with the Z-score greater than 3.5 corrected above are possible outliers. Let's apply it to income:

```
# calculate median of the absolute dispersion for income
ymad <- mad(na.omit(sdat$income))
# calculate z-score
zs <- (sdat$income - mean(na.omit(sdat$income)))/ymad
# count the number of outliers
sum(na.omit(zs > 3.5))
```

```
## [1] 59
```

According to modified Z-score, variable income has 59 outliers. Refer to (Iglewicz and Hoaglin, 1993) for other ways of detecting outliers.

The impact of outliers depends on the model. Some models are sensitive to outliers, such as linear regression, logistic regression. Some are pretty robust to outliers, such as tree models, support vector machine. Also, the outlier is not wrong data. It is real observation so cannot be deleted at will. If a model is sensitive to outliers, we can use *spatial sign transformation* (Serneels S, 2006) to minimize the problem. It projects the original sample points to the surface of a sphere by:

$$x_{ij}^* = \frac{x_{ij}}{\sqrt{\sum_{j=1}^{p} x_{ij}^2}}$$

where x_{ij} represents the ith observation and j^{th} variable. As shown in the equation, every observation for sample i is divided by its square mode. The denominator is the Euclidean distance to the center of the p-dimensional predictor space. Three things to pay attention here:

1. It is important to center and scale the predictor data before using this transformation
2. Unlike centering or scaling, this manipulation of the predictors transforms them as a group
3. If there are some variables to remove (for example, highly correlated variables), do it before the transformation

Function `spatialSign()` `caret` package can conduct the transformation. Take `income` and `age` as an example:

```r
# KNN imputation
sdat <- sim.dat[, c("income", "age")]
imp <- preProcess(sdat, method = c("knnImpute"), k = 5)
sdat <- predict(imp, sdat)
transformed <- spatialSign(sdat)
transformed <- as.data.frame(transformed)
par(mfrow = c(1, 2), oma = c(2, 2, 2, 2))
plot(income ~ age, data = sdat, col = "blue", main = "Before")
plot(income ~ age, data = transformed, col = "blue", main = "After")
```
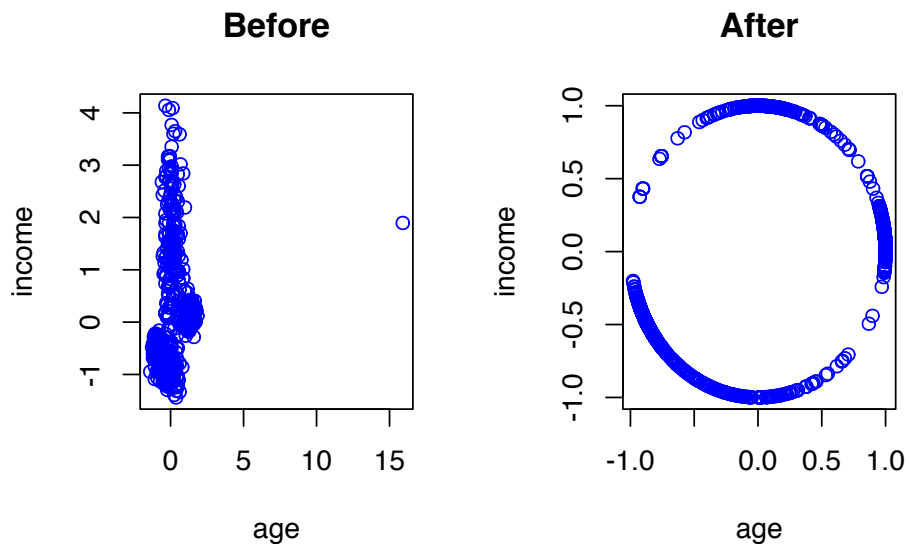

Some readers may have found that the above code does not seem

to standardize the data before transformation. Recall the introduction of KNN, `preProcess()` with `method="knnImpute"` by default will standardize data.

5.6   Collinearity

It is probably the technical term known by the most un-technical people. When two predictors are very strongly correlated, including both in a model may lead to confusion or problem with a singular matrix. There is an excellent function in `corrplot` package with the same name `corrplot()` that can visualize correlation structure of a set of predictors. The function has the option to reorder the variables in a way that reveals clusters of highly correlated ones.

```r
# select non-survey numerical variables
sdat <- subset(sim.dat, select = c("age", "income", "store_exp",
    "online_exp", "store_trans", "online_trans"))
# use bagging imputation here
imp <- preProcess(sdat, method = "bagImpute")
sdat <- predict(imp, sdat)
# get the correlation matrix
correlation <- cor(sdat)
# plot
par(oma = c(2, 2, 2, 2))
corrplot.mixed(correlation, order = "hclust", tl.pos = "lt",
    upper = "ellipse")
```
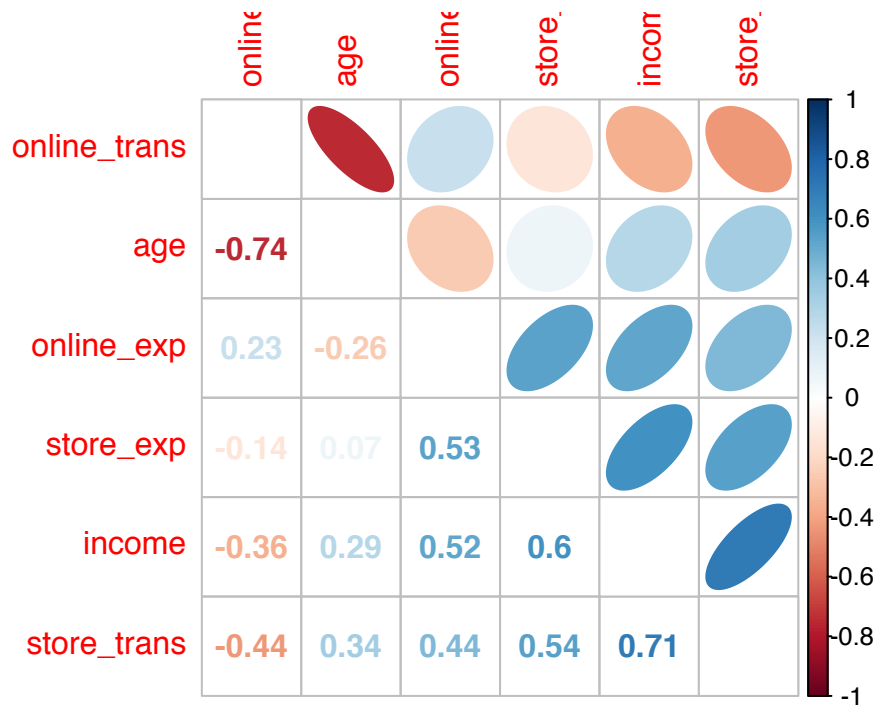
The closer the correlation is to 0, the lighter the color is and the closer the shape is to a circle. The elliptical means the correlation is not equal to 0 (because we set the upper = "ellipse"), the greater the correlation, the narrower the ellipse. Blue represents a positive correlation; red represents a negative correlation. The direction of the ellipse also changes with the correlation. The correlation coefficient is shown in the lower triangle of the matrix.

The variables relationship from previous scatter matrix are clear here: the negative correlation between age and online shopping, the positive correlation between income and amount of purchasing. Some correlation is very strong (such as the correlation between online_trans and age is -0.7414) which means the two variables contain duplicate information.

Section 3.5 of "Applied Predictive Modeling" (Kuhn and Johnston, 2013) presents a heuristic algorithm to remove a minimum number of predictors to ensure all pairwise correlations are below a certain threshold:

(1)   Calculate the correlation matrix of the predictors.
(2)   Determine the two predictors associated with the largest absolute pairwise correlation (call them predictors A and B).
(3)   Determine the average correlation between A and the other variables. Do the same for predictor B.
(4)   If A has a larger average correlation, remove it; otherwise, remove predictor B.
(5)   Repeat Step 2-4 until no absolute correlations are above the threshold.

The `findCorrelation()` function in package `caret` will apply the above algorithm.

```
(highCorr <- findCorrelation(cor(sdat), cutoff = 0.7))
```

```
## [1] 2 6
```

It returns the index of columns need to be deleted. It tells us that we need to remove the first column to make sure the correlations are all below 0.7.

```
# delete highly correlated columns
sdat <- sdat[-highCorr]
# check the new correlation matrix
(cor(sdat))
```

The absolute value of the elements in the correlation matrix after removal are all below 0.7. How strong does a correlation have to get, before you should start worrying about multicollinearity? There is no easy answer to that question. You can treat the threshold as a tuning parameter and pick one that gives you best prediction accuracy.

5.7   Sparse Variables

Other than the highly related predictors, predictors with degenerate distributions can cause the problem too. Removing those variables can significantly improve some models' performance and stability (such as linear regression and logistic regression but the tree based model is impervious to this type of predictors). One extreme example is a variable with a single value which is called zero-variance variable. Variables with very low frequency of unique values are near-zero variance predictors. In general, detecting those variables follows two rules:

- The fraction of unique values over the sample size
- The ratio of the frequency of the most prevalent value to the frequency of the second most prevalent value.

`nearZeroVar()` function in the `caret` package can filter near-zero variance predictors according to the above rules. In order to show the useage of the function, let's arbitaryly add some problematic variables to the origional data `sim.dat`:

```r
# make a copy
zero_demo <- sim.dat
# add two sparse variable zero1 only has one unique value zero2 is a
# vector with the first element 1 and the rest are 0s
zero_demo$zero1 <- rep(1, nrow(zero_demo))
zero_demo$zero2 <- c(1, rep(0, nrow(zero_demo) - 1))
```

The function will return a vector of integers indicating which columns to remove:

```
nearZeroVar(zero_demo,freqCut = 95/5, uniqueCut = 10)
```

```
## [1] 20 21
```

As expected, it returns the two columns we generated. You can go ahead to remove them. Note the two arguments in the function `freqCut =` and `uniqueCut =` are corresponding to the previous two rules.

- `freqCut`: the cutoff for the ratio of the most common value to the second most common value
- `uniqueCut`: the cutoff for the percentage of distinct values out of the number of total samples

5.8   Re-encode Dummy Variables

A dummy variable is a binary variable (0/1) to represent subgroups of the sample. Sometimes we need to recode categories to smaller bits of information named "dummy variables." For example, some questionnaires have five options for each question, A, B, C, D, and E. After you get the data, you will usually convert the corresponding categorical variables for each question into five nominal variables, and then use one of the options as the baseline.

Let's encode `gender` and `house` from `sim.dat` to dummy variables. There are two ways to implement this. The first is to use `class.ind()` from `nnet` package. However, it only works on one variable at a time.

```
dumVar <- nnet::class.ind(sim.dat$gender)
head(dumVar)
```

```
##      Female Male
## [1,]      1    0
## [2,]      1    0
## [3,]      0    1
## [4,]      0    1
## [5,]      0    1
## [6,]      0    1
```

Since it is redundant to keep both, we need to remove one of them when modeling. Another more powerful function is `dummy-Vars()` from `caret`:

```
# use "origional variable name + level" as new name
dumMod <- dummyVars(~gender + house + income,
                    data = sim.dat,
                    levelsOnly = F)
head(predict(dumMod, sim.dat))
```

```
##   gender.Female gender.Male house.No house.Yes income
## 1             1           0        0         1 120963
## 2             1           0        0         1 122008
## 3             0           1        0         1 114202
## 4             0           1        0         1 113616
## 5             0           1        0         1 124253
## 6             0           1        0         1 107661
```

`dummyVars()` can also use formula format. The variable on the right-hand side can be both categorical and numeric. For a numerical variable, the function will keep the variable unchanged. The advantage is that you can apply the function to a data frame without removing numerical variables. Other than that, the function can create interaction term:

```
dumMod <- dummyVars(~gender + house + income + income:gender,
                    data = sim.dat,
                    levelsOnly = F)
head(predict(dumMod, sim.dat))
```

```
##   gender.Female gender.Male house.No house.Yes income
## 1             1           0        0         1 120963
## 2             1           0        0         1 122008
## 3             0           1        0         1 114202
## 4             0           1        0         1 113616
## 5             0           1        0         1 124253
## 6             0           1        0         1 107661
##   genderFemale:income genderMale:income
## 1              120963                 0
## 2              122008                 0
## 3                   0            114202
## 4                   0            113616
## 5                   0            124253
## 6                   0            107661
```

If you think the impact income levels on purchasing behavior is different for male and female, then you may add the interaction term between income and gender. You can do this by adding income: gender in the formula.

6

Data Wrangling

This chapter focuses on some of the most frequently used data manipulations and shows how to implement them in R and Python. It is critical to explore the data with descriptive statistics (mean, standard deviation, etc.) and data visualization before analysis. Transform data so that the data structure is in line with the requirements of the model. You also need to summarize the results after analysis.

Load the R packages first:

```r
# install packages from CRAN
p_needed <- c('readr','dplyr','data.table','reshape2','tidyr')
packages <- rownames(installed.packages())
p_to_install <- p_needed[!(p_needed %in% packages)]
if (length(p_to_install) > 0) {
    install.packages(p_to_install)
}

lapply(p_needed, require, character.only = TRUE)
```

6.1 Read and write data

6.1.1 `readr`

You must be familiar with `read.csv()`, `read.table()` and `write.csv()` in base R. Here we will introduce a more efficient package

from RStudio in 2015 for reading and writing data: `readr` package. The corresponding functions are `read_csv()`, `read_table()` and `write_csv()`. The commands look quite similar, but `readr` is different in the following respects:

1. It is 10x faster. The trick is that `readr` uses C++ to process the data quickly.

2. It doesn't change the column names. The names can start with a number and "." will not be substituted to "_". For example:

```
read_csv("2015,2016,2017
1,2,3
4,5,6")
```

```
## # A tibble: 2 x 3
##   `2015` `2016` `2017`
##    <dbl>  <dbl>  <dbl>
## 1      1      2      3
## 2      4      5      6
```

1. `readr` functions do not convert strings to factors by default, are able to parse dates and times and can automatically determine the data types in each column.

2. The killing character, in my opinion, is that `readr` provides **progress bar**. What makes you feel worse than waiting is not knowing how long you have to wait.

```
> library(readr)
> read_csv("/Users/happyrabbit/Documents/GitHub/NLP/RawData/newagtalk-20150607/message.csv")
|========================================================|  74% 518 MB
```

The major functions of readr is to turn flat files into data frames:

- `read_csv()`: reads comma delimited files

- `read_csv2()`: reads semicolon separated files (common in countries where , is used as the decimal place)
- `read_tsv()`: reads tab delimited files
- `read_delim()`: reads in files with any delimiter
- `read_fwf()`: reads fixed width files. You can specify fields either by their widths with `fwf_widths()` or their position with `fwf_positions()`

- `read_table()`: reads a common variation of fixed width files where columns are separated by white space
- `read_log()`: reads Apache style log files

The good thing is that those functions have similar syntax. Once you learn one, the others become easy. Here we will focus on `read_csv()`.

The most important information for `read_csv()` is the path to your data:

```
sim.dat <- read_csv("http://bit.ly/2P5gTw4")
head(sim.dat)
```

```
# A tibble: 6 x 19
    age gender income house store_exp online_exp store_trans online_trans    Q1
  <int> <chr>   <dbl> <chr>    <dbl>      <dbl>       <int>        <int> <int>
1    57 Female 1.21e5 Yes       529.       304.           2            2     4
2    63 Female 1.22e5 Yes       478.       110.           4            2     4
3    59 Male   1.14e5 Yes       491.       279.           7            2     5
4    60 Male   1.14e5 Yes       348.       142.          10            2     5
5    51 Male   1.24e5 Yes       380.       112.           4            4     4
6    59 Male   1.08e5 Yes       338.       196.           4            5     4
# ... with 10 more variables: Q2 <int>, Q3 <int>, Q4 <int>, Q5 <int>, Q6 <int>,
#   Q7 <int>, Q8 <int>, Q9 <int>, Q10 <int>, segment <chr>
```

The function reads the file to R as a `tibble`. You can consider `tibble` as next iteration of the data frame. They are different with data frame for the following aspects:

- It never changes an input's type (i.e., no more `stringsAsFactors = FALSE!`)
- It never adjusts the names of variables
- It has a refined print method that shows only the first 10 rows and all the columns that fit on the screen. You can also control the default print behavior by setting options.

Refer to `http://r4ds.had.co.nz/tibbles.html` for more information about 'tibble'.

When you run `read_csv()` it prints out a column specification that gives the name and type of each column. To better understanding how `readr` works, it is helpful to type in some baby data set and check the results:

```r
dat <- read_csv("2015,2016,2017
100,200,300
canola,soybean,corn")
print(dat)
```

```
## # A tibble: 2 x 3
##   `2015` `2016`  `2017`
##   <chr>  <chr>   <chr>
## 1 100    200     300
## 2 canola soybean corn
```

You can also add comments on the top and tell R to skip those lines:

```r
dat <- read_csv("# I will never let you know that
         # my favorite food is carrot
         Date,Food,Mood
         Monday,carrot,happy
         Tuesday,carrot,happy
```

```
          Wednesday,carrot,happy
          Thursday,carrot,happy
          Friday,carrot,happy
          Saturday,carrot,extremely happy
          Sunday,carrot,extremely happy",
          skip = 2)
print(dat)
```

```
## # A tibble: 7 x 3
##   Date      Food   Mood
##   <chr>     <chr>  <chr>
## 1 Monday    carrot happy
## 2 Tuesday   carrot happy
## 3 Wednesday carrot happy
## 4 Thursday  carrot happy
## 5 Friday    carrot happy
## 6 Saturday  carrot extremely happy
## 7 Sunday    carrot extremely happy
```

If you don't have column names, set `col_names = FALSE` then R will assign names "x1","x2"… to the columns:

```
dat <- read_csv("Saturday,carrot,extremely happy
          Sunday,carrot,extremely happy", col_names = FALSE)
print(dat)
```

```
## # A tibble: 2 x 3
##   X1       X2     X3
##   <chr>    <chr>  <chr>
## 1 Saturday carrot extremely happy
## 2 Sunday   carrot extremely happy
```

You can also pass `col_names` a character vector which will be

used as the column names. Try to replace `col_names=FALSE` with `col_names=c("Date","Food","Mood")` and see what happen.

As mentioned before, you can use `read_csv2()` to read semicolon separated files:

```
dat <- read_csv2("Saturday; carrot; extremely happy \n Sunday; carrot; extremely happy", col_n
```

```
## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.
```

```
print(dat)
```

```
## # A tibble: 2 x 3
##   X1       X2      X3
##   <chr>    <chr>   <chr>
## 1 Saturday carrot extremely happy
## 2 Sunday   carrot extremely happy
```

Here "\n" is a convenient shortcut for adding a new line.

You can use `read_tsv()` to read tab delimited files

```
dat <- read_tsv("every\tman\tis\ta\tpoet\twhen\the\tis\tin\tlove\n",
    col_names = FALSE)
print(dat)
```

```
## # A tibble: 1 x 10
##   X1    X2    X3    X4    X5    X6    X7    X8    X9
##   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 every man   is    a     poet  when  he    is    in
## # ... with 1 more variable: X10 <chr>
```

Or more generally, you can use `read_delim()` and assign separating character

```
dat <- read_delim("THE|UNBEARABLE|RANDOMNESS|OF|LIFE\n",
    delim = "|", col_names = FALSE)
print(dat)
```

```
## # A tibble: 1 x 5
##   X1    X2        X3         X4    X5
##   <chr> <chr>     <chr>      <chr> <chr>
## 1 THE   UNBEARABLE RANDOMNESS OF    LIFE
```

Another situation you will often run into is the missing value. In marketing survey, people like to use "99" to represent missing. You can tell R to set all observation with value "99" as missing when you read the data:

```
dat <- read_csv("Q1,Q2,Q3
                5, 4,99",
                na = "99")
print(dat)
```

```
## # A tibble: 1 x 3
##      Q1    Q2 Q3
##   <dbl> <dbl> <lgl>
## 1     5     4 NA
```

For writing data back to disk, you can use `write_csv()` and `write_tsv()`. The following two characters of the two functions increase the chances of the output file being read back in correctly:

- Encode strings in UTF-8
- Save dates and date-times in ISO8601 format so they are easily parsed elsewhere

For example:

```r
write_csv(sim.dat, "sim_dat.csv")
```

For other data types, you can use the following packages:

- `Haven`: SPSS, Stata and SAS data
- `Readxl` and `xlsx`: excel data(.xls and .xlsx)
- `DBI`: given data base, such as RMySQL, RSQLite and RPost-greSQL, read data directly from the database using SQL

Some other useful materials:

- For getting data from the internet, you can refer to the book "XML and Web Technologies for Data Sciences with R".

- R data import/export manual[1]
- `rio` package https://github.com/leeper/rio

6.1.2 `data.table`— enhanced `data.frame`

What is `data.table`? It is an R package that provides an enhanced version of `data.frame`. The most used object in R is `data frame`. Before we move on, let's briefly review some basic characters and manipulations of data.frame:

- It is a set of rows and columns.
- Each row is of the same length and data type
- Every column is of the same length but can be of differing data types
- It has characteristics of both a matrix and a list
- It uses `[]` to subset data

---

[1]https://cran.r-project.org/doc/manuals/r-release/R-data.html#Acknowledgements

We will use the clothes customer data to illustrate. There are two dimensions in `[]`. The first one indicates the row and second one indicates column. It uses a comma to separate them.

```
# read data
sim.dat <- readr::read_csv("http://bit.ly/2P5gTw4")
```

```
## Parsed with column specification:
## cols(
##   age = col_double(),
##   gender = col_character(),
##   income = col_double(),
##   house = col_character(),
##   store_exp = col_double(),
##   online_exp = col_double(),
##   store_trans = col_double(),
##   online_trans = col_double(),
##   Q1 = col_double(),
##   Q2 = col_double(),
##   Q3 = col_double(),
##   Q4 = col_double(),
##   Q5 = col_double(),
##   Q6 = col_double(),
##   Q7 = col_double(),
##   Q8 = col_double(),
##   Q9 = col_double(),
##   Q10 = col_double(),
##   segment = col_character()
## )
```

```
# subset the first two rows
sim.dat[1:2, ]
# subset the first two rows and column 3 and 5
```

```r
sim.dat[1:2, c(3, 5)]
# get all rows with age>70
sim.dat[sim.dat$age > 70, ]
# get rows with age> 60 and gender is Male select column 3 and 4
sim.dat[sim.dat$age > 68 & sim.dat$gender == "Male", 3:4]
```

Remember that there are usually different ways to conduct the
same manipulation. For example, the following code presents three
ways to calculate an average number of online transactions for male
and female:

```r
tapply(sim.dat$online_trans, sim.dat$gender, mean)


aggregate(online_trans ~ gender, data = sim.dat, mean)


sim.dat %>%
  group_by(gender) %>%
  summarise(Avg_online_trans = mean(online_trans))
```

There is no gold standard to choose a specific function to manipu-
late data. The goal is to solve the real problem, not the tool itself.
So just use whatever tool that is convenient for you.

The way to use `[]` is straightforward. But the manipulations are
limited. If you need more complicated data reshaping or aggrega-
tion, there are other packages to use such as `dplyr`, `reshape2`, `tidyr`
etc. But the usage of those packages are not as straightforward as
`[]`. You often need to change functions. Keeping related operations
together, such as subset, group, update, join etc, will allow for:

- concise, consistent and readable syntax irrespective of the set of
  operations you would like to perform to achieve your end goal
- performing data manipulation fluidly without the cognitive bur-
  den of having to change among different functions

- by knowing precisely the data required for each operation, you can automatically optimize operations effectively

`data.table` is the package for that. If you are not familiar with other data manipulating packages and are interested in reducing programming time tremendously, then this package is for you.

Other than extending the function of `[]`, `data.table` has the following advantages:

- Offers fast import, subset, grouping, update, and joins for large data files
- It is easy to turn data frame to data table
- Can behave just like a data frame

You need to install and load the package:

Use `data.table()` to convert the existing data frame `sim.dat` to data table:

```
dt <- data.table(sim.dat)
class(dt)
```

```
## [1] "data.table" "data.frame"
```

Calculate mean for counts of online transactions:

```
dt[, mean(online_trans)]
```

```
## [1] 13.55
```

You can't do the same thing using data frame:

```
sim.dat[,mean(online_trans)]
```

```
Error in mean(online_trans) : object 'online_trans' not found
```

If you want to calculate mean by group as before, set "by =" argument:

```
dt[ , mean(online_trans), by = gender]
```

```
##     gender    V1
## 1: Female 15.38
## 2:   Male 11.26
```

You can group by more than one variables. For example, group by "gender" and "house":

```
dt[ , mean(online_trans), by = .(gender, house)]
```

```
##     gender house     V1
## 1: Female   Yes 11.312
## 2:   Male   Yes  8.772
## 3: Female    No 19.146
## 4:   Male    No 16.486
```

Assign column names for aggregated variables:

```
dt[ , .(avg = mean(online_trans)), by = .(gender, house)]
```

```
##     gender house    avg
## 1: Female   Yes 11.312
## 2:   Male   Yes  8.772
## 3: Female    No 19.146
## 4:   Male    No 16.486
```

`data.table` can accomplish all operations that `aggregate()` and `tapply()`can do for data frame.

- General setting of `data.table`

Different from data frame, there are three arguments for data table:

It is analogous to SQL. You don't have to know SQL to learn data table. But experience with SQL will help you understand data table. In SQL, you select column `j` (use command SELECT) for row `i` (using command WHERE). GROUP BY in SQL will assign the variable to group the observations.

Let's review our previous code:

```
dt[ , mean(online_trans), by = gender]
```

The code above is equal to the following SQL

```sql
SELECT  gender, avg(online_trans) FROM sim.dat GROUP BY gender
```

R code:

```r
dt[ , .(avg = mean(online_trans)), by = .(gender, house)]
```

is equal to SQL

```sql
SELECT gender, house, avg(online_trans) AS avg FROM sim.dat GROUP BY gender, house
```

R code

```r
dt[ age < 40, .(avg = mean(online_trans)), by = .(gender, house)]
```

is equal to SQL

```sql
SELECT gender, house, avg(online_trans) AS avg FROM sim.dat WHERE age < 40 GROUP BY gender, ho
```

You can see the analogy between `data.table` and `SQL`. Now let's focus on operations in data table.

- select row

```r
# select rows with age<20 and income > 80000
dt[age < 20 & income > 80000]
```

```
##     age gender income house store_exp online_exp
## 1:  19 Female  83535    No     227.7       1491
## 2:  18 Female  89416   Yes     209.5       1926
## 3:  19 Female  92813    No     186.7       1042
##     store_trans online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9
## 1:           1           22  2  1  1  2  4  1  4  2  4
## 2:           3           28  2  1  1  1  4  1  4  2  4
## 3:           2           18  3  1  1  2  4  1  4  3  4
##    Q10 segment
## 1:   1   Style
## 2:   1   Style
## 3:   1   Style
```

```
# select the first two rows
dt[1:2]
```

```
##     age gender income house store_exp online_exp
## 1:  57 Female 120963   Yes     529.1      303.5
## 2:  63 Female 122008   Yes     478.0      109.5
##     store_trans online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9
## 1:           2            2  4  2  1  2  1  4  1  4  2
## 2:           4            2  4  1  1  2  1  4  1  4  1
##    Q10 segment
## 1:   4   Price
## 2:   4   Price
```

- select column

Selecting columns in `data.table` don't need $:

```
# select column "age" but return it as a vector
# the argument for row is empty so the result will return all observations
ans <- dt[, age]
head(ans)
```

```
## [1] 57 63 59 60 51 59
```

To return `data.table` object, put column names in `list()`:

```r
# Select age and online_exp columns and return as a data.table instead
ans <- dt[, list(age, online_exp)]
head(ans)
```

Or you can also put column names in `.()`:

```r
ans <- dt[, .(age, online_exp)]
# head(ans)
```

To select all columns from "`age`" to "`income`":

```r
ans <- dt[, age:income, with = FALSE]
head(ans,2)
```

```
##    age gender income
## 1:  57 Female 120963
## 2:  63 Female 122008
```

Delete columns using `-` or `!`:

```r
# delete columns from  age to online_exp
ans <- dt[, -(age:online_exp), with = FALSE]
ans <- dt[, !(age:online_exp), with = FALSE]
```

- tabulation

In data table. `.N` means to count

```
# row count
dt[, .N]
```

```
## [1] 1000
```

If you assign the group variable, then it will count by groups:

```
# counts by gender
dt[, .N, by= gender]
```

```
##     gender   N
## 1: Female 554
## 2:   Male 446
```

```
# for those younger than 30, count by gender
 dt[age < 30, .(count=.N), by= gender]
```

```
##     gender count
## 1: Female   292
## 2:   Male    86
```

Order table:

```
# get records with the highest 5 online expense:
head(dt[order(-online_exp)],5)
```

```
   age gender    income house store_exp online_exp store_trans ...
1:  40 Female 217599.7    No  7023.684   9479.442          10
2:  41 Female       NA   Yes  3786.740   8638.239          14
3:  36   Male 228550.1   Yes  3279.621   8220.555           8
4:  31 Female 159508.1   Yes  5177.081   8005.932          11
5:  43 Female 190407.4   Yes  4694.922   7875.562           6
...
```

Since data table keep some characters of data frame, they share some operations:

```
dt[order(-online_exp)][1:5]
```

You can also order the table by more than one variable. The following code will order the table by `gender`, then order within `gender` by `online_exp`:

```
dt[order(gender, -online_exp)][1:5]
```

- Use `fread()` to import dat

Other than `read.csv` in base R, we have introduced 'read_csv' in 'readr'. `read_csv` is much faster and will provide progress bar which makes user feel much better (at least make me feel better). `fread()` in `data.table` further increase the efficiency of reading data. The following are three examples of reading the same data file `topic.csv`. The file includes text data scraped from an agriculture forum with 209670 rows and 6 columns:

```
system.time(topic<-read.csv("http://bit.ly/2zam5ny"))
```

```
   user   system elapsed
  3.561    0.051    4.888
```

```
system.time(topic<-readr::read_csv("http://bit.ly/2zam5ny"))
```

```
  user  system elapsed
 0.409   0.032   2.233
```

```
system.time(topic<-data.table::fread("http://bit.ly/2zam5ny"))
```

```
  user  system elapsed
 0.276   0.096   1.117
```

It is clear that `read_csv()` is much faster than `read.csv()`. `fread()` is a little faster than `read_csv()`. As the size increasing, the difference will become for significant. Note that `fread()` will read file as `data.table` by default.

6.2 Summarize data

6.2.1 `apply()`, `lapply()` and `sapply()` in base R

There are some powerful functions to summarize data in base R, such as `apply()`, `lapply()` and `sapply()`. They do the same basic things and are all from "apply" family: apply functions over parts of data. They differ in two important respects:

1. the type of object they apply to
2. the type of result they will return

When do we use `apply()`? When we want to apply a function to margins of an array or matrix. That means our data need to be structured. The operations can be very flexible. It returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

For example you can compute row and column sums for a matrix:

```r
## simulate a matrix
x <- cbind(x1 =1:8, x2 = c(4:1, 2:5))
dimnames(x)[[1]] <- letters[1:8]
apply(x, 2, mean)
```

```
##  x1  x2
## 4.5 3.0
```

```r
col.sums <- apply(x, 2, sum)
row.sums <- apply(x, 1, sum)
```

You can also apply other functions:

```r
ma <- matrix(c(1:4, 1, 6:8), nrow = 2)
ma
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    1    7
## [2,]    2    4    6    8
```

```r
apply(ma, 1, table)  #--> a list of length 2
```

```
## [[1]]
##
## 1 3 7
## 2 1 1
##
## [[2]]
##
## 2 4 6 8
## 1 1 1 1
```

```
apply(ma, 1, stats::quantile) # 5 x n matrix with rownames
```

```
##       [,1] [,2]
## 0%       1  2.0
## 25%      1  3.5
## 50%      2  5.0
## 75%      4  6.5
## 100%     7  8.0
```

Results can have different lengths for each call. This is a trickier example. What will you get?

```
## Example with different lengths for each call
z <- array(1:24, dim = 2:4)
zseq <- apply(z, 1:2, function(x) seq_len(max(x)))
zseq          ## a 2 x 3 matrix
typeof(zseq) ## list
dim(zseq) ## 2 3
zseq[1,]
apply(z, 3, function(x) seq_len(max(x)))
```

- `lapply()` applies a function over a list, data.frame or vector and returns a list of the same length.
- `sapply()` is a user-friendly version and wrapper of `lapply()`. By default it returns a vector, matrix or if `simplify = "array"`, an array if appropriate. `apply(x, f, simplify = FALSE, USE.NAMES = FALSE)` is the same as `lapply(x, f)`. If `simplify=TRUE`, then it will return a `data.frame` instead of `list`.

Let's use some data with context to help you better understand the functions.

- Get the mean and standard deviation of all numerical variables in the dataset.

```r
# Read data
sim.dat <- read.csv("http://bit.ly/2P5gTw4")
# Get numerical variables
sdat <- sim.dat[, !lapply(sim.dat, class) == "factor"]
## Try the following code with apply() function apply(sim.dat,2,class)
## What is the problem?
```

The data frame `sdat` only includes numeric columns. Now we can go head and use `apply()` to get mean and standard deviation for each column:

```r
apply(sdat, MARGIN=2,function(x) mean(na.omit(x)))
```

```
##          age         income     store_exp    online_exp
##    3.884e+01      1.135e+05     1.357e+03     2.120e+03
##  store_trans online_trans            Q1            Q2
##    5.350e+00      1.355e+01     3.101e+00     1.823e+00
##           Q3             Q4            Q5            Q6
##    1.992e+00      2.763e+00     2.945e+00     2.448e+00
##           Q7             Q8            Q9           Q10
##    3.434e+00      2.396e+00     3.085e+00     2.320e+00
```

Here we defined a function using `function(x) mean(na.omit(x))`. It is a very simple function. It tells R to ignore the missing value when calculating the mean. `MARGIN=2` tells R to apply the function to each column. It is not hard to guess what `MARGIN=1` mean. The result show that the average online expense is much higher than store expense. You can also compare the average scores across different questions. The command to calculate standard deviation is very similar. The only difference is to change `mean()` to `sd()`:

```r
apply(sdat, MARGIN=2,function(x) sd(na.omit(x)))
```

```
##           age        income      store_exp    online_exp
##        16.417     49842.287       2774.400      1731.224
##  store_trans online_trans             Q1            Q2
##         3.696         7.957          1.450         1.168
##            Q3            Q4             Q5            Q6
##         1.402         1.155          1.284         1.439
##            Q7            Q8             Q9           Q10
##         1.456         1.154          1.118         1.136
```

Even the average online expense is higher than store expense, the standard deviation for store expense is much higher than online expense which indicates there is very likely some big/small purchase in store. We can check it quickly:

```
summary(sdat$store_exp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    -500     205     329    1357     597   50000
```

```
summary(sdat$online_exp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      69     420    1942    2120    2441    9479
```

There are some odd values in store expense. The minimum value is -500 which indicates that you should preprocess data before analyzing it. Checking those simple statistics will help you better understand your data. It then gives you some idea how to preprocess and analyze them. How about using `lapply()` and `sapply()`?

Run the following code and compare the results:

```
lapply(sdat, function(x) sd(na.omit(x)))
sapply(sdat, function(x) sd(na.omit(x)))
sapply(sdat, function(x) sd(na.omit(x)), simplify = FALSE)
```

6.2.2 `dplyr` package

`dplyr` provides a flexible grammar of data manipulation focusing on tools for working with data frames (hence the `d` in the name). It is faster and more friendly:

- It identifies the most important data manipulations and make they easy to use from R
- It performs faster for in-memory data by writing key pieces in C++ using `Rcpp`
- The interface is the same for data frame, data table or database.

We will illustrate the following functions in order:

1. Display
2. Subset
3. Summarize
4. Create new variable
5. Merge

Display

- `tbl_df()`: Convert the data to `tibble` which offers better checking and printing capabilities than traditional data frames. It will adjust output width according to fit the current window.

```
tbl_df(sim.dat)
```

- `glimpse()`: This is like a transposed version of `tbl_df()`

```
glimpse(sim.dat)
```

Subset

Get rows with `income` more than 300000:

```
filter(sim.dat, income >300000) %>%
  tbl_df()
```

Here we meet a new operator `%>%`. It is called "Pipe operator" which pipes a value forward into an expression or function call. What you get in the left operation will be the first argument or the only argument in the right operation.

```
x %>% f(y) = f(x, y)
y %>% f(x, ., z) = f(x, y, z )
```

It is an operator from `magrittr` which can be really beneficial. Look at the following code. Can you tell me what it does?

```
ave_exp <- filter(
  summarise(
    group_by(
      filter(
        sim.dat,
        !is.na(income)
      ),
      segment
    ),
    ave_online_exp = mean(online_exp),
    n = n()
  ),
  n > 200
)
```

Now look at the identical code using "`%>%`":

```r
ave_exp <- sim.dat %>%
 filter(!is.na(income)) %>%
 group_by(segment) %>%
 summarise(
   ave_online_exp = mean(online_exp),
   n = n() ) %>%
  filter(n > 200)
```

Isn't it much more straightforward now? Let's read it:

1. Delete observations from `sim.dat` with missing income values
2. Group the data from step 1 by variable `segment`
3. Calculate mean of online expense for each segment and save the result as a new variable named `ave_online_exp`
4. Calculate the size of each segment and saved it as a new variable named `n`
5. Get segments with size larger than 200

You can use `distinct()` to delete duplicated rows.

```r
dplyr::distinct(sim.dat)
```

`sample_frac()` will randomly select some rows with a specified percentage. `sample_n()` can randomly select rows with a specified number.

```r
dplyr::sample_frac(sim.dat, 0.5, replace = TRUE)
dplyr::sample_n(sim.dat, 10, replace = TRUE)
```

`slice()` will select rows by position:

```
dplyr::slice(sim.dat, 10:15)
```

It is equivalent to `sim.dat[10:15,]`.

`top_n()` will select the order top n entries:

```
dplyr::top_n(sim.dat,2,income)
```

If you want to select columns instead of rows, you can use `select()`. The following are some sample codes:

```
# select by column name
dplyr::select(sim.dat,income,age,store_exp)

# select columns whose name contains a character string
dplyr::select(sim.dat, contains("_"))

# select columns whose name ends with a character string
# similar there is "starts_with"
dplyr::select(sim.dat, ends_with("e"))

# select columns Q1,Q2,Q3,Q4 and Q5
select(sim.dat, num_range("Q", 1:5))

# select columns whose names are in a group of names
dplyr::select(sim.dat, one_of(c("age", "income")))

# select columns between age and online_exp
dplyr::select(sim.dat, age:online_exp)

# select all columns except for age
dplyr::select(sim.dat, -age)
```

Summarize

A standard marketing problem is customer segmentation. It usually starts with designing survey and collecting data. Then run a cluster analysis on the data to get customer segments. Once we have different segments, the next is to understand how each group of customer look like by summarizing some key metrics. For example, we can do the following data aggregation for different segments of clothes customers.

```
sim.dat %>%
  group_by(segment) %>%
  summarise(Age = round(mean(na.omit(age)), 0),
            FemalePct = round(mean(gender == "Female"), 2),
            HouseYes = round(mean(house == "Yes"), 2),
            store_exp = round(mean(na.omit(store_exp), trim = 0.1), 0),
            online_exp = round(mean(online_exp), 0),
            store_trans = round(mean(store_trans), 1),
            online_trans = round(mean(online_trans), 1))
```

```
##    Age FemalePct HouseYes store_exp online_exp
## 1  39      0.55     0.57       840       2120
##    store_trans online_trans
## 1          5.3         13.5
```

Now, let's peel the onion.

The first line sim.dat is easy. It is the data you want to work on. The second line group_by(segment) tells R that in the following steps you want to summarise by variable segment. Here we only summarize data by one categorical variable, but you can group by multiple variables, such as group_by(segment, house). The third argument summarise tells R the manipulation(s) to do. Then list the exact actions inside summarise() . For example, Age=round(mean(na.omit(age)),0) tell R the following things:

1. Calculate the mean of column age ignoring missing value for each customer segment

2. Round the result to the specified number of decimal places
3. Store the result in a new variable named `Age`

The rest of the command above is similar. In the end, we calculate the following for each segment:

1. `Age`: average age for each segment
2. `FemalePct`: percentage for each segment
3. `HouseYes`: percentage of people who own a house
4. `stroe_exp`: average expense in store
5. `online_exp`: average expense online
6. `store_trans`: average times of transactions in the store
7. `online_trans`: average times of online transactions

There is a lot of information you can extract from those simple averages.

- Conspicuous: average age is about 40. It is a group of middle-age wealthy people. 1/3 of them are female, and 2/3 are male. They buy regardless the price. Almost all of them own house (0.86). It makes us wonder what is wrong with the rest 14%?

- Price: They are older people with average age 60. Nearly all of them own a house(0.94). They are less likely to purchase online (store_trans=6 while online_trans=3). It is the only group that is less likely to buy online.

- Quality: The average age is 35. They are not way different with Conspicuous regarding age. But they spend much less. The percentages of male and female are similar. They prefer online shopping. More than half of them don't own a house (0.66).

- Style: They are young people with average age 24. The majority of them are female (0.81). Most of them don't own a house (0.73). They are very likely to be digital natives and prefer online shopping.

You may notice that Style group purchase more frequently online

(`online_trans`) but the expense (`online_exp`) is not higher. It makes us wonder what is the average expense each time, so you have a better idea about the price range of the group.

The analytical process is aggregated instead of independent steps. The current step will shed new light on what to do next. Sometimes you need to go back to fix something in the previous steps. Let's check average one-time online and instore purchase amounts:

```r
sim.dat %>%
  group_by(segment) %>%
  summarise(avg_online = round(sum(online_exp)/sum(online_trans), 2),
            avg_store = round(sum(store_exp)/sum(store_trans), 2))
```

```
##   avg_online avg_store
## 1      156.5     253.6
```

Price group has the lowest averaged one-time purchase. The Conspicuous group will pay the highest price. When we build customer profile in real life, we will also need to look at the survey summarization. You may be surprised how much information simple data manipulations can provide.

Another comman task is to check which column has missing values. It requires the program to look at each column in the data. In this case you can use `summarise_all`:

```r
# apply function anyNA() to each column
# you can also assign a function vector such as: c("anyNA","is.factor")
dplyr::summarise_all(sim.dat, funs_(c("anyNA")))
```

```
##     age gender income house store_exp online_exp
## 1 FALSE  FALSE   TRUE FALSE     FALSE      FALSE
##   store_trans online_trans    Q1    Q2    Q3    Q4
## 1       FALSE        FALSE FALSE FALSE FALSE FALSE
##      Q5    Q6    Q7    Q8    Q9   Q10 segment
## 1 FALSE FALSE FALSE FALSE FALSE FALSE   FALSE
```

The above code returns a vector indicating if there is any value missing in each column.

Create new variable

There are often situations where you need to create new variables. For example, adding online and store expense to get total expense. In this case, you will apply **window function** to the columns and return a column with the same length. `mutate()` can do it for you and append one or more new columns:

```
dplyr::mutate(sim.dat, total_exp = store_exp + online_exp)
```

The above code sums up two columns and appends the result (`total_exp`) to sim.dat. Another similar function is `transmute()`. The difference is that `transmute()` will delete the original columns and only keep the new ones.

```
dplyr::transmute(sim.dat, total_exp = store_exp + online_exp)
```

Merge

Similar to SQL, there are different joins in `dplyr`. We create two baby data sets to show how the functions work.

```
(x <- data.frame(cbind(ID = c("A", "B", "C"), x1 = c(1, 2, 3))))
```

```
##   ID x1
## 1  A  1
## 2  B  2
## 3  C  3
```

```
(y <- data.frame(cbind(ID = c("B", "C", "D"), y1 = c(T, T, F))))
```

```
##   ID    y1
## 1  B   TRUE
## 2  C   TRUE
## 3  D  FALSE
```

```
# join to the left
# keep all rows in x
left_join(x, y, by = "ID")
```

```
## Warning: Column `ID` joining factors with different
## levels, coercing to character vector
```

```
##   ID x1    y1
## 1  A  1  <NA>
## 2  B  2  TRUE
## 3  C  3  TRUE
```

```
# get rows matched in both data sets
inner_join(x, y, by = "ID")
```

```
## Warning: Column `ID` joining factors with different
## levels, coercing to character vector
```

```
##   ID x1    y1
## 1  B  2  TRUE
## 2  C  3  TRUE
```

```r
# get rows in either data set
full_join(x, y, by = "ID")
```

```
## Warning: Column `ID` joining factors with different
## levels, coercing to character vector
```

```
##   ID   x1    y1
## 1  A    1  <NA>
## 2  B    2  TRUE
## 3  C    3  TRUE
## 4  D <NA> FALSE
```

```r
# filter out rows in x that can be matched in y
# it doesn't bring in any values from y
semi_join(x, y, by = "ID")
```

```r
# the opposite of  semi_join()
# it gets rows in x that cannot be matched in y
# it doesn't bring in any values from y
anti_join(x, y, by = "ID")
```

There are other functions(`intersect()`, `union()` and `setdiff()`). Also the data frame version of `rbind` and `cbind` which are `bind_rows()` and `bind_col()`. We are not going to go through them all. You can try them yourself. If you understand the functions we introduced so far. It should be easy for you to figure out the rest.

6.3 Tidy and Reshape Data

"Tidy data" represent the information from a dataset as data frames where each row is an observation, and each column contains the values of a variable (i.e., an attribute of what we are observing). Depending on the situation, the requirements on what to present as rows and columns may change. To make data easy to work with for the problem at hand, in practice, we often need to convert data between the "wide" and the "long" format. The process feels like kneading the dough.

There are two commonly used packages for this kind of manipulations: `tidyr` and `reshape2`. We will show how to tidy and reshape data using the two packages. By comparing the functions to show how they overlap and where they differ.

6.3.1 `reshape2` package

It is a reboot of the previous package `reshape`. Take a baby subset of our exemplary clothes consumers data to illustrate:

```
sdat<-sim.dat[1:5,1:6]
```

For the above data `sdat`, what if we want to have a variable indicating the purchasing channel (i.e. online or in-store) and another column with the corresponding expense amount? Assume we want to keep the rest of the columns the same. It is a task to change data from "wide" to "long". There are two general ways to shape data:

- Use `melt()` to convert an object into a molten data frame, i.e., from wide to long
- Use `dcast()` to cast a molten data frame into the shape you want, i.e., from long to wide

```
(mdat <- melt(sdat, measure.vars = c("store_exp", "online_exp"),
              variable.name = "Channel",
              value.name = "Expense"))
```

```
##      age gender income house    Channel Expense
## 1    57 Female 120963   Yes  store_exp   529.1
## 2    63 Female 122008   Yes  store_exp   478.0
## 3    59   Male 114202   Yes  store_exp   490.8
## 4    60   Male 113616   Yes  store_exp   347.8
## 5    51   Male 124253   Yes  store_exp   379.6
## 6    57 Female 120963   Yes online_exp   303.5
## 7    63 Female 122008   Yes online_exp   109.5
## 8    59   Male 114202   Yes online_exp   279.2
## 9    60   Male 113616   Yes online_exp   141.7
## 10   51   Male 124253   Yes online_exp   112.2
```

You melted the data frame `sdat` by two variables: `store_exp` and `online_exp` (`measure.vars=c("store_exp","online_exp")`). The new variable name is `Channel` set by command `variable.name = "Channel"`. The value name is `Expense` set by command `value.name = "Expense"`.

You can run a regression to study the effect of purchasing channel as follows:

```
# Here we use all observations from sim.dat
# Don't show result here
mdat <- melt(sim.dat[, 1:6], measure.vars = c("store_exp", "online_exp"),
    variable.name = "Channel", value.name = "Expense")
fit <- lm(Expense ~ gender + house + income + Channel + age, data = mdat)
summary(fit)
```

You can `melt()` list, matrix, table too. The syntax is similar, and we won't go through every situation. Sometimes we want to convert the data from "long" to "wide". For example, **you want to**

compare the online and in-store expense between male
and female based on the house ownership.

```
dcast(mdat, house + gender ~ Channel, sum)
```

```
## Using Expense as value column: use value.var to override.
```

```
##   house gender store_exp online_exp
## 1   Yes Female      1007      413.0
## 2   Yes   Male      1218      533.2
```

In the above code, what is the left side of ~ are variables that
you want to group by. The right side is the variable you want to
spread as columns. It will use the column indicating value from
`melt()` before. Here is "`Expense`".

6.3.2 `tidyr` package

The other package that will do similar manipulations is `tidyr`. Let's
get a subset to illustrate the usage.

```
# practice functions we learnt before
sdat <- sim.dat[1:5, ] %>%
  dplyr::select(age, gender, store_exp, store_trans)
sdat %>% tbl_df()
```

```
## # A tibble: 5 x 4
##     age gender store_exp store_trans
##   <int> <fct>      <dbl>       <int>
## 1    57 Female      529.           2
## 2    63 Female      478.           4
## 3    59 Male        491.           7
## 4    60 Male        348.          10
## 5    51 Male        380.           4
```

gather() function in tidyr is analogous to melt() in reshape2. The following code will do the same thing as we did before using melt():

```
msdat <- tidyr::gather(sdat, "variable","value", store_exp, store_trans)
msdat %>% tbl_df()
```

```
## # A tibble: 10 x 4
##       age gender variable    value
##    <int> <fct>  <chr>       <dbl>
## 1     57 Female store_exp    529.
## 2     63 Female store_exp    478.
## 3     59 Male   store_exp    491.
## 4     60 Male   store_exp    348.
## 5     51 Male   store_exp    380.
## 6     57 Female store_trans    2
## 7     63 Female store_trans    4
## 8     59 Male   store_trans    7
## 9     60 Male   store_trans   10
## 10    51 Male   store_trans    4
```

Or if we use the pipe operation, we can write the above code as:

```
sdat %>% gather("variable", "value", store_exp, store_trans)
```

It is identical with the following code using melt():

```
melt(sdat, measure.vars = c("store_exp", "store_trans"),
     variable.name = "variable",
     value.name = "value")
```

The opposite operation to gather() is spread(). The previous one stacks columns and the latter one spread the columns.

```
msdat %>% spread(variable, value)
```

```
##   age gender store_exp store_trans
## 1  51   Male     379.6           4
## 2  57 Female     529.1           2
## 3  59   Male     490.8           7
## 4  60   Male     347.8          10
## 5  63 Female     478.0           4
```

Another pair of functions that do opposite manipulations are `separate()` and `unite()`.

```
sepdat<- msdat %>%
  separate(variable,c("Source","Type"))
sepdat %>% tbl_df()
```

```
## # A tibble: 10 x 5
##      age gender Source Type  value
##    <int> <fct>  <chr>  <chr> <dbl>
## 1     57 Female store  exp    529.
## 2     63 Female store  exp    478.
## 3     59 Male   store  exp    491.
## 4     60 Male   store  exp    348.
## 5     51 Male   store  exp    380.
## 6     57 Female store  trans    2
## 7     63 Female store  trans    4
## 8     59 Male   store  trans    7
## 9     60 Male   store  trans   10
## 10    51 Male   store  trans    4
```

You can see that the function separates the original column "`variable`" to two new columns "`Source`" and "`Type`". You can use `sep=` to set the string or regular expression to separate the column. By default, it is "`_`".

The `unite()` function will do the opposite: combining two columns. It is the generalization of `paste()` to a data frame.

```
sepdat %>%
  unite("variable", Source, Type, sep = "_")
```

```
##     age gender     variable value
## 1   57 Female    store_exp 529.1
## 2   63 Female    store_exp 478.0
## 3   59   Male    store_exp 490.8
## 4   60   Male    store_exp 347.8
## 5   51   Male    store_exp 379.6
## 6   57 Female store_trans   2.0
## 7   63 Female store_trans   4.0
## 8   59   Male store_trans   7.0
## 9   60   Male store_trans  10.0
## 10  51   Male store_trans   4.0
```

The reshaping manipulations may be the trickiest part. You have to practice a lot to get familiar with those functions. Unfortunately, there is no shortcut.

7

Model Tuning Strategy

When training a machine learning model, there are many decisions to make. For example, when training a random forest, you need to decide the number of trees and the number of variables at each node. For lasso method, you need to determine the penalty parameter. There may be standard settings for some of the parameters, but it's unlikely to guess the right values for all of these correctly. Other than that, making good choices on how you split the data into training and testing sets can make a huge difference in helping you find a high-performance model efficiently.

This chapter will illustrate the practical aspects of model tuning. We will talk about different types of model error, sources of model error, hyperparameter tuning, how to set up your data and how to make sure your model implementation is correct. In practice applying machine learning is a highly iterative process.

7.1 Systematic Error and Random Error

Assume $\mathbf{X}$ is $n \times p$ observation matrix and $\mathbf{y}$ is response variable, we have:

$$\mathbf{y} = f(\mathbf{X}) + \tag{7.1}$$

where   is the random error with a mean of zero. The function $f(\cdot)$ is our modeling target, which represents the information in the response variable that predictors can explain. The main goal of estimating $f(\cdot)$ is inference or prediction, or sometimes both. In

general, there is a trade-off between flexibility and interpretability of the model. So data scientists need to comprehend the delicate balance between these two.

Depending on the modeling purposes, the requirement for interpretability varies. If the prediction is the only goal, then as long as the prediction is accurate enough, the interpretability is not under consideration. In this case, people can use "black box" model, such as random forest, boosting tree, neural network and so on. These models are very flexible but nearly impossible to explain. Their accuracy is usually higher on the training set, but not necessary when it predicts. It is not surprising since those models have a huge number of parameters and high flexibility that they can "memorize" the entire training data. A paper by Chiyuan Zhang et al. in 2017 pointed out that "Deep neural networks (even just two-layer net) easily fit random labels" (Zhang et al., 2017). The traditional forms of regularization, such as weight decay, dropout, and data augmentation, fail to control generalization error. It poses a conceptual challenge to statistical theory and also calls our attention when we use such black-box models.

There are two kinds of application problems: complete information problem and incomplete information problem. The complete information problem has all the information you need to know the correct response. Take the famous cat recognition, for example, all the information you need to identify a cat is in the picture. In this situation, the algorithm that penetrates the data the most wins. There are some other similar problems such as the self-driving car, chess game, facial recognition and speech recognition. But in most of the data science applications, the information is incomplete. If you want to know whether a customer is going to purchase again or not, it is unlikely to have 360-degree of the customer's information. You may have their historical purchasing record, discounts and service received. But you don't know if the customer sees your advertisement, or has a friend recommends competitor's product, or encounters some unhappy purchasing experience somewhere. There could be a myriad of factors that will influence the customer's purchase decision while what you have as data is only a

small part. To make things worse, in many cases, you don't even know what you don't know. Deep learning doesn't have any advantage in solving those problems. Instead, some parametric models often work better in this situation. You will comprehend this more after learning the different types of model error. Assume we have $\hat{f}$ which is an estimator of f. Then we can further get $\hat{\mathbf{y}} = \hat{f}(\mathbf{X})$. The predicted error is divided into two parts, systematic error, and random error:

$$E(\mathbf{y} - \hat{\mathbf{y}})^2 = E[f(\mathbf{X}) + \ - \hat{f}(\mathbf{X})]^2 = \underbrace{E[f(\mathbf{X}) - \hat{f}(\mathbf{X})]^2}_{(1)} + \underbrace{Var(\ )}_{(2)}$$

$$(7.2)$$

It is also called Mean Square Error (MSE) where (1) is the systematic error. It exists because $\hat{f}$ usually does not entirely describe the "systematic relation" between X and y which refers to the stable relationship that exists across different samples or time. Model improvement can help reduce this kind of error; (2) is the random error which represents the part of y that cannot be explained by X. A more complex model does not reduce the error. There are three reasons for random error:

1. the current sample is not representative, so the pattern in one sample set does not generalize to a broader scale.
2. The information is incomplete. In other words, you don't have all variables needed to explain the response.
3. Measurement error in the variables.

Deep learning has significant success solving problems with complete information and usually low measurement error. As mentioned before, in a task like image recognition, all you need are the pixels in the pictures. So in deep learning applications, increasing the sample size can improve the model performance significantly. But it may not perform well in problems with incomplete information. The biggest problem with the black-box model is that it fits random error, i.e., over-fitting. The notable feature of random error is that it varies over different samples. So one way to determine whether overfitting happens is to reserve a part of the data as the

test set and then check the performance of the trained model on the test data. Note that overfitting is a general problem from which any model could suffer. However, since black-box models usually have a large number of parameters, it is much more suspectable to over-fitting.
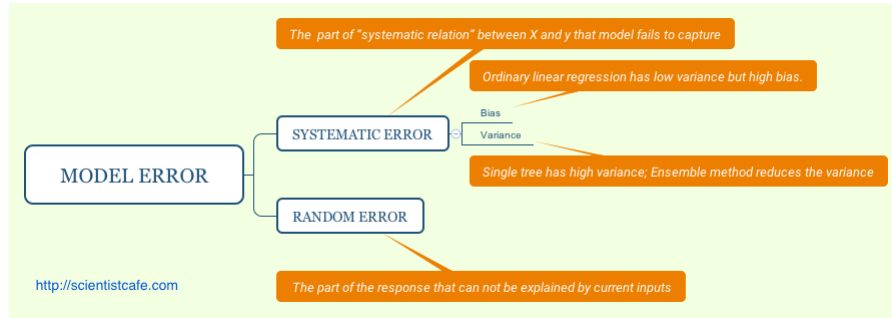

FIGURE 7.1: Types of Model Error

7.1.1   Variance-Bias Trade-Off

The systematic error can be further decomposed as:

$$
\begin{aligned}
E[f(\mathbf{X}) - \hat{f}(\mathbf{X})]^2 &= \quad E\left(f(\mathbf{X}) - E[\hat{f}(\mathbf{X})] + E[\hat{f}(\mathbf{X})] - \hat{f}(\mathbf{X})\right)^2 \\
&= E\left(E[\hat{f}(\mathbf{X})] - f(\mathbf{X})\right)^2 + E\left(\hat{f}(\mathbf{X}) - E[\hat{f}(\mathbf{X})]\right)^2 \\
&= \quad [Bias(\hat{f}(\mathbf{X}))]^2 + Var(\hat{f}(\mathbf{X}))
\end{aligned}
\tag{7.3}
$$

The systematic error consists of two parts, $Bias(\hat{f}(\mathbf{X}))$ and $Var(\hat{f}(\mathbf{X}))$. To minimize the systematic error, we need to minimize both. The bias represents the error caused by the model's approximation of the reality, i.e., systematic relation, which may be very complex. For example, linear regression assumes a linear relationship between the predictors and the response, but rarely is there a perfect linear relationship in real life. So linear regression is more likely to have a high bias. In general, the more flexible the model is, the higher the variance. However, this does not guarantee complex models will outperform a much simpler one, such

as linear regression. If the real relationship f is linear, then linear regression is unbiased. It is difficult for a more flexible model to compete. Good learning method requires low variance and bias. However, it is easy to find a model with extremely low bias but high variance (by fitting a tree) or a method with low variance but high variance (by fitting a straight line). That why we call it trade-off.

To explore bias and variance, let's begin with a simple simulation. We will simulate a data with a non-linear relationship and fit different models on it. An intuitive way to show these is to compare the plots of various models.

The code below simulates one predictor (x) and one response variable (fx). The relationship between x and fx is non-linear.

```r
source(ids_url('R/multiplot.r'))
# randomly simulate some non-linear samples
x = seq(1, 10, 0.01) * pi
e = rnorm(length(x), mean = 0, sd = 0.2)
fx <- sin(x) + e + sqrt(x)
dat = data.frame(x, fx)
```

Then fit a simple linear regression on these data:

```r
# plot fitting result
library(ggplot2)
ggplot(dat, aes(x, fx)) + geom_point() + geom_smooth(method = "lm", se = FALSE)
```

Despite a large sample size, trained linear regression cannot describe the relationship very well. In other words, in this case, the model has a high bias (Fig. 7.2). People also call it underfitting.

Since the estimated parameters will be somewhat different for different samples, there is the variance of estimates. Intuitively, it gives you some sense that if we fit the same model with different
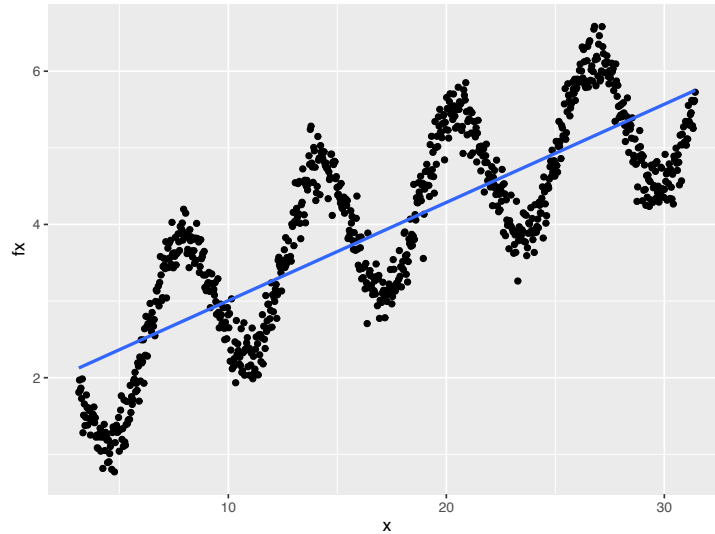
FIGURE 7.2: High bias model

samples (presumably, they are from the same population), how much will the estimates change. Ideally, the change is trivial. For high variance models, small changes in the training data result in very different estimates. In general, a model with high flexibility also has high variance., such as the CART tree, and the initial boosting method. To overcome that problem, the Random Forest and Gradient Boosting Model aim to reduce the variance by summarizing the results obtained from different samples.

Let's fit the above data using a smoothing method which is highly flexible and can fit the current data tightly:

```
ggplot(dat, aes(x, fx)) + geom_smooth(span = 0.03)
```

The resulting plot (Fig. 7.3) indicates the smoothing method fit the data much better so it has a much smaller bias. However, this method has a high variance. If we simulate different subsets of the sample, the result curve will change significantly:
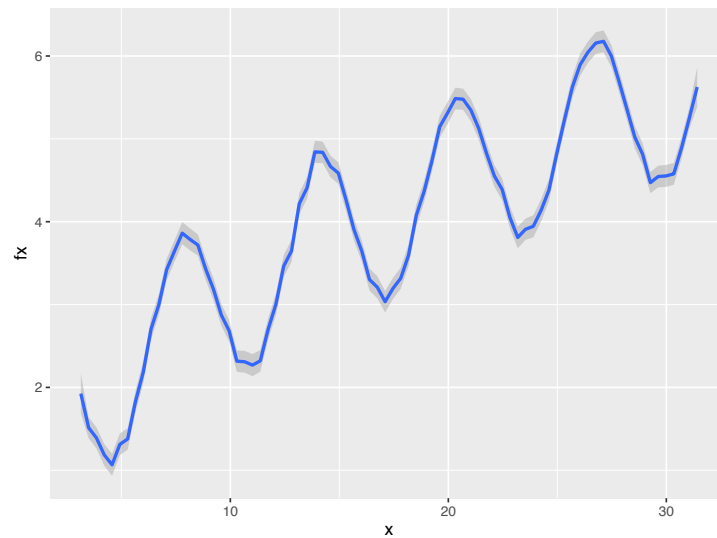
FIGURE 7.3: High variance model

```
# set random seed
set.seed(2016)
# sample part of the data to fit model sample 1
idx1 = sample(1:length(x), 100)
dat1 = data.frame(x1 = x[idx1], fx1 = fx[idx1])
p1 = ggplot(dat1, aes(x1, fx1)) + geom_smooth(span = 0.03)
# sample 2
idx2 = sample(1:length(x), 100)
dat2 = data.frame(x2 = x[idx2], fx2 = fx[idx2])
p2 = ggplot(dat2, aes(x2, fx2)) + geom_smooth(span = 0.03)
# sample 3
idx3 = sample(1:length(x), 100)
dat3 = data.frame(x3 = x[idx3], fx3 = fx[idx3])
p3 = ggplot(dat3, aes(x3, fx3)) + geom_smooth(span = 0.03)
# sample 4
idx4 = sample(1:length(x), 100)
dat4 = data.frame(x4 = x[idx4], fx4 = fx[idx4])
p4 = ggplot(dat4, aes(x4, fx4)) + geom_smooth(span = 0.03)
multiplot(p1, p2, p3, p4, cols = 2)
```
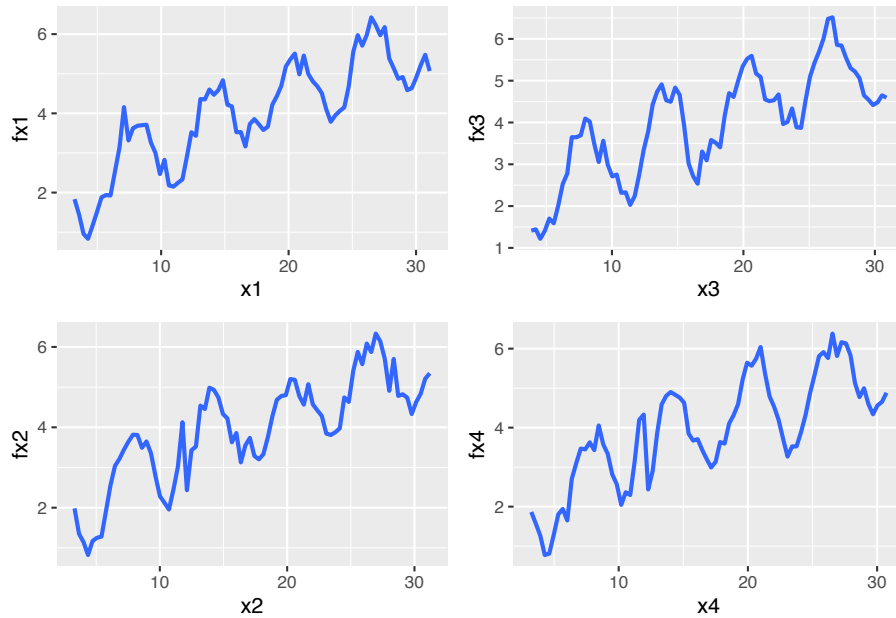
The fitted lines (blue) change over different samples which means it
has high variance. People also call it overfitting. Fitting the linear
model using the same four subsets, the result barely changes:

```
p1 = ggplot(dat1, aes(x1, fx1)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE)
p2 = ggplot(dat2, aes(x2, fx2)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE)
p3 = ggplot(dat3, aes(x3, fx3)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE)
p4 = ggplot(dat4, aes(x4, fx4)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE)
multiplot(p1, p2, p3, p4, cols = 2)
```
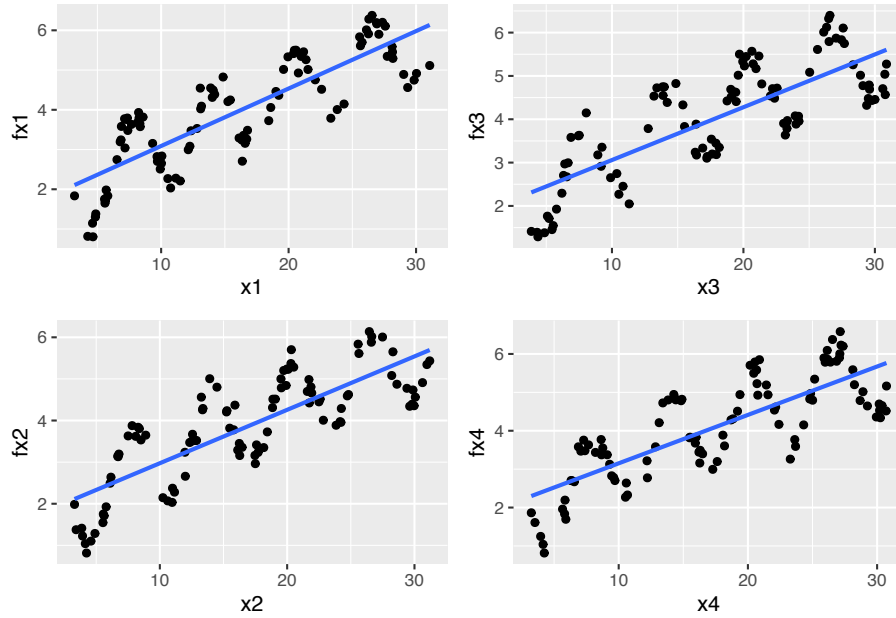
In general, the variance $(Var(\widehat{f}(\mathbf{X})))$ **increases** and the bias $(Bias(\widehat{f}(\mathbf{X})))$ **decreases** as the model flexibility increases. Variance and bias together determine the systematic error. As we increase the flexibility of the model, at first the rate at which $Bias(\widehat{f}(\mathbf{X}))$ decreases is faster than $Var(\widehat{f}(\mathbf{X}))$, so the MSE decreases. However, to some degree, higher flexibility has little effect on $Bias(\widehat{f}(\mathbf{X}))$ but $Var(\widehat{f}(\mathbf{X}))$ increases significantly, so the MSE increases.

7.1.2   Measurement Error in the Response

The measurement error in the response contributes to the random error ( ). This part of the error is irreducible if you change the data collection mechanism, and so it makes the root mean square error (RMSE) and R^2 have the corresponding upper and lower limits. RMSE and R^2 are commonly used performance measures for the regression model which we will talk in more detail later. Therefore, the random error term not only represents the part of fluctuations the model cannot explain but also contains measurement error in the response variables. Section 20.2 of Applied Predictive Modeling

(Kuhn and Johnston, 2013) has an example that shows the effect of the measurement error in the response variable on the model performance (RMSE and R^2).

The authors increased the error in the response proportional to a base level error which was gotten using the original data without introducing extra noise. Then fit a set of models repeatedly using the "contaminated" data sets to study the change of $RMSE$ and R^2 as the level of noise. Here we use clothing consumer data for a similar illustration. Suppose many people do not want to disclose their income and so we need to use other variables to establish a model to predict income. We set up the following model:

```r
# load data
sim.dat <- read.csv("http://bit.ly/2P5gTw4")
ymad <- mad(na.omit(sim.dat$income))
# calculate z-score
zs <- (sim.dat$income - mean(na.omit(sim.dat$income)))/ymad
# which(na.omit(zs>3.5)): identify outliers which(is.na(zs)):
# identify missing values
idex <- c(which(na.omit(zs > 3.5)), which(is.na(zs)))
# delete rows with outliers and missing values
sim.dat <- sim.dat[-idex, ]
fit <- lm(income ~ store_exp + online_exp + store_trans + online_trans,
    data = sim.dat)
```

The output shows that without additional noise, the root mean square error (RMSE) of the model is 29567, R^2 is 0.6.

Let's add various degrees of noise (0 to 3 times the RMSE) to the variable `income`:

$$RMSE \times (0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0)$$

```r
noise <- matrix(rep(NA, 7 * nrow(sim.dat)), nrow = nrow(sim.dat),
    ncol = 7)
for (i in 1:nrow(sim.dat)) {
    noise[i, ] <- rnorm(7, rep(0, 7), summary(fit)$sigma * seq(0,
        3, by = 0.5))
}
```

We then examine the effect of noise intensity on R^2 for models with different complexity. The models with complexity from low to high are: ordinary linear regression, partial least square regression(PLS), multivariate adaptive regression spline (MARS), support vector machine (SVM, the kernel function is radial basis function), and random forest.

```r
# fit ordinary linear regression
rsq_linear <- rep(0, ncol(noise))
for (i in 1:7) {
    withnoise <- sim.dat$income + noise[, i]
    fit0 <- lm(withnoise ~ store_exp + online_exp + store_trans +
        online_trans, data = sim.dat)
    rsq_linear[i] <- summary(fit0)$adj.r.squared
}
```

PLS is a method of linearizing nonlinear relationships through hidden layers. It is similar to the principal component regression (PCR), except that PCR does not take into account the information of the dependent variable when selecting the components, and its purpose is to find the linear combinations (i.e., unsupervised) that capture the most variance of the independent variables. When the independent variables and response variables are related, PCR can well identify the systematic relationship between them. However, when there exist independent variables not associated with response variable, it will undermine PCR's performance. And PLS maximizes the linear combination of dependencies with the re-

sponse variable. In the current case, the more complicated PLS
does not perform better than simple linear regression.

```r
# pls: conduct PLS and PCR
library(pls)
rsq_pls <- rep(0, ncol(noise))
# fit PLS
for (i in 1:7) {
    withnoise <- sim.dat$income + noise[, i]
    fit0 <- plsr(withnoise ~ store_exp + online_exp + store_trans +
        online_trans, data = sim.dat)
    rsq_pls[i] <- max(drop(R2(fit0, estimate = "train", intercept = FALSE)$val))
}
```

```r
# earth: fit mars
library(earth)
rsq_mars <- rep(0, ncol(noise))
for (i in 1:7) {
    withnoise <- sim.dat$income + noise[, i]
    fit0 <- earth(withnoise ~ store_exp + online_exp + store_trans +
        online_trans, data = sim.dat)
    rsq_mars[i] <- fit0$rsq
}
```

```r
# caret: awesome package for tuning predictive model
library(caret)
rsq_svm <- rep(0, ncol(noise))
# Need some time to run
for (i in 1:7) {
    idex <- which(is.na(sim.dat$income))
    withnoise <- sim.dat$income + noise[, i]
    trainX <- sim.dat[, c("store_exp", "online_exp", "store_trans",
```

```
        "online_trans")]
    trainY <- withnoise
    fit0 <- train(trainX, trainY, method = "svmRadial", tuneLength = 15,
        trControl = trainControl(method = "cv"))
    rsq_svm[i] <- max(fit0$results$Rsquared)
}
```

```
# randomForest: random forest model
library(randomForest)
rsq_rf <- rep(0, ncol(noise))
# ntree=500 number of trees na.action = na.omit ignore
# missing value
for (i in 1:7) {
    withnoise <- sim.dat$income + noise[, i]
    fit0 <- randomForest(withnoise ~ store_exp + online_exp +
        store_trans + online_trans, data = sim.dat, ntree = 500,
        na.action = na.omit)
    rsq_rf[i] <- tail(fit0$rsq, 1)
}
library(reshape2)
rsq <- data.frame(cbind(Noise = c(0, 0.5, 1, 1.5, 2, 2.5, 3),
    rsq_linear, rsq_pls, rsq_mars, rsq_svm, rsq_rf))
rsq <- melt(rsq, id.vars = "Noise", measure.vars = c("rsq_linear",
    "rsq_pls", "rsq_mars", "rsq_svm", "rsq_rf"))
```

```
library(ggplot2)
ggplot(data = rsq, aes(x = Noise, y = value, group = variable,
    colour = variable)) + geom_line() + geom_point() + ylab("R2")
```

Fig. 7.4 shows that:

All model performance decreases sharply with increasing noise intensity. To better anticipate model performance, it helps to understand the way variable is measured. It is something need to make
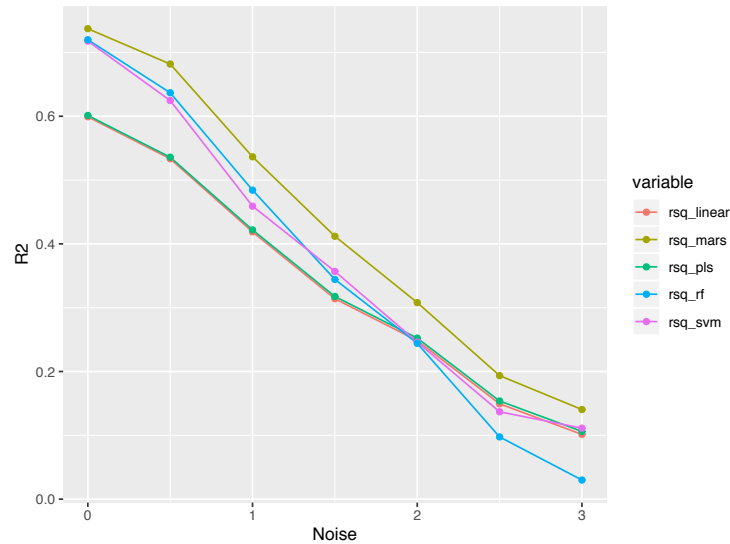
FIGURE 7.4: Test set R^2 profiles for income models when measurement system noise increases. `rsq_linear`: linear regression, `rsq_pls`: Partial Least Square, `rsq_mars`: Multiple Adaptive Regression Spline Regression, `rsq_svm`: Support Vector Machine `rsq_rf`: Random Forest

clear at the beginning of an analytical project. A data scientist should be aware of the quality of the data in the database. For data from the clients, it is an important to understand the quality of the data by communication.

More complex model is not necessarily better. The best model in this situation is MARS, not random forests or SVM. Simple linear regression and PLS perform the worst when noise is low. MARS is more complicated than the linear regression and PLS, but it is simpler and easier to explain than random forest and SVM.

When noise increases to a certain extent, the potential structure becomes vaguer, and complex random forest model starts to fail. When the systematic measurement error is significant, a more straightforward but not naive model may be a better choice. It is always a good practice to try different models, and select the sim-

plest model in the case of similar performance. Model evaluation and selection represent the career "maturity" of a data scientist.

7.1.3 Measurement Error in the Independent Variables

The traditional statistical model usually assumes that the measurement of the independent variable has no error which is not possible in practice. Considering the error in the independent variables is necessary. The impact of the error depends on the following factors: (1) the magnitude of the randomness; (2) the importance of the corresponding variable in the model, and (3) the type of model used. Use variable `online_exp` as an example. The approach is similar to the previous section. Add varying degrees of noise and see its impact on the model performance. We add the following different levels of noise (0 to 3 times the standard deviation) to `online_exp`:

$$\sigma_0 \times (0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0)$$

where σ_0 is the standard error of `online_exp`.

```
noise<-matrix(rep(NA,7*nrow(sim.dat)),nrow=nrow(sim.dat),ncol=7)
for (i in 1:nrow(sim.dat)){
noise[i,]<-rnorm(7,rep(0,7),sd(sim.dat$online_exp)*seq(0,3,by=0.5))
}
```

Likewise, we examine the effect of noise intensity on different models (R^2). The models with complexity from low to high are: ordinary linear regression, partial least square regression(PLS), multivariate adaptive regression spline (MARS), support vector machine (SVM, the Kernel function is radial basis function), and random forest. The code is similar as before so not shown here.

Comparing Fig. 7.5 and Fig. 7.4, the influence of the two types of error is very different. The error in response cannot be overcome for any model, but it is not the case for the independent variables.
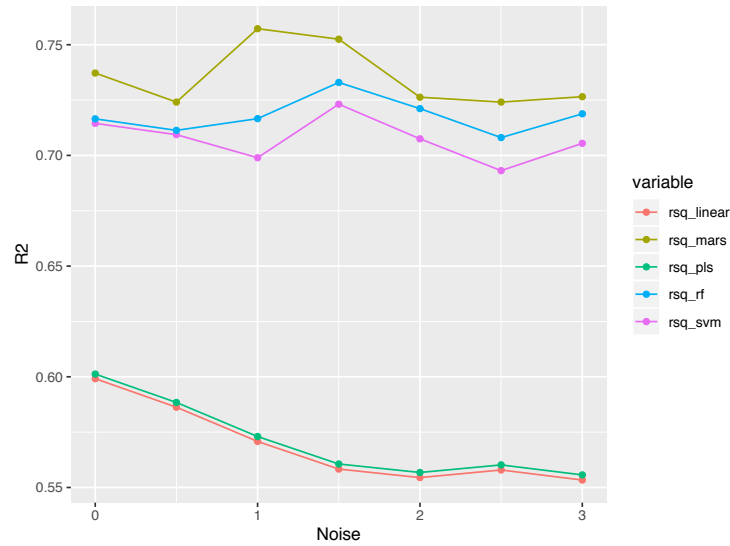
FIGURE 7.5: Test set R^2 profiles for income models when noise in `online_exp` increases. `rsq_linear` : linear regression, `rsq_pls` : Partial Least Square, `rsq_mars`: Multiple Adaptive Regression Spline Regression, `rsq_svm`: Support Vector Machine `rsq_rf`: Random Forest

Imagine an extreme case, if `online_exp` is completely random, that is, no information in it, the impact on the performance of random forest and support vector machine is marginal. Linear regression and PLS still perform similarly. With the increase of noise, the performance starts to decline faster. To a certain extent, it becomes steady. In general, if an independent variable contains error, other variables associated with it can compensate to some extent.

7.2   Data Splitting and Resampling

Those highly adaptable models can model complex relationships. However, they tend to overfit which leads to the poor prediction by learning too much from the data. It means that the model is

susceptible to the specific sample used to fit it. When future data is not exactly like the past data, the model prediction may have big mistakes. A simple model like ordinary linear regression tends instead to underfit which leads to a bad prediction by learning too little from the data. It systematically over-predicts or under-predicts the data regardless of how well future data resemble past data. Without evaluating models, the modeler will not know about the problem before the future samples. Data splitting and resampling are fundamental techniques to build sound models for prediction.

7.2.1 Data Splitting

Data splitting is to put part of the data aside as testing set (or Hold-outs, out of bag samples) and use the rest for model training. Training samples are also called in-sample. Model performance metrics evaluated using in-sample are retrodictive, not predictive.

The traditional business intelligence usually handles data description. Answer simple questions by querying and summarizing the data, such as:

- What is the monthly sales of a product in 2015?
- What is the number of visits to our site in the past month?

- What is the sales difference in 2015 for two different product designs?

There is no need to go through the tedious process of splitting the data, tuning and testing model to answer questions of this kind. Instead, people usually use as complete data as possible and then sum or average the parts of interest.

Many models have parameters which cannot be directly estimated from the data, such as λ in the lasso (penalty parameter), the number of trees in the random forest. This type of model parameter is called tuning parameter, and there is no analytical formula

available to calculate the optimized value. Tuning parameters often control the complexity of the model. A poor choice can result in over-fitting or under-fitting. A standard approach to estimate tuning parameters is through cross-validation which is a data re-sampling approach.

To get a reasonable precision of the performance based on a single test set, the size of the test set may need to be large. So a conventional approach is to use a subset of samples to fit the model and use the rest to evaluate model performance. This process will repeat multiple times to get a performance profile. In that sense, resampling is based on splitting. The general steps are:

- Define a set of candidate values for tuning parameter(s)
    - For each candidate value in the set
        * Resample data
        * Fit model
        * Predict hold-out
        * Calculate performance
- Aggregate the results
- Determine the final tuning parameter
- Refit the model with the entire data set

The above is an outline of the general procedure to tune parameters. Now let's focus on the critical part of the process: data splitting. Ideally, we should evaluate model using samples that were not used to build or fine-tune the model. So it provides an unbiased sense of model effectiveness. When the sample size is large, it is a good practice to set aside part of the samples to evaluate the final model. People use "training" data to indicate samples used to fit or fine-tune the model and "test" or "validation" data set is used to validate performance.

The first decision to make for data splitting is to decide the proportion of data in the test set. There are two factors to consider here: (1) sample size; (2) computation intensity. If the sample size is large enough which is the most common situation according to
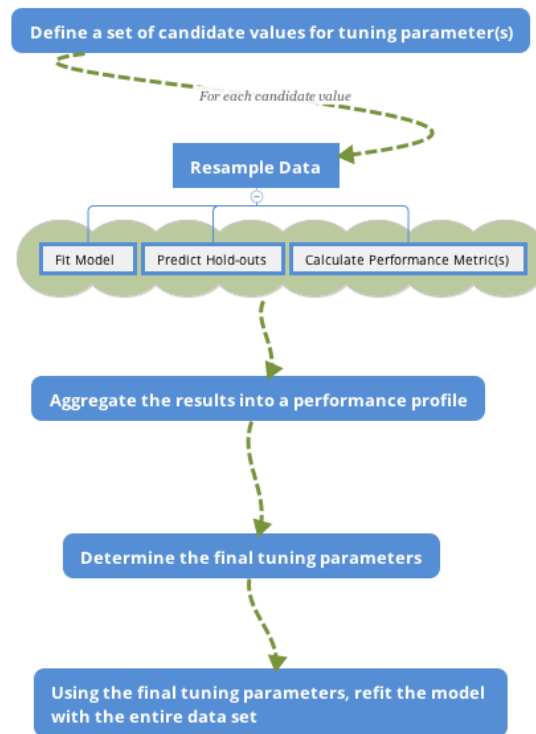
FIGURE 7.6: Parameter Tuning Process

my experience, you can try to use 20%, 30% and 40% of the data as the test set, and see which one works the best. If the model is computationally intense, then you may consider starting from a smaller sample of data to train the model hence will have a higher portion of data in the test set. Depending on how it performs, you may need to increase the training set. If the sample size is small, you can use cross-validation or bootstrap which is the topic in the next section.

The next decision is to decide which samples are in the test set. There is a desire to make the training and test sets as similar as possible. A simple way is to split data by random sampling which, however, does not control for any of the data attributes, such as the percentage of the retained customer in the data. So

it is possible that the distribution of outcomes is substantially different between the training and test sets. There are three main ways to split the data that account for the similarity of resulted data sets. We will describe the three approaches using the clothing company customer data as examples.

(1)   Split data according to the outcome variable

Assume the outcome variable is customer segment (column `seg-ment`) and we decide to use 80% as training and 20% test. The goal is to make the proportions of the categories in the two sets as similar as possible. The `createDataPartition()` function in `caret` will return a balanced splitting based on assigned variable.

```r
# load data
sim.dat <- read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/
library(caret)
# set random seed to make sure reproducibility
set.seed(3456)
trainIndex <- createDataPartition(sim.dat$segment, p = 0.8, list = FALSE,
    times = 1)
head(trainIndex)
```

```
##       Resample1
## [1,]         1
## [2,]         2
## [3,]         3
## [4,]         4
## [5,]         6
## [6,]         7
```

The `list = FALSE` in the call to `createDataPartition` is to return a data frame. The `times = 1` tells R how many times you want to split the data. Here we only do it once, but you can repeat the splitting multiple times. In that case, the function will return multiple vectors indicating the rows to training/test. You can set

times⊠2 and rerun the above code to see the result. Then we can use the returned indicator vector `trainIndex` to get training and test sets:

```
# get training set
datTrain <- sim.dat[trainIndex, ]
# get test set
datTest <- sim.dat[-trainIndex, ]
```

According to the setting, there are 800 samples in the training set and 200 in test set. Let's check the distribution of the two sets:

```
library(plyr)
ddply(datTrain, "segment", summarise, count = length(segment),
    percentage = round(length(segment)/nrow(datTrain), 2))
```

```
##         segment count percentage
## 1 Conspicuous   160        0.20
## 2       Price   200        0.25
## 3     Quality   160        0.20
## 4       Style   280        0.35
```

```
ddply(datTest, "segment", summarise, count = length(segment),
    percentage = round(length(segment)/nrow(datTest), 2))
```

```
##         segment count percentage
## 1 Conspicuous    40        0.20
## 2       Price    50        0.25
## 3     Quality    40        0.20
## 4       Style    70        0.35
```

The percentages are the same for these two sets. In practice, it is possible that the distributions are not exactly identical but should be close.

(2)   Divide data according to predictors

An alternative way is to split data based on the predictors. The goal is to get a diverse subset from a dataset so that the sample is representative. In other words, we need an algorithm to identify the n most diverse samples from a dataset with size N. However, the task is generally infeasible for non-trivial values of n and N (Willett, 2004). And hence practicable approaches to dissimilarity-based selection involve approximate methods that are sub-optimal. A major class of algorithms split the data on *maximum dissimilarity sampling*. The process starts from:

- Initialize a single sample as starting test set
- Calculate the dissimilarity between this initial sample and each remaining samples in the dataset
- Add the most dissimilar unallocated sample to the test set

To move forward, we need to define the dissimilarity between groups. Each definition results in a different version of the algorithm and hence a different subset. It is the same problem as in hierarchical clustering where you need to define a way to measure the distance between clusters. The possible approaches are to use minimum, maximum, sum of all distances, the average of all distances, etc. Unfortunately, there is not a single best choice, and you may have to try multiple methods and check the resulted sample sets. R users can implement the algorithm using `maxDissim()` function from `caret` package. The `obj` argument is to set the definition of dissimilarity. Refer to the help documentation for more details (`?maxDissim`).

Let's use two variables (`age` and `income`) from the customer data as an example to illustrate how it works in R and compare maximum dissimilarity sampling with random sampling.

```
library(lattice)
# select variables
testing <- subset(sim.dat, select = c("age", "income"))
```

Random select 5 samples as initial subset (start) , the rest will be in samplePool:

```
set.seed(5)
# select 5 random samples
startSet <- sample(1:dim(testing)[1], 5)
start <- testing[startSet, ]
# save the rest in data frame 'samplePool'
samplePool <- testing[-startSet, ]
```

Use maxDissim() to select another 5 samples from samplePool that are as different as possible with the initical set start:

```
selectId <- maxDissim(start, samplePool, obj = minDiss, n = 5)
minDissSet <- samplePool[selectId, ]
```

The obj = minDiss in the above code tells R to use minimum dissimilarity to define the distance between groups. Next, random select 5 samples from samplePool in data frame RandomSet:

```
selectId <- sample(1:dim(samplePool)[1], 5)
RandomSet <- samplePool[selectId, ]
```

Plot the resulted set to compare different sampling methods:

```r
start$group <- rep("Initial Set", nrow(start))
minDissSet$group <- rep("Maximum Dissimilarity Sampling", nrow(minDissSet))
RandomSet$group <- rep("Random Sampling", nrow(RandomSet))
xyplot(age ~ income, data = rbind(start, minDissSet, RandomSet), grid = TRUE,
    group = group, auto.key = TRUE)
```
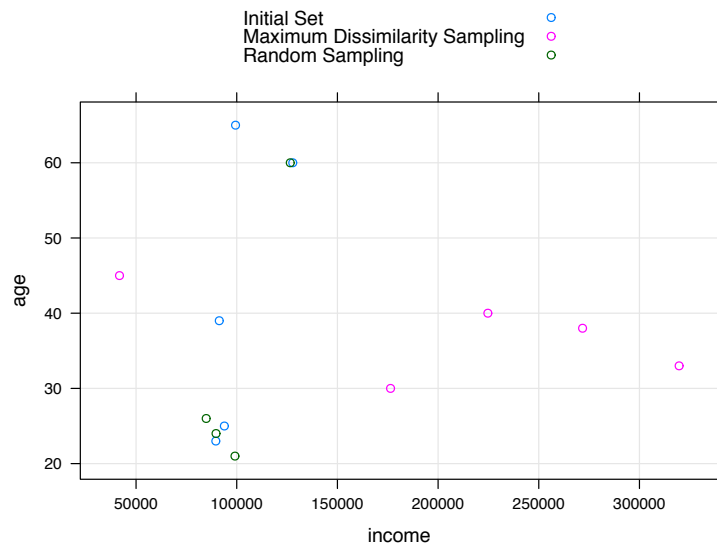

FIGURE 7.7: Compare Maximum Dissimilarity Sampling with Random Sampling

The points from maximum dissimilarity sampling are far away from the initial samples (Fig. 7.7, while the random samples are much closer to the initial ones. Why do we need a diverse subset? Because we hope the test set to be representative. If all test set samples are from respondents younger than 30, model performance on the test set has a high risk to fail to tell you how the model will perform on more general population.

- Divide data according to time

For time series data, random sampling is usually not the best way.

There is an approach to divide data according to time-series. Since time series is beyond the scope of this book, there is not much discussion here. For more detail of this method, see (Hyndman and Athanasopoulos, 2013). We will use a simulated first-order autoregressive model [AR (1)] time-series data with 100 observations to show how to implement using the function `createTimeSlices ()` in the `caret` package.

```r
# simulte AR(1) time series samples
timedata = arima.sim(list(order=c(1,0,0), ar=-.9), n=100)
# plot time series
plot(timedata, main=(expression(AR(1)~~~phi==-.9)))
```
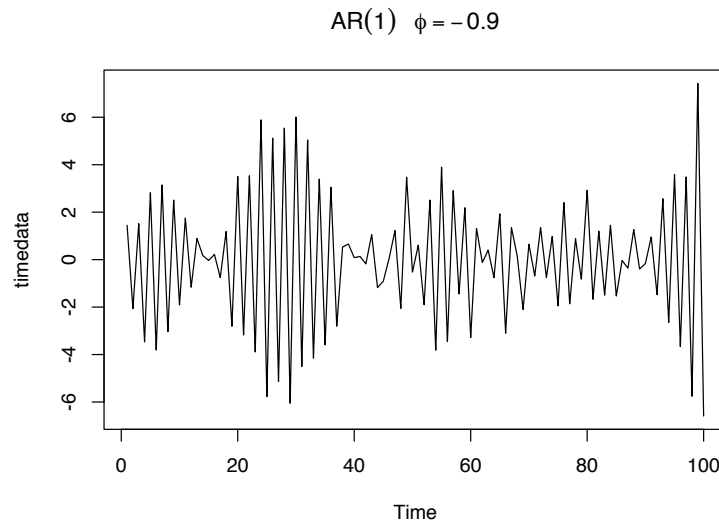

FIGURE 7.8: Divide data according to time

Fig. 7.8 shows 100 simulated time series observation. The goal is to make sure both training and test set to cover the whole period.

```r
timeSlices <- createTimeSlices(1:length(timedata),
                  initialWindow = 36, horizon = 12, fixedWindow = T)
str(timeSlices,max.level = 1)
```

```
## List of 2
##  $ train:List of 53
##  $ test :List of 53
```

There are three arguments in the above `createTimeSlices()`.

- `initialWindow`: The initial number of consecutive values in each training set sample
- `horizon`: the number of consecutive values in test set sample
- `fixedWindow`: if FALSE, all training samples start at 1

The function returns two lists, one for the training set, the other for the test set. Let's look at the first training sample:

```
# get result for the 1st training set
trainSlices <- timeSlices[[1]]
# get result for the 1st test set
testSlices <- timeSlices[[2]]
# check the index for the 1st training and test set
trainSlices[[1]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## [18] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
## [35] 35 36
```

```
testSlices[[1]]
```

```
##  [1] 37 38 39 40 41 42 43 44 45 46 47 48
```

The first training set is consist of sample 1-36 in the dataset (`initialWindow = 36`). Then sample 37-48 are in the first test set (`horizon = 12`). Type `head(trainSlices)` or `head(testSlices)` to check

the later samples. If you are not clear about the argument `fixed-Window`, try to change the setting to be `F` and check the change in `trainSlices` and `testSlices`.

Understand and implement data splitting is not difficult. But there are two things to note:

1. The randomness in the splitting process will lead to uncertainty in performance measurement.
2. When the dataset is small, it can be too expensive to leave out test set. In this situation, if collecting more data is just not possible, the best shot is to use leave-one-out cross-validation which is in the next section.

7.2.2 Resampling

You can consider resampling as repeated splitting. The basic idea is: use part of the data to fit model and then use the rest of data to calculate model performance. Repeat the process multiple times and aggregate the results. The differences in resampling techniques usually center around the ways to choose subsamples. There are two main reasons that we may need resampling:

1. Estimate tuning parameters through resampling. Some examples of models with such parameters are Support Vector Machine (SVM), models including the penalty (LASSO) and random forest.

2. For models without tuning parameter, such as ordinary linear regression and partial least square regression, the model fitting doesn't require resampling. But you can study the model stability through resampling.

We will introduce three most common resampling techniques: k-fold cross-validation, repeated training/test splitting, and bootstrap.

7.2.2.1   k-fold cross-validation

k-fold cross-validation is to partition the original sample into k equal size subsamples (folds). Use one of the k folds to validate the model and the rest $k - 1$ to train model. Then repeat the process k times with each of the k folds as the test set. Aggregate the results into a performance profile.

Denote by $\hat{f}^{-\kappa}(X)$ the fitted function, computed with the κth fold removed and x_i^κ the predictors for samples in left-out fold. The process of k-fold cross-validation is as follows:

---

    1.   Partition the original sample into k equal size folds
    2.   for $\kappa = 1...k$

- Use data other than fold κ to train the model $\hat{f}^{-\kappa}(X)$
- Apply $\hat{f}^{-\kappa}(X)$ to predict fold κ to get $\hat{f}^{-\kappa}(x_i^\kappa)$

    3.   Aggregate the results
$$\widehat{Error} = \frac{1}{N}\Sigma_{\kappa=1}^k \Sigma_{x_i^\kappa} L(y_i^\kappa, \hat{f}^{-\kappa}(x_i^\kappa))$$

It is a standard way to find the value of tuning parameter that gives you the best performance. It is also a way to study the variability of model performance.

The following figure represents a 5-fold cross-validation example.

A special case of k-fold cross-validation is Leave One Out Cross Validation (LOOCV) where $k = 1$. When sample size is small, it is desired to use as many data to train the model. Most of the functions have default setting $k = 10$. The choice is usually 5-10 in practice, but there is no standard rule. The more folds to use, the more samples are used to fit model, and then the performance estimate is closer to the theoretical performance. Meanwhile, the variance of the performance is larger since the samples to fit model
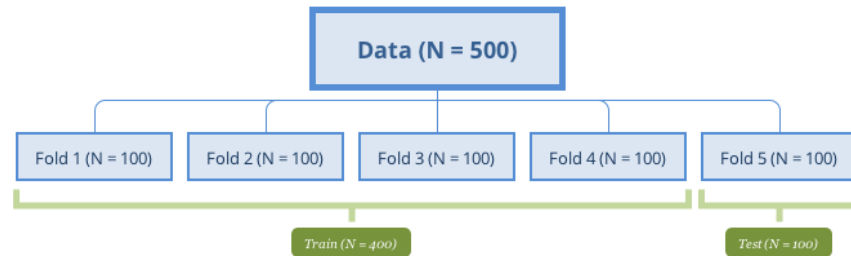
FIGURE 7.9: 5-fold cross-validation

in different iterations are more similar. However, LOOCV has high computational cost since the number of interactions is the same as the sample size and each model fit uses a subset that is nearly the same size of the training set. On the other hand, when k is small (such as 2 or 3), the computation is more efficient, but the bias will increase. When the sample size is large, the impact of k becomes marginal.

Chapter 7 of (Hastie T, 2008) presents a more in-depth and more detailed discussion about the bias-variance trade-off in k-fold cross-validation.

You can implement k-fold cross-validation using `createFolds()` in `caret`:

```
library(caret)
class<-sim.dat$segment
# creat k-folds
set.seed(1)
cv<-createFolds(class,k=10,returnTrain=T)
str(cv)
```

```
## List of 10
##  $ Fold01: int [1:900] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Fold02: int [1:900] 1 2 3 4 5 6 7 9 10 11 ...
##  $ Fold03: int [1:900] 1 2 3 4 5 6 7 8 10 11 ...
```

```
##  $ Fold04: int [1:900] 1 2 3 4 5 6 7 8 9 11 ...
##  $ Fold05: int [1:900] 1 3 4 6 7 8 9 10 11 12 ...
##  $ Fold06: int [1:900] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Fold07: int [1:900] 2 3 4 5 6 7 8 9 10 11 ...
##  $ Fold08: int [1:900] 1 2 3 4 5 8 9 10 11 12 ...
##  $ Fold09: int [1:900] 1 2 4 5 6 7 8 9 10 11 ...
##  $ Fold10: int [1:900] 1 2 3 5 6 7 8 9 10 11 ...
```

The above code creates ten folds (`k=10`) according to the customer segments (we set `class` to be the categorical variable `segment`). The function returns a list of 10 with the index of rows in training set.

7.2.2.2   Repeated Training/Test Splits

In fact, this method is nothing but repeating the training/test set division on the original data. Fit the model with the training set, and evaluate the model with the test set. Unlike k-fold cross-validation, the test set generated by this procedure may have duplicate samples. A sample usually shows up in more than one test sets. There is no standard rule for split ratio and number of repetitions. The most common choice in practice is to use 75% to 80% of the total sample for training. The remaining samples are for validation. The more sample in the training set, the less biased the model performance estimate is. Increasing the repetitions can reduce the uncertainty in the performance estimates. Of course, it is at the cost of computational time when the model is complex. The number of repetitions is also related to the sample size of the test set. If the size is small, the performance estimate is more volatile. In this case, the number of repetitions needs to be higher to deal with the uncertainty of the evaluation results.

We can use the same function (`createDataPartition ()`) as before. If you look back, you will see `times = 1`. The only thing to change is to set it to the number of repetitions.

```
trainIndex <- createDataPartition(sim.dat$segment, p = .8, list = FALSE, times = 5)
dplyr::glimpse(trainIndex)
```

```
##  int [1:800, 1:5] 1 3 4 5 6 7 8 9 10 11 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:5] "Resample1" "Resample2" "Resample3" "Resample4" ...
```

Once know how to split the data, the repetition comes naturally.

7.2.2.3 Bootstrap Methods

Bootstrap is a powerful statistical tool (a little magic too). It can be used to analyze the uncertainty of parameter estimates (Efron and Tibshirani, 1986) quantitatively. For example, estimate the standard deviation of linear regression coefficients. The power of this method is that the concept is so simple that it can be easily applied to any model as long as the computation allows. However, you can hardly obtain the standard deviation for some models by using the traditional statistical inference.

Since it is with replacement, a sample can be selected multiple times, and the bootstrap sample size is the same as the original data. So for every bootstrap set, there are some left-out samples, which is also called "out-of-bag samples." The out-of-bag sample is used to evaluate the model. Efron points out that under normal circumstances (Efron, 1983), bootstrap estimates the error rate of the model with more certainty.The probability of an observation i in bootstrap sample B is:

$$
\begin{aligned}
Pri \in B &= 1 - \left(1 - \frac{1}{N}\right)^N \\
&\approx 1 - e^{-1} \\
&= 0.632
\end{aligned}
$$

On average, 63.2% of the observations appeared at least once in a bootstrap sample, so the estimation bias is similar to 2-fold cross-validation. As mentioned earlier, the smaller the number of folds, the larger the bias. Increasing the sample size will ease the problem. In general, bootstrap has larger bias and smaller uncertainty than cross-validation. Efron came up the following ".632 estimator" to alleviate this bias:

$$(0.632 \times original\ bootstrap\ estimate) + (0.368 \times apparent\ error\ rate)$$

The apparent error rate is the error rate when the data is used twice, both to fit the model and to check its accuracy and it is apparently over-optimistic. The modified bootstrap estimate reduces the bias but can be unstable with small samples size. This estimate can also be unduly optimistic when the model severely overfits since the apparent error rate will be close to zero. Efron and Tibshirani (Efron and Tibshirani, 1997) discuss another technique, called the "632+ method," for adjusting the bootstrap estimates.

8

Measuring Performance

In order to compare different models, we need a way to measure model performance. There are different metrics to use. Sometimes, it is better to look at models through multiple lens. In this chapter, we will introduce some of the most common measurement metrics.

8.1 Regression Model Performance

8.2 Classification Model Performance

9

Regression Models

9.1   Ordinary Least Squares

9.2   Multivariate Adaptive Regression Splines

9.3   Generalized Linear Model

9.4   PCR and PLS

10

Regularization Methods

The regularization method is also known as the shrinkage method. It is a technique that constrains or regularizes the coefficient estimates. By imposing a penalty on the size of coefficients, it shrinks the coefficient estimates towards zero. It also intrinsically conduct feature selection and is naturally resistant to non-informative predictors. It may not be obvious why this technique improves model performance, but it turns out to be a very effective modeling technique. In this chapter, we will introduce two best-known regularization methods: ridge regression and lasso. The elastic net is a combination of ridge and lasso, or it is a general representation of the two.

We talked about the variance bias trade-off in section 7.1.1. The variance of a learning model is the amount by which $\hat{f}$ would change if we estimated it using a different training data set. In general, model variance increases as flexibility increases. The regularization technique decreases the model flexibility by shrinking the coefficient and hence significantly reduce the model variance.

10.1   Ridge Regression

Recall that the least square estimates minimize RSS:

$$RSS = \Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2$$

Ridge regression (Hoerl and Kennard, 1970) is similar but it finds $\hat{\beta}^R$ that optimizes a slightly different function:

$$\Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\Sigma_{j=1}^{p}\beta_j^2 = RSS + \lambda\Sigma_{j=1}^{p}\beta_j^2 \quad (10.1)$$

where $\lambda > 0$ is a tuning parameter. As with the least squares, ridge regression considers minimizing RSS. However, it adds a shrinkage penalty $\lambda\Sigma_{j=1}^{p}\beta_j^2$ that takes account of the number of parameters in the model. When $\lambda = 0$, it is identical to least squares. As λ gets larger, the coefficients start shrinking towards 0. When $\lambda \to \infty$, the rest of the coefficients $\beta_1, ..., \beta_p$ are close to 0. Here, the penalty is not applied to β_0. The tuning parameter λ is used to adjust the impact of the two parts in equation (10.1). Every value of λ corresponds to a set of parameter estimates.

There are many R packages for ridge regression, such as lm.ridge() function from MASS, function enet() from, elasticnet. If you know the value of λ, you can use either of the function to fit ridge regression. A more convenient way is to use train() function from caret. Let's use the 10 survey questions to predict the total purchase amount (sum of online and store purchase).

```r
# install packages from CRAN
p_needed <- c('caret', 'elasticnet', 'glmnet', 'devtools')
packages <- rownames(installed.packages())
p_to_install <- p_needed[!(p_needed %in% packages)]
if (length(p_to_install) > 0) {
    install.packages(p_to_install)
}


lapply(p_needed, require, character.only = TRUE)
```

```r
dat <- read.csv("http://bit.ly/2P5gTw4")
# data cleaning: delete wrong observations since expense can't be negative
dat <- subset(dat, store_exp > 0 & online_exp > 0)
# get predictors
trainx <- dat[ , grep("Q", names(dat))]
```

```r
# get response
trainy <- dat$store_exp + dat$online_exp
```

Use `train()` function to tune parameter. Since ridge regression adds the penalty parameter λ in front of the sum of squares of the parameters, the scale of the parameters matters. So here it is better to center and scale the predictors. This preprocessing is recommended for all techniques that put penalty to parameter estimates. In this example, the 10 survey questions are already with the same scale so data preprocessing doesn't make too much different. It is a good idea to set the preprocessing as a standard.

```r
# set cross validation
ctrl <- trainControl(method = "cv", number = 10)
# set the parameter range
ridgeGrid <- data.frame(.lambda = seq(0, .1, length = 20))
set.seed(100)
ridgeRegTune <- train(trainx, trainy,
                      method = "ridge",
                      tuneGrid = ridgeGrid,
                      trControl = ctrl,
                      ## center and scale predictors
                      preProc = c("center", "scale"))
ridgeRegTune
```

```
## Ridge Regression
##
## 999 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 899, 899, 899, 899, 899, 900, ...
## Resampling results across tuning parameters:
```

```
##
##    lambda    RMSE   Rsquared   MAE
##    0.000000  1744   0.7952     754.0
##    0.005263  1744   0.7954     754.9
##    0.010526  1744   0.7955     755.9
##    0.015789  1744   0.7955     757.3
##    0.021053  1745   0.7956     758.8
##    0.026316  1746   0.7956     760.6
##    0.031579  1747   0.7956     762.4
##    0.036842  1748   0.7956     764.3
##    0.042105  1750   0.7956     766.4
##    0.047368  1751   0.7956     768.5
##    0.052632  1753   0.7956     770.6
##    0.057895  1755   0.7956     772.7
##    0.063158  1757   0.7956     774.9
##    0.068421  1759   0.7956     777.2
##    0.073684  1762   0.7956     779.6
##    0.078947  1764   0.7955     782.1
##    0.084211  1767   0.7955     784.8
##    0.089474  1769   0.7955     787.6
##    0.094737  1772   0.7955     790.4
##    0.100000  1775   0.7954     793.3
##
## RMSE was used to select the optimal model using
##  the smallest value.
## The final value used for the model was lambda
##  = 0.005263.
```

The results show that the best value of λ is 0.005 and the RMSE
and R^2 are 1744 and 0.7954 correspondingly. You can see from the
figure 10.1, as the λ increase, the RMSE first slightly decreases and
then increases.

```
plot(ridgeRegTune)
```

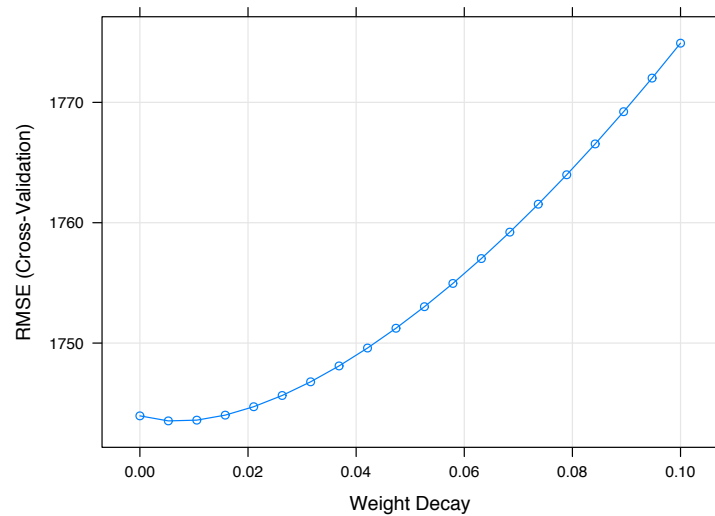Once you have the tuning parameter value, there are different func-

FIGURE 10.1: Test mean squared error for the ridge regression

tions to fit a ridge regression. Let's look at how to use `enet()` in `elasticnet` package.

```
ridgefit = enet(x = as.matrix(trainx), y = trainy, lambda = 0.01,
                # center and scale predictors
                normalize = TRUE)
```

Note here `ridgefit` only assigns the value of the tuning parameter for ridge regression. Since the elastic net model include both ridge and lasso penalty, we need to use `predict()` function to get the model fit. You can get the fitted results by setting `s = 1` and `mode = "fraction"`. Here `s = 1` means we only use the ridge parameter. We will come back to this when we get to lasso regression.

```
ridgePred <- predict(ridgefit, newx = as.matrix(trainx),
                     s = 1, mode = "fraction", type = "fit")
```

By setting `type = "fit"`, the above returns a list object. The `fit` item has the predictions:

```r
names(ridgePred)
```

```
## [1] "s"        "fraction" "mode"     "fit"
```

```r
head(ridgePred$fit)
```

```
##      1      2      3      4      5      6
## 1290.5  224.2  591.4 1220.6  853.4  908.2
```

If you want to check the estimated coefficients, you can set
`type="coefficients"`:

```r
ridgeCoef<-predict(ridgefit,newx = as.matrix(trainx),
                   s=1, mode="fraction", type="coefficients")
```

It also returns a list and the estimates are in the `coefficients` item:

```r
# didn't show the results
RidgeCoef = ridgeCoef$coefficients
```

Comparing to the least square regression, ridge regression performs
better because of the bias-variance-trade-off we mentioned in sec-
tion 7.1.1. As the penalty parameter λ increases, the flexibility
of the ridge regression decreases. It decreases the variance of the
model but increases the bias at the same time.

10.2 LASSO

Even though the ridge regression shrinks the parameter estimates towards 0, it won't shink any estimates to be exactly 0 which means it includes all predictors in the final model. So it can't select variables. It may not be a problem for prediction but it is a huge disadvantage if you want to interpret the model especially when the number of variables is large. A popular alternative to the ridge penalty is the **Least Absolute Shrinkage and Selection Operator** (LASSO) (R, 1996).

Similar to ridge regression, lasso adds a penalty. The lasso coefficients $\hat{\beta}_\lambda^L$ minimize the following:

$$\Sigma_{i=1}^n (y_i - \beta_0 - \Sigma_{j=1}^p \beta_j x_{ij})^2 + \lambda \Sigma_{j=1}^p |\beta_j| = RSS + \lambda \Sigma_{j=1}^p |\beta_j| \quad (10.2)$$

The only difference between lasso and ridge is the penalty. In statistical parlance, ridge uses L_2 penalty (β_j^2) and lasso uses L_1 penalty $(|\beta_j|)$. L_1 penalty can shrink the estimates to 0 when λ is big enough. So lasso can be used as a feature selection tool. It is a huge advantage because it leads to a more explainable model.

Similar to other models with tuning parameters, lasso regression requires cross-validation to tune the parameter. You can use `train()` in a similar way as we showed in the ridge regression section. To tune parameter, we need to set cross-validation and parameter range. Also, it is advised to standardize the predictors:

```
ctrl <- trainControl(method = "cv", number = 10)
lassoGrid <- data.frame(fraction = seq(.8, 1, length = 20))
set.seed(100)
lassoTune <- train(trainx, trainy,
                    ## set the method to be lasso
                    method = "lars",
                    tuneGrid = lassoGrid,
```

```
                        trControl = ctrl,
                        ## standardize the predictors
                        preProc = c("center", "scale"))
lassoTune
```

```
## Least Angle Regression
##
## 999 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 899, 899, 899, 899, 899, 900, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE  Rsquared  MAE
##   0.8000    1763  0.7921    787.5
##   0.8105    1760  0.7924    784.1
##   0.8211    1758  0.7927    780.8
##   0.8316    1756  0.7930    777.7
##   0.8421    1754  0.7933    774.6
##   0.8526    1753  0.7936    771.8
##   0.8632    1751  0.7939    769.1
##   0.8737    1749  0.7942    766.6
##   0.8842    1748  0.7944    764.3
##   0.8947    1746  0.7947    762.2
##   0.9053    1745  0.7949    760.1
##   0.9158    1744  0.7951    758.3
##   0.9263    1743  0.7952    756.7
##   0.9368    1743  0.7953    755.5
##   0.9474    1742  0.7954    754.5
##   0.9579    1742  0.7954    754.0
##   0.9684    1742  0.7954    753.6
##   0.9789    1743  0.7953    753.4
##   0.9895    1743  0.7953    753.5
##   1.0000    1744  0.7952    754.0
```

```
##
## RMSE was used to select the optimal model using
##   the smallest value.
## The final value used for the model was fraction
##   = 0.9579.
```

The results show that the best value of the tuning parameter (fraction from the output) is 0.957 and the RMSE and R^2 are 1742 and 0.7954 correspondingly. The performance is nearly the same with ridge regression. You can see from the figure 10.2, as the λ increase, the RMSE first decreases and then increases.
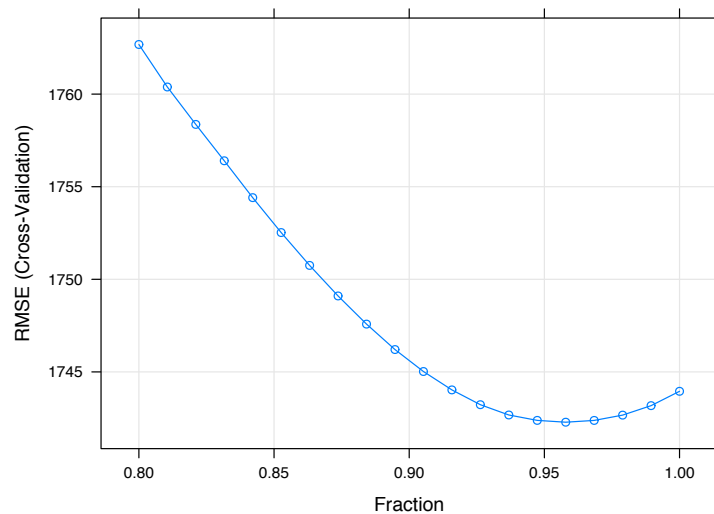
```
plot(lassoTune)
```


FIGURE 10.2: Test mean squared error for the lasso regression

Once you select a value for tuning parameter, there are different functions to fit lasso regression, such as lars() in lars, enet() in elasticnet, glmnet() in glmnet. They all have very similar syntax.

Here we continue using enet(). The syntax is similar to ridge regression. The only difference is that you need to set lambda = 0

because the argument `lambda` here is to control the ridge penalty.
When it is 0, the function will return the lasso model object.

```r
lassoModel<- enet(x = as.matrix(trainx), y = trainy, lambda = 0, normalize = TRUE)
```

Set the fraction value to be 0.957 (the value we got above):

```r
lassoFit <- predict(lassoModel, newx = as.matrix(trainx), s = 0.957, mode = "fraction", type =
```

Again by setting `type = "fit"`, the above returns a list object. The
`fit` item has the predictions:

```r
head(lassoFit$fit)
```

```
##      1      2      3      4      5      6
## 1357.3  300.5  690.2 1228.2  838.4 1010.1
```

You need to set `type = "coefficients"` to get parameter estimates:

```r
lassoCoef <- predict(lassoModel, newx = as.matrix(trainx), s = 0.95, mode = "fraction", type =
```

It also returns a list and the estimates are in the `coefficients` item:

```r
# didn't show the results
LassoCoef = lassoCoef$coefficients
```

Many researchers applied lasso to other learning methods, such as
linear discriminant analysis (Line Clemmensen and Ersbøll, 2011),
partial least squares regression(Chun and Keleş, 2010). However,

since the L_1 norm is not differentiable, optimization for lasso regression is more complicated. People come up with different algorithms to solve the computation problem. The biggest breakthrough is Least Angle Regression [LARS] from Bradley Efron etc. This algorithm works well for lasso regression especially when the dimension is high.

10.3   Variable selection property of the lasso

You may ask why lasso (L_1 penalty) has the feature selection property but not the ridge (L_2 penalty). To answer that question, let's look at the alternative representations of the optimization problem for lasso and ridge. For lasso regression, it is identical to optimize the following two functions:

$$\Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\Sigma_{j=1}^{p}|\beta_j| = RSS + \lambda\Sigma_{j=1}^{p}|\beta_j| \quad (10.3)$$

$$\min_{\beta}\left\{\Sigma_{i=1}^{n}\left(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij}\right)^2\right\}, \ \Sigma_{j=1}^{p}|\beta_j| \leq s \quad (10.4)$$

For any value of tuning parameter λ, there exists a s such that the coefficient estimates optimize equation (10.3) also optimize equation (10.4). Similarly, for ridge regression, the two representations are identical:

$$\Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\Sigma_{j=1}^{p}\beta_j^2 = RSS + \lambda\Sigma_{j=1}^{p}\beta_j^2 \quad (10.5)$$

$$\min_{\beta}\left\{\Sigma_{i=1}^{n}\left(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij}\right)^2\right\}, \ \Sigma_{j=1}^{p}\beta_j^2 \leq s \quad (10.6)$$

When p 2, lasso estimates ($\hat{\beta}_{lasso}$ in figure 10.3) have the smallest RSS among all points that satisfy $|\beta * 1| + |\beta_2| \leq s$ (i.e. within the diamond in figure 10.3). Ridge estimates have the smallest RSS

among all points that satisfy $\beta_1^2 + \beta_2^2 \leq s$ (i.e. within the circle in figure 10.3). As s increases, the diamond and circle regions expand and get less restrictive. If s is large enough, the restrictive region will cover the least squares estimate ($\hat{\beta}_{LSE}$). Then, equations (10.4) and (10.6) will simply yield the least squares estimate. In contrast, if s is small, the grey region in figure 10.3 will be small and hence restrict the magnitude of β. With the alternative formulations, we can answer the question of feature selection property.
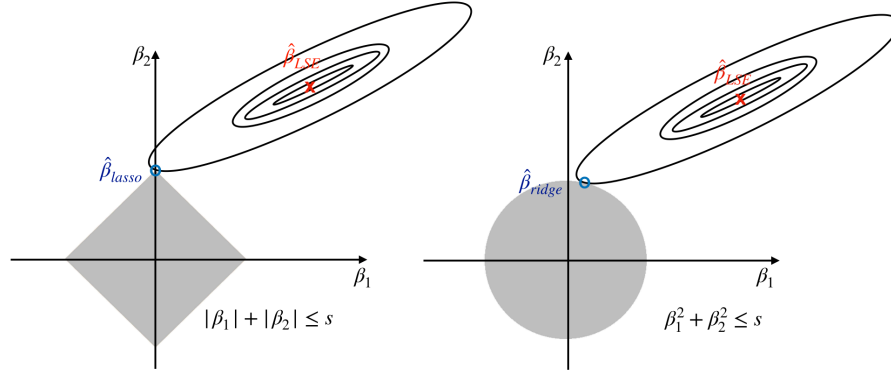

FIGURE 10.3: Contours of the RSS and constrain functions for the lasso (left) and ridge regression (right)

Figure 10.3 illustrates the situations for lasso (left) and ridge (right). The least square estimate is marked as $\hat{\beta}$. The restrictive regions are in grey, the diamond region is for the lasso, the circle region is for the ridge. The least square estimates lie outside the grey region, so they are different from the lasso and ridge estimates. The ellipses centered around $\hat{\beta}$ represent contours of RSS. All the points on a given ellipse share an RSS. As the ellipses expand, the RSS increases. The lasso and ridge estimates are the first points at which an ellipse contracts the grey region. Since ridge regression has a circular restrictive region that doesn't have a sharp point, the intersecting point can't drop on the axis. But it is possible for lasso since it has corners at each of the axes. When the intersecting point is on an axis, one of the parameter estimates is 0. If p > 2, the restrictive regions become sphere or

hypersphere. In that case, when the intersecting point drops on an axis, multiple coefficient estimates can equal 0 simultaneously.

10.4 Elastic Net

Elastic Net is a generalization of lasso and ridge regression(Zou and Hastie, 2005). It combines the two penalties. The estimates of coefficients optimize the following function:

$$\Sigma_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda_1 \Sigma_{j=1}^p \beta_j^2 + \lambda_2 \Sigma_{j=1}^p |\beta_j| \qquad (10.7)$$

Ridge penalty shrinks the coefficients of correlated predictors towards each other while the lasso tends to pick one and discard the others. So lasso estimates have a higher variance. However, ridge regression doesn't have a variable selection property. The advantage of the elastic net is that it keeps the feature selection quality from the lasso penalty as well as the effectiveness of the ridge penalty. (Zou and Hastie, 2005) suggest that it deals with highly correlated variables more effectively.

We can still use train() function to tune the parameters in the elastic net. As before, set the cross-validation and parameter range. Standardize the predictors:

```
enetGrid <- expand.grid(.lambda = seq(0,0.2,length=20),
                        .fraction = seq(.8, 1, length = 20))
set.seed(100)
enetTune <- train(trainx, trainy,
                  method = "enet",
                  tuneGrid = enetGrid,
                  trControl = ctrl,
                  preProc = c("center", "scale"))
enetTune
```

```
Elasticnet


999 samples
 10 predictor


Pre-processing: centered (10), scaled (10)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 899, 899, 899, 899, 899, 900, ...
Resampling results across tuning parameters:

  lambda    fraction   RMSE   Rsquared   MAE
  0.00000   0.8000     1763   0.7921     787.5
  0.00000   0.8105     1760   0.7924     784.1
  .
  .
  .
  0.09474   0.9158     1760   0.7945     782.5
  0.09474   0.9263     1761   0.7947     782.5
  0.09474   0.9368     1761   0.7949     782.7
  0.09474   0.9474     1763   0.7950     783.3
  0.09474   0.9579     1764   0.7951     784.3
  0.09474   0.9684     1766   0.7953     785.7
  0.09474   0.9789     1768   0.7954     787.1
  0.09474   0.9895     1770   0.7954     788.8
  0.09474   1.0000     1772   0.7955     790.4
 [ reached getOption("max.print") -- omitted 200 rows ]


RMSE was used to select the optimal model using the smallest value.
The final values used for the model were fraction = 0.9579 and lambda = 0.
```

The results show that the best values of the tuning parameters are fraction is 0.9579 and lambda is 0 It also indicates that the final model is lasso only (the ridge penalty parameter lambda is 0). The RMSE and R^2 are 1742.2843 and 0.7954 correspondingly.

10.5 Penalized Generalized Linear Model

Adding penalties is a general technique that can be applied to many methods other than linear regression. In this section, we will introduce the penalized generalized linear model. It is to fit the generalized linear model by minimizing a penalized maximum likelihood. The penalty can be L_1, L_2 or a combination of the two. The estimates of coefficients minimize the following:

$$\min_{\beta_0,\ } \frac{1}{N} \Sigma_{i=1}^{N} w_i l(y_i, \beta_0 + {}^{\mathbf{T}}\mathbf{x_i}) + \lambda[(1-\alpha) \|\ \|_2^2 /2 + \alpha \|\ \|_1]$$

where

$$l(y_i, \beta_0 + {}^{\mathbf{T}}\mathbf{x_i}) = -log[\mathcal{L}(y_i, \beta_0 + {}^{\mathbf{T}}\mathbf{x_i})]$$

It is the negative logarithm of the likelihood, $\mathcal{L}(y_i, \beta_0 + {}^{\mathbf{T}}\mathbf{x_i})$. Maximize likelihood is to minimize $l(y_i, \beta_0 + {}^{\mathbf{T}}\mathbf{x_i})$.

Parameter α decides the penalty, i.e, between L_2 ($\alpha = 0$) and L_1 ($\alpha = 1$). λ controls the weight of the whole penalty item. The higher λ is, the more weight the penalty carries comparing to likelihood. As discussed above, the ridge penalty shrinks the coefficients towards 0 but can't be exactly 0. The lasso penalty can set 0 estimates so it has the property of feature selection. The elastic net combines both. Here we have two tuning parameters, α and λ.

10.5.1 Introduction to `glmnet` package

`glmnet` is a package that fits a penalized generalized linear model using *cyclical coordinate descent*. It successively optimizes the objective function over each parameter with others fixed, and cycles repeatedly until convergence. Since the linear model is a special case of the generalized linear model, `glmnet` can also fit a penalized linear model. Other than that, it can also fit penalized logistic regression, multinomial, Poisson, and Cox regression models.

The default family option in the function `glmnet()` is `gaussian`. It is the linear regression we discussed so far in this chapter. But the parameterization is a little different in the generalized linear model framework (we have α and λ). Let's start from our previous example, using the same training data but `glmnet()` to fit model:

```r
library(glmnet)
dat <- read.csv("http://bit.ly/2P5gTw4")
# data cleaning: delete wrong observations since expense can't be negative
dat <- subset(dat, store_exp > 0 & online_exp > 0)
# get predictors
trainx <- dat[, grep("Q", names(dat))]
# get response
trainy <- dat$store_exp + dat$online_exp
glmfit = glmnet(as.matrix(trainx), trainy)
```

The object `glmfit` returned by `glmnet()` has the information of the fitted model for the later operations. An easy way to extract the components is through various functions on `glmfit`, such as `plot()`, `print()`, `coef()` and `predict()` . For example, the following code visualizes the path of coefficients as penalty increases:

```r
plot(glmfit, label = T)
```
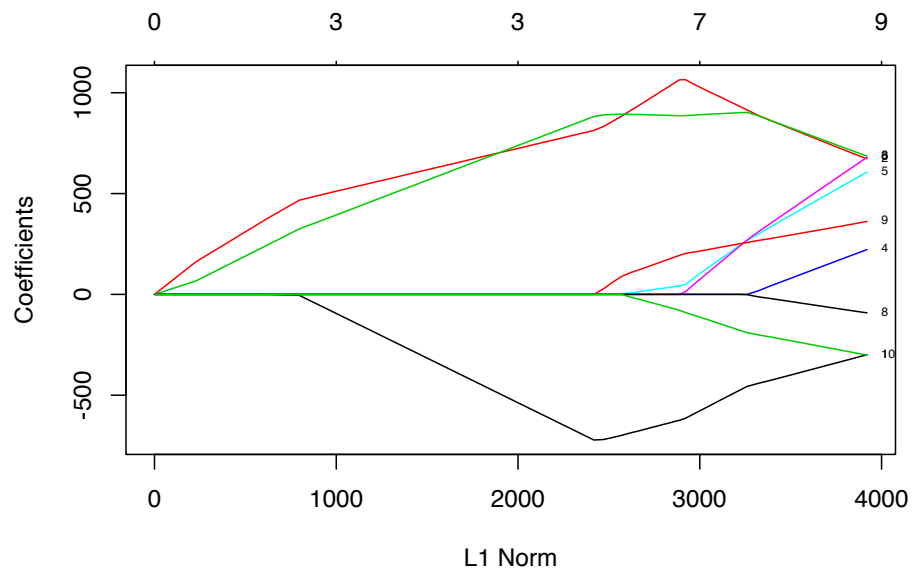
Each curve in the plot represents one predictor. The default setting is $\alpha = 1$ which means there is only lasso penalty. From left to right, L_I norm is increasing which means λ is decreasing. The bottom x-axis is L_1 norm (i.e. $\| \quad \|_1$). The upper x-axis is the effective degrees of freedom (df) for the lasso. You can check the detail for every step by:

```
print(glmfit)
```

```
Call:  glmnet(x = as.matrix(trainx), y = trainy)

    Df  %Dev Lambda
1    0 0.000   3040
2    2 0.104   2770
3    2 0.192   2530
4    2 0.265   2300
5    3 0.326   2100
6    3 0.389   1910
7    3 0.442   1740
8    3 0.485   1590
9    3 0.521   1450
```

...

The first column `Df` is the degree of freedom (i.e. the number of non-zero coefficients), `%Dev` is the percentage of deviance explained and `Lambda` is the value of tuning parameter λ. By default, the function will try 100 different values of λ. However, if as λ changes, the `%Dev` doesn't change sufficiently, the algorithm will stop before it goes through all the values of λ. We didn't show the full output above. But it only uses 68 different values of λ. You can also set the value of λ using `s=` :

```
coef(glmfit, s = 1200)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     1
## (Intercept) 2255.2
## Q1           -390.9
## Q2            653.6
## Q3            624.4
## Q4                .
## Q5                .
## Q6                .
## Q7                .
## Q8                .
## Q9                .
## Q10               .
```

When $\lambda = 1200$, there are three coefficients with non-zero estimates(`Q1`, `Q2` and `Q3`). You can apply models with different values of tuning parameter to new data using `predict()`:

```
newdat = matrix(sample(1:9, 30, replace = T), nrow = 3)
predict(glmfit, newdat, s = c(1741, 2000))
```

```
##             1    2
## [1,] 6004 5968
## [2,] 7101 6674
## [3,] 9158 8411
```

Each column corresponds to a value of λ. To tune the value of λ, we can easily use `cv.glmnet()` function to do cross-validation.`cv.glmnet()` returns the cross-validation results as a list object. We store the object in `cvfit` and use it for further operations.

```
cvfit = cv.glmnet(as.matrix(trainx), trainy)
```

We can plot the object using `plot()`. The red dotted line is the cross-validation curve. Each red point is the cross-validation mean squared error for a value of λ. The grey bars around the red points indicate the upper and lower standard deviation. The two gray dotted vertical lines represent the two selected values of λ, one gives the minimum mean cross-validated error (`lambda.min`), the other gives the error that is within one standard error of the minimum (`lambda.1se`).

```
plot(cvfit)
```

9   9   9   9   9   9   9   7   7   6   6   4   3   3   3   3   2
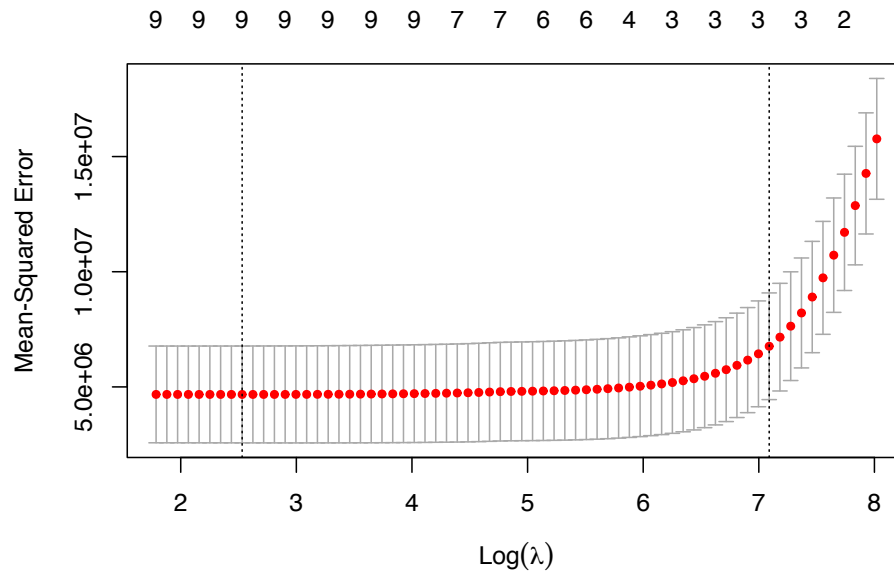
You can check the two selected λ values by:

```r
# lambda with minimum mean cross-validated error
cvfit$lambda.min
```

```
## [1] 12.57
```

```r
# lambda with one standard error of the minimum
cvfit$lambda.1se
```

```
## [1] 1200
```

You can look at the coefficient estimates for different λ by:

```r
# coefficient estimates for model with the error
# that is within one standard error of the minimum
coef(cvfit, s = "lambda.1se")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     1
## (Intercept) 2255.3
## Q1          -391.1
## Q2           653.7
## Q3           624.5
## Q4              .
## Q5              .
## Q6              .
## Q7              .
## Q8              .
## Q9              .
## Q10             .
```

10.5.2 Penalized logistic regression

10.5.2.1 Multivariate logistic regression model

Logistic regression is a traditional statistical method for a two-category classification problem. It is simple yet useful. Here we use the swine disease breakout data as an example to illustrate the learning method and code implementation. Refer to section 3.3 for more details about the dataset. The goal is to predict if a farm will have a swine disease outbreak (i.e build a risk scoring system).

Consider risk scoring system construction using a sample of n observations, with information collected for G categorical predictors and one binary response variable for each observation. The predictors are 120 survey questions (i.e. G=120). There were three possible answers for each question (A, B and C). So each predictor is encoded to two dummy variables (we consider C as the baseline.). Let $\mathbf{x_{i,g}}$ be the vector of dummy variables associated with the gth categorical predictor for the ith observation, where $i = 1, \cdots, n$, $g = 1, \cdots, G$. For example, if the first farm chooses B for question 2, then the corresponding observation is $\mathbf{x_{12}} = (0, 1)^T$. Each question has a degree of freedom of 2.

We denote the degrees of freedom of the gth predictor by df_g,

which is also the length of vector $\mathbf{x_{i,g}}$. Let y_i ($= 1$, diseased; or 0, not diseased) be the binary response for the ith observation. Denote the probability of disease for ith subject by θ_i, the model can be formulated as:

$$y_i \sim Bounoulli(\theta_i)$$

$$log\left(\frac{\theta_i}{1 - \theta_i}\right) = \eta\left(x_i\right) = \beta_0 + \sum_{g=1}^{G} \mathbf{x_{i,g}}^{T}\,\mathbf{g}$$

where β_0 is the intercept and $\mathbf{g}$ is the parameter vector corresponding to the gth predictor. As we mentioned, here $\mathbf{g}$ has length 2.

Traditional estimation of logistic parameters $=$ $(\beta_0^T,\, \mathbf{1}^T,\, \mathbf{2}^T, ...,\, \mathbf{G}^T)^T$ is done through maximizing the log-likelihood

$$
\begin{aligned}
l(\ ) &= log[\prod_{i=1}^{n} \theta_i^{y_i}(1 - \theta_i)^{1-y_i}] \\
&= \sum_{i=1}^{n}\{y_i log(\theta_i) + (1 - y_i)log(1 - \theta_i)\} \\
&= \sum_{i=1}^{n}\{\ y_i\eta\left(\mathbf{x_i}\right) - log[1 + exp(\eta\left(\mathbf{x_i}\right))]\ \}
\end{aligned}
$$

For logistic regression analysis with a large number of explanatory variables, complete- or quasi-complete-separation may result which makes the maximum likelihood estimation unstable (Wedderburn, 1976), (A and J, 1984). For example:

```r
library(MASS)
dat <- read.csv("http://bit.ly/2KXb1Qi")
fit <- glm(y~., dat, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or
## 1 occurred
```

There is an error saying "`algorithm did not converge`". It is because there is complete separation. It happens when there are a large number of explanatory variables which makes the estimation of the coefficients unstable. To stabilize the estimation of parameter coefficients, one popular approach is the lasso algorithm with L_1 norm penalty proposed by Tibshirani(R, 1996). Because the lasso algorithm can estimate some variable coefficients to be 0, it can also be used as a variable selection tool.

10.5.2.2 Penalized logistic regression

Penalized logistic regression adds penalty to the likelihood function:

$$\min_{\in \mathbb{R}^{p+1}} -\sum_{i=1}^{n} \{\, y_i \eta\,(\mathbf{x_i}) - log[1 + exp(\eta\,(\mathbf{x_i}))] \,\} + \lambda[(1-\alpha) \parallel \quad \parallel_2^2 /2] + \alpha \parallel \quad \parallel_1]$$

```
dat <- read.csv("http://bit.ly/2KXb1Qi")
trainx = dplyr::select(dat, -y)
trainy = dat$y
fit <- glmnet(as.matrix(trainx), trainy, family = "binomial")
```

The error message is gone when we use penalized regression. We can visualize the shrinking path of coefficients as penalty increases. The use of `predict()` function is a little different. For the generalized linear model, you can return different results by setting the `type` argument. The choices are:

- `link`: return the link function value
- `response`: return the probability
- `class`: return the category $(0/1)$
- `coefficients`: return the coefficient estimates

- `nonzero`: return an indicator for non-zero estimates (i.e. which variables are selected)

The default setting is to predict the probability of the second level of the response variable. For example, the second level of the response variable for `trainy` here is:

```
levels(as.factor(trainy))
```

```
## [1] "0" "1"
```

The first column of the above output is the predicted link function value when $\lambda = 0.02833$. The second column of the output is the predicted link function when $\lambda = 0.0311$. Similarly, you can change the setting for `type` to produce different outputs. You can use the `cv.glmnet()` function to tune parameters. The parameter setting is nearly the same as before, the only difference is the setting of `type.measure`. Since the response is categorical, not continuous, we have different performance measurements. The most common settings of `type.measure` for classification are:
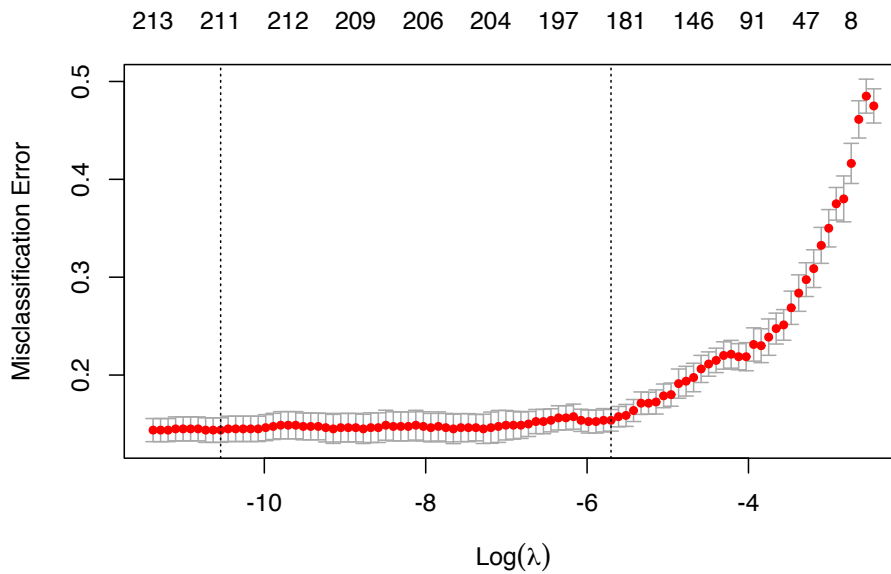
- `class`: error rate
- `auc`: it is the area under the ROC for the dichotomous problem
  So the model is to predict the probability of outcome "1". Take a baby example of 3 observations and 2 values of λ to show the usage of `predict()` function:

```
newdat = as.matrix(trainx[1:3, ])
predict(fit, newdat, type = "link", s = c(2.833e-02, 3.110e-02))
```

```
##           1       2
## 1  0.1943  0.1443
## 2 -0.9913 -1.0077
## 3 -0.5841 -0.5496
```

For example:

```
cvfit = cv.glmnet(as.matrix(trainx), trainy, family = "binomial", type.measure = "class")
plot(cvfit)
```


The above uses error rate as performance criteria and use 10-fold cross-validation. Similarly, you can get the λ value for the minimum error rate and the error rate that is 1 standard error from the minimum:

```
cvfit$lambda.min
```

```
## [1] 2.643e-05
```

```
cvfit$lambda.1se
```

```
## [1] 0.003334
```

You can use the same way to get the parameter estimates and make prediction.

10.5.2.3   Group lasso logistic regression

For models with categorical survey questions (explanatory variables), however, the original lasso algorithm only selects individual dummy variables instead of sets of the dummy variables grouped by the question in the survey. Another disadvantage of applying lasso to grouped variables is that the estimates are affected by the way dummy variables are encoded. Thus the group lasso (Yuan and Lin, 2007) method has been proposed to enable variable selection in linear regression models on groups of variables, instead of on single variables. For logistic regression models, the group lasso algorithm was first studied by Kim et al. (Y. Kim and Kim, 2006). They proposed a gradient descent algorithm to solve the corresponding constrained problem, which does, however, depend on unknown constants. Meier et al. (L Meier and Buhlmann, 2008) proposed a new algorithm that could work directly on the penalized problem and its convergence property does not depend on unknown constants. The algorithm is especially suitable for high-dimensional problems. It can also be applied to solve the corresponding convex optimization problem in generalized linear models. The group lasso estimator proposed by Meier et al. (L Meier and Buhlmann, 2008) for logistic regression has been shown to be statistically consistent, even with a large number of categorical predictors.

In this section, we illustrate how to use the logistic group lasso algorithm to construct risk scoring systems for predicting disease. Instead of maximizing the log-likelihood in the maximum likelihood method, the logistic group lasso estimates are calculated by minimizing the convex function:

$$S_\lambda(\ ) = -l(\ ) + \lambda \sum_{g=1}^{G} s(df_g) \parallel_{\mathbf{g}} \parallel_2$$

where λ is a tuning parameter for the penalty and $s(\cdot)$ is a function to rescale the penalty. In lasso algorithms, the selection of λ is usually determined by cross-validation using data. For $s(\cdot)$, we use the square root function $s(df_g) = df_g^{0.5}$ as suggested in Meier et

al.(L Meier and Buhlmann, 2008). It ensures the penalty is of the order of the number of parameters df_g as used in (Yuan and Lin, 2007).

Here we consider selection of the tuning parameter λ from a multiplicative grid of 148 values $\{0.96\lambda_{max}, 0.96^2\lambda_{max}, 0.96^3\lambda_{max}, ..., 0.96^{148}\lambda_{max}\}$, as in Meier et al. (L Meier and Buhlmann, 2008). Here λ_{max} is defined as

$$\lambda_{max} = \max_{g\in\{1,...,G\}} \left\{ \frac{1}{s(df_g)} \parallel \mathbf{x_g}^T(\mathbf{y}-\bar{\mathbf{y}}) \parallel_2 \right\}, \qquad (10.8)$$

such that when $\lambda = \lambda_{max}$, only the intercept is in the model. When λ goes to 0, the model is equivalent to ordinary logistic regression.

11

Tree-Based Methods

The tree-based models can be used for regression and classification. The goal is to stratify or segment the predictor space into a number of sub-regions. For a given observation, use the mean or the mode of the training observations in the sub-region as the prediction. Tree-based methods are conceptually simple yet powerful. This type of model is often referred to as Classification And Regression Trees (CART). They are popular tools for many reasons:

1. Do not require user to specify the form of the relationship between predictors and response
2. Do not require (or if they do, very limited) data pre-processing and can handle different types of predictors (sparse, skewed, continuous, categorical, etc.)
3. Robust to co-linearity
4. Can handle missing data
5. Many pre-built packages make implementation as easy as a button push

CART can refer to the tree model in general, but most of the time, it represents the algorithm initially proposed by Breiman (et. al, 1984). After Breiman, there are many new algorithms, such as ID3, C4.5, and C5.0. C5.0 is an improved version of C4.5, but since C5.0 is not open source, the C4.5 algorithm is more popular. C4.5 was a major competitor of CART. But now, all those seem outdated. The most popular tree models are Random Forest (RF) and Gradient Boosting Machine (GBM). Despite being out of favor in application, it is important to understand the mechanism of the basic tree algorithm. Because the later models are based on the same foundation.

The original CART algorithm targets binary classification, and the later algorithms can handle multi-category classification. A single tree is easy to explain but has poor accuracy. More complicated tree models, such as RF and GBM, can provide much better prediction at the cost of explainability. As the model becoming more complicated, it is more like a black-box which makes it very difficult to explain the relationship among predictors. There is always a trade-off between explainability and predictability.
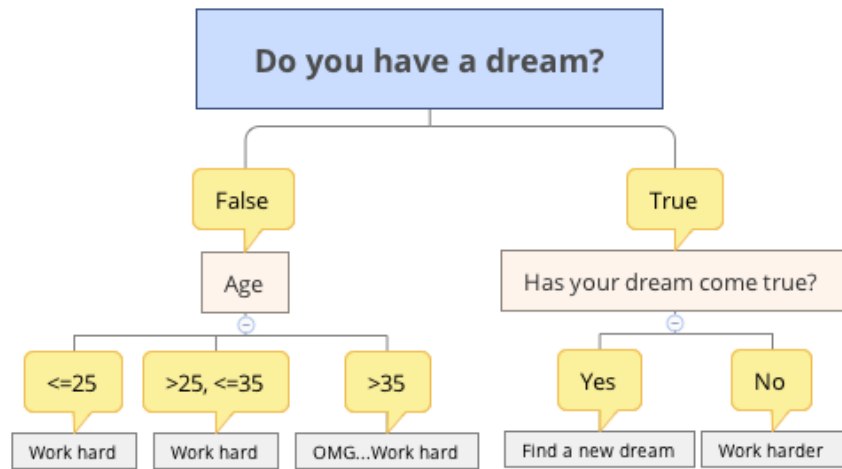
The reason why it is called "tree" is of course because the structure has similarities. But the direction of the decision tree is opposite to the real tree, the root is on the top, and the leaf is on the bottom. From the root node, a decision tree divides to different branches and generates more nodes. The new nodes are child nodes, and the previous node is the parent node. At each child node, the algorithm will decide whether to continue dividing. If it stops, the node is called a leaf node. If it continues, then the node becomes the new parent node and splits to produce the next layer of child nodes. At each non-leaf node, the algorithm needs to decide split into branches. The leaf node contains the final "decision," final class the sample belongs to or the sample's value has. Here are the important definitions in the tree model:

- **Classification tree**: the outcome is discrete

- **Regression tree**: the outcome is continuous (e.g. the price of a house, or a patient's length of stay in a hospital)
- **Non-leaf node (or split node)**: the algorithm needs to decide a split at each non-leaf node (eg: $age \leq 25$, $25 < age \leq 35$, $age > 35$)
- **Root node** the beginning node where the tree starts
- **Leaf node (or Terminal node)**: the node stops splitting. It has the final decision of the model
- **Degree of the node**: the number of subtrees of a node
- **Degree of the tree**: the maximum degree of a node in the tree
- **Pruning**: remove parts of the tree that do not provide power to classify instances
- **Branch (or Subtree)**: the whole part under a non-leaf node
- **Child**: the node directly after and connected to another node
- **Parent**: the converse notion of a child

Single tree is easy to explain but has high variance and low accuracy, and hence is very limited. Minor changes in the training data can lead to large changes in the fitted tree. A series of rectangular decision regions defined by a single tree is often too naive to represent the relationship between the dependent variable and the predictors. To overcome these shortcomings, researchers have proposed ensemble methods which combine many trees. Ensemble tree models typically have much better predictive performance than a single tree. We will introduce those models in later sections.

11.1   Splitting Criteria

The splitting criteria used by the regression tree and the classification tree are different. Like the regression tree, the goal of the classification tree is to divide the data into smaller, more homogeneous groups. Homogeneity means that most of the samples at each node are from one class. The original CART algorithm uses Gini impurity as the splitting criterion; The later ID3, C4.5, and

C5.0 use entropy. We will look at three most common splitting criteria.

Gini impurity

Gini impurity(et. al, 1984) is a measure of non-homogeneity. It is widely used in classification tree. For a two-class problem, the Gini impurity for a given node is defined as:

$$p_1(1 - p_1) + p_2(1 - p_2)$$

where p_1 and p_2 are probabilities for the two classes respectively. It is easy to see that when the sample set is pure, one of the probability is 0 and the Gini score is the smallest. Conversely, when $p_1 = p_2 = 0.5$, the Gini score is the largest, in which case the purity of the node is the smallest. Let's look at an example. Suppose we want to determine which students are computer science (CS) majors. Here is the simple hypothetical classification tree result obtained with the gender variable.
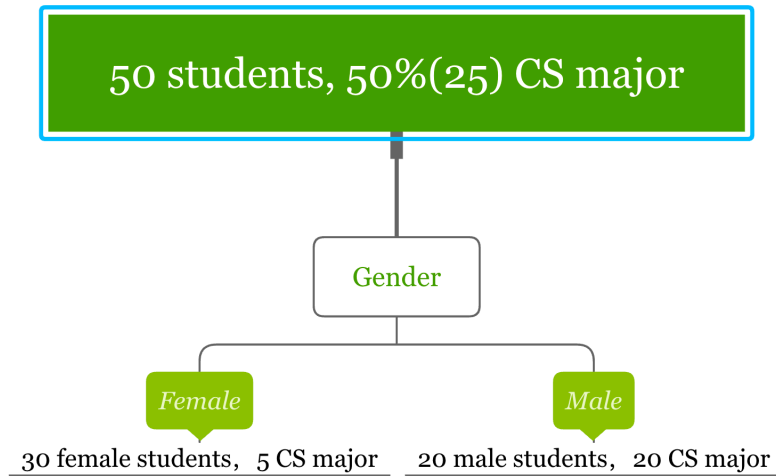
Let's calculate the Gini impurity for splitting node "Gender":

1. Gini impurity for "Female" $= \frac{1}{6} \times \frac{5}{6} + \frac{5}{6} \times \frac{1}{6} = \frac{5}{18}$
2. Gini impurity for "Male" $= 0 \times 1 + 1 \times 0 = 0$

The Gini impurity for the node "Gender" is the following weighted average of the above two scores:

$$\frac{3}{5} \times \frac{5}{18} + \frac{2}{5} \times 0 = \frac{1}{6}$$

The Gini impurity for the 50 samples in the parent node is $\frac{1}{2}$. It is easy to calculate the Gini impurity drop from $\frac{1}{2}$ to $\frac{1}{6}$ after splitting. The split using "gender" causes a Gini impurity decrease of $\frac{1}{3}$. The algorithm will use different variables to split the data and choose the one that causes the most substantial Gini impurity decrease.

Information gain

Looking at the samples in the following three nodes, which one is the easiest to describe? It is obviously C. Because all the samples in C are of the same type, so the description requires the least amount of information. On the contrary, B needs more information, and A needs the most information. In other words, C has the highest purity, B is the second, and A has the lowest purity. We need less information to describe nodes with higher purity.

A   B   C
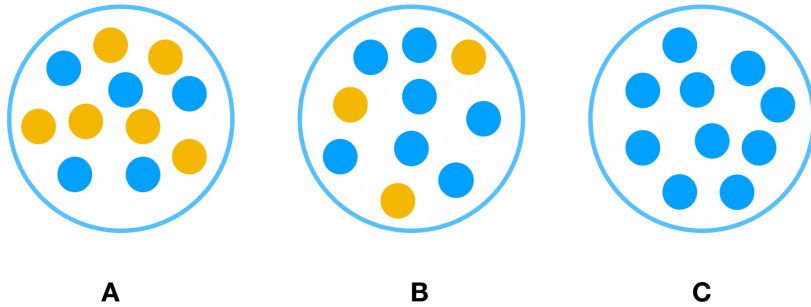
A measure of the degree of disorder is entropy which is defined as:

$$Entropy = -plog_2p - (1-p)log_2(1-p)$$

where p is the percentage of one type of samples. If all the samples in one node are of one type (such as C), the entropy is 0. If the proportion of each type in a node is 50% 50%, the entropy is 1. We can use entropy as splitting criteria. The goal is to decrease entropy as the tree grows.

Similarly, the entropy of a splitting node is the weighted average of the entropy of each child. In the above tree for the students, the entropy of the root node with all 50 students is $-\frac{25}{50}log_2\frac{25}{50} - \frac{25}{50}log_2\frac{25}{50} = 1$. Here an entropy of 1 indicates that the purity of the node is the lowest, that is, each type takes up half of the samples.

The entropy of the split using variable "gender" can be calculated in three steps:

1. Entropy for "Female" $= -\frac{5}{30}log_2\frac{5}{30} - \frac{25}{30}log_2\frac{25}{30} = 0.65$
2. Entropy for "Male" $= 0 \times 1 + 1 \times 0 = 0$
3. Entropy for the node "Gender" is the weighted average of the above two entropy numbers: $\frac{3}{5} \times 0.65 + \frac{2}{5} \times 0 = 0.39$

So entropy decreases from 1 to 0.39 after the split.

Sum of Square Error (SSE)

The previous two metrics are for classification tree. The SSE is the most widely used splitting metric for regression. Suppose you want to divide the data set S into two groups of S_1 and S_2, where the selection of S_1 and S_2 needs to minimize the sum of squared errors:

$$SSE = \Sigma_{i \in S_1}(y_i - \bar{y}_1)^2 + \Sigma_{i \in S_2}(y_i - \bar{y}_2)^2 \qquad (11.1)$$

In equation (11.1), $\bar{y}_1$ and $\bar{y}_1$ are the average of the sample in S_1 and S_2. The way regression tree grows is to automatically decide on the splitting variables and split points that can maximize **SSE reduction**. Since this process is essentially a recursive segmentation, this approach is also called recursive partitioning.

Take a look at this simple regression tree for the height of 10 students:
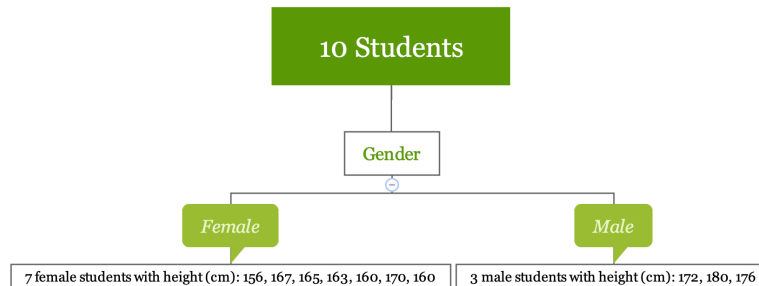
You can calculate the SSE using the following code:

1. SSE for "Female" is 136
2. SSE for "Male" is 32
3. SSE for splitting node "Gender" is the sum of the above two numbers which is 168

SSE for the 10 students in root node is 522.9. After the split, SSE decreases from 522.9 to 168.

If there is another possible way of splitting, divide it by major, as follows:
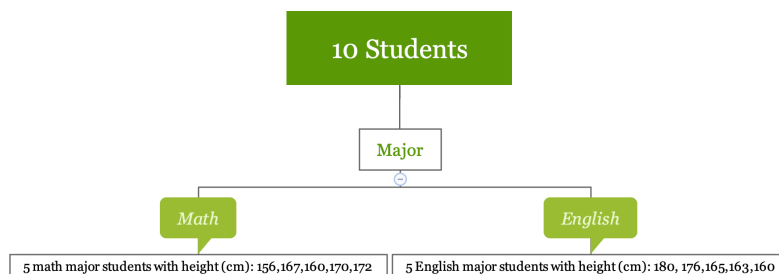
In this situation:

1. SSE for "Math" is 184
2. SSE for "English" is 302.8
3. SSE for splitting node "Major" is the sum of the above two numbers which is 486.8

Splitting data using variable "gender" reduced SSE from 522.9 to 168; using variable "major" reduced SSE from 522.9 to 486.8. Based on SSE reduction, you should use gender to split the data.

The three splitting criteria mentioned above are the basis for building a tree model.

11.2   Tree Pruning

Pruning is the process that reduces the size of decision trees. It reduces the risk of overfitting by limiting the size of the tree or removing sections of the tree that provide little power.

Limit the size

You can limit the tree size by setting some parameters.

- Minimum sample size at each node: Defining the minimum sample size at the node helps to prevent the leaf nodes having only one sample. The sample size can be a tuning parameter. If it is too large, the model tends to under-fit. If it is too small, the model tends to over-fit. In the case of severe class imbalance, the minimum sample size may need to be smaller because the number of samples in a particular class is small.

- Maximum depth of the tree: If the tree grows too deep, the model tends to over-fit. It can be a tuning parameter.

- Maximum number of terminal nodes: Limit on the terminal nodes works the same as the limit on the depth of the tree. They are proportional.

- The number of variables considered for each split: the algorithm randomly selects variables used in finding the optimal split point at each level. In general, the square root of the number of all variables works best, which is also the default setting for many functions. However, people often treat it as a tuning parameter.

Remove branches

Another way is to first let the tree grow as much as possible and then go back to remove insignificant branches. The process reduces the depth of the tree. The idea is to overfit the training set and then correct using the testing set. There are different implementations.

- cost/complexity penalty

The idea is that the pruning minimizes the penalized error SSE_λ with a certain value of tuning parameter λ.

$$SSE_\lambda = SSE + \lambda \times (complexity)$$

Here complexity is a function of the number of leaves. For every given λ, we want to find the smallest tree that minimizes this penalized error. Breiman presents the algorithm to solve the optimization(et. al, 1984).

To find the optimal pruning tree, you need to iterate through a series of values of λ and calculate the corresponding SSE. For the same λ, SSE changes over different samples. Breiman et al. suggested using cross-validation (et. al, 1984) to study the variation of SSE under each λ value. They also proposed a standard deviation criterion to give the simplest tree: within one standard deviation, find the simplest tree that minimizes the absolute error. Another method is to choose the tree size that minimizes the numerical error (Hastie T, 2008).

- Error-based pruning

This method was first proposed by Quinlan (Quinlan, 1999). The idea behind is intuitive. All split nodes of the tree are included in the initial candidate pool. Pruning a split node means removing the entire subtree under the node and setting the node as a terminal node. The data is divided into 3 subsets for:

(1) training a complete tree;

(2)   pruning;

(3)   testing the final model.

You train a complete tree using the subset (1) and apply the tree on the subset (2) to calculate the accuracy. Then prune the tree based on a node and apply that on the subset (2) to calculate another accuracy. If the accuracy after pruning is higher or equal to that from the complete tree, then we set the node as a terminal node. Otherwise, keep the subtree under the node. The advantage of this method is that it is easy to compute. However, when the size of the subset (2) is much smaller than that of the subset (1), there is a risk of over-pruning. Some researchers found that this method results in more accurate trees than pruning process based on tree size (F. Espoito and Semeraro, 1997).

• Error-complexity pruning

This method is to search for a trade-off between error and complexity. Assume we have a splitting node t, and the corresponding subtree T. The error cost of the node is defined as:

$$R(t) = r(t) \times p(t)$$

where $r(t)$ is the error rate associate with the node:

$$r(t) = \frac{misclassified\ sample\ size\ of\ the\ node}{sample\ size\ of\ the\ node}$$

$p(t)$ is the ratio of the sample of the node to the total sample

$$p(t) = \frac{sample\ size\ of\ the\ node}{total\ sample\ size}$$

If we keep note t, the error cost of the subtree T is:

$$R(T) = \Sigma_{i=no.\ of\ leaves} R(i)$$

The error-complexity measure of the node is:

$$a(t) = \frac{R(t) - R(T)_t}{no.\ of\ leaves - 1}$$

Based on the metrics above, the pruning process is (Patel and Upadhyay, 2012)

1. Calculate a for every node t
2. Prune the node with the lowest value
3. Repeat 1 and 2. It produces a pruned tree each time and they form a forest.
4. Select the tree with the best overall accuracy

- Minium error pruning

Niblett and Brotko came it up in 1991(Cestnik and Bratko, 1991). The process is bottom-up which seeks a single tree that minimizes the expected error rate on new samples. If we prune a splitting point t, all the samples under t will be classified as from one category, say category c. If we prune the subtree, the expected error rate is:

$$E(t) = \frac{n_t - n_{t,c} + k - 1}{n_t + k}$$

where:

$$k = number\ of\ categories$$
$$n_t = sample\ size\ under\ node\ t$$
$$n_{t,c} = number\ of\ sample\ under\ t\ that\ belong\ to\ category\ c$$

Based on the above definition, the pruning process is(F. Espoito and Semeraro, 1997):

- Calculate the expected error rate for each non-leave node if that subtree is pruned
- Calculate the expected error rate for each non-leave node if that subtree is not pruned
- If pruning the node leads to higher expected rate, then keep the subtree; otherwise, prune it.

11.3   Regression and Decision Tree Basic

11.3.1   Regression Tree

Let's look at the process of building a regression tree(Gareth James and Tibshirani, 2015). There are two steps:

1. Divide predictors space — that is a set of possible values of $X_1, X_2, ..., X_p$— into J distinct and non-overlapping regions: $R_1, R_2, ..., R_J$
2. For every observation that falls into the region R_j, the prediction in the mean of the response values for the training observations in R_j

Let's go back to the previous baby example. If we use the variable "Gender" to divide the observations, we obtain two regions R_1 (female) and R_2 (male).
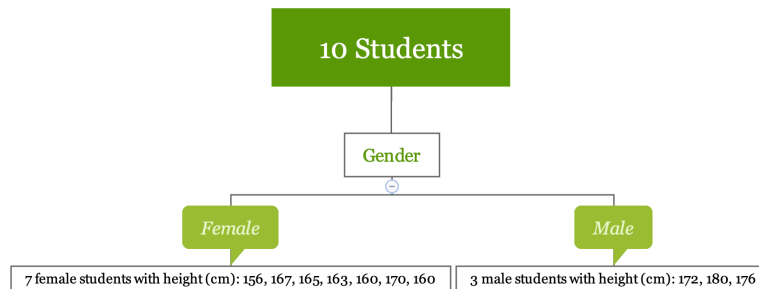

```
y1=c(156,167,165,163,160,170,160)
y2=c(172,180,176)
```

The sample average for region R_1 is 163, for region R_2 is 176. For a new observation, if it is female, the model predicts the height to be 163, if it is male, the predicted height is 176. Calculating the

mean is easy. Let's look at the first step in more detail which is to divide the space into $R_1, R_2, \ldots, R_J$.

In theory, the region can be any shape. However, to simplify the problem, we divide the predictor space into high-dimensional rectangles. The goal is to divide the space in a way that minimize RSS. Practically, it is nearly impossible to consider all possible partitions of the feature space. So we use an approach named recursive binary splitting, a top-down , greedy algorithm. The process starts from the top of the tree (root node) and then successively splits the predictor space. Each split produces two branches (hence binary). At each step of the process, it chooses the best split at that particular step, rather than looking ahead and picking a split that leads to a better tree in general (hence greedy).

$$R_1(j, s) = \{X | X_j < s\} \ and \ R_2(j, s) = \{X | X_j \geq s\}$$

Calculate the RSS decrease after the split. For different (j, s), search for the combination that minimizes the RSS, that is to minimize the following:

$$\Sigma_{i:x_i \in R_1(j,s)}(y_i - \hat{y}_{R_1})^2 + \Sigma_{i:x_i \in R_2(j,s)}(y_i - \hat{y}_{R_2})^2$$

where $\hat{y}_{R_1}$ is the mean of all samples in R_1, $\hat{y}_{R_2}$ is the mean of samples in R_2. It can be quick to optimize the equation above. Especially when p is not too large.

Next, we continue to search for the split that optimize the RSS. Note that the optimization is limited in the sub-region. The process keeps going until a stopping criterion is reaches. For example, continue until no region contains more than 5 samples or the RSS decreases less than 1%. The process is like a tree growing.
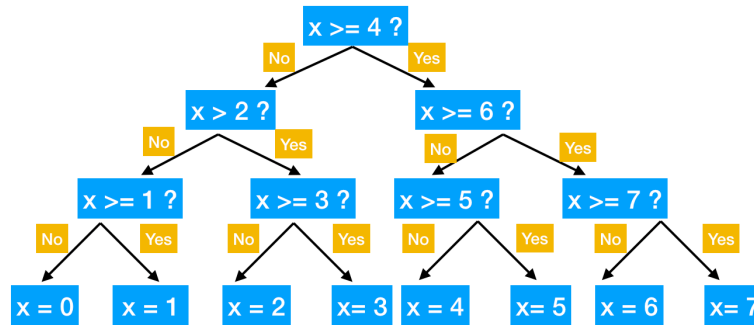
There are multiple R packages for building regression tree, such as `ctree`, `rpart` and `tree`. `rpart` is widely used for building single tree. The split is based on CART algorithm, using `rpart()` function from the package. There are some parameters that controls the model fitting, such as the minimum number of observations that must exist in a node in order for a split to be attempted, the minimum number of observations in any leaf node etc. You can can set those parameter using `rpart.control`.

A more convenient way is to use `train()` function in `caret` package. The package can call `rpart()` function and train the model through cross-validation. In this case, the most common parameters are `cp` (complexity parameter) and `maxdepth` (the maximum depth of any node of the final tree). To tune the complexity parameter, set `method = "rpart"`. To tune the maximum tree depth, set `method = "rpart2"` :

```r
library(rpart)
dat <- read.csv("http://bit.ly/2P5gTw4")
# data cleaning: delete wrong observations
dat <- subset(dat, store_exp > 0 & online_exp > 0)
# use the 10 survey questions as predictors
trainx <- dat[, grep("Q", names(dat))]
# use the sum of store and online expenditure as response variable
# total expenditure = store expenditure + online expenditure
```

```
trainy <- dat$store_exp + dat$online_exp
set.seed(100)
rpartTune <- train(trainx, trainy, method = "rpart2", tuneLength = 10,
    trControl = trainControl(method = "cv"))
plot(rpartTune)
```
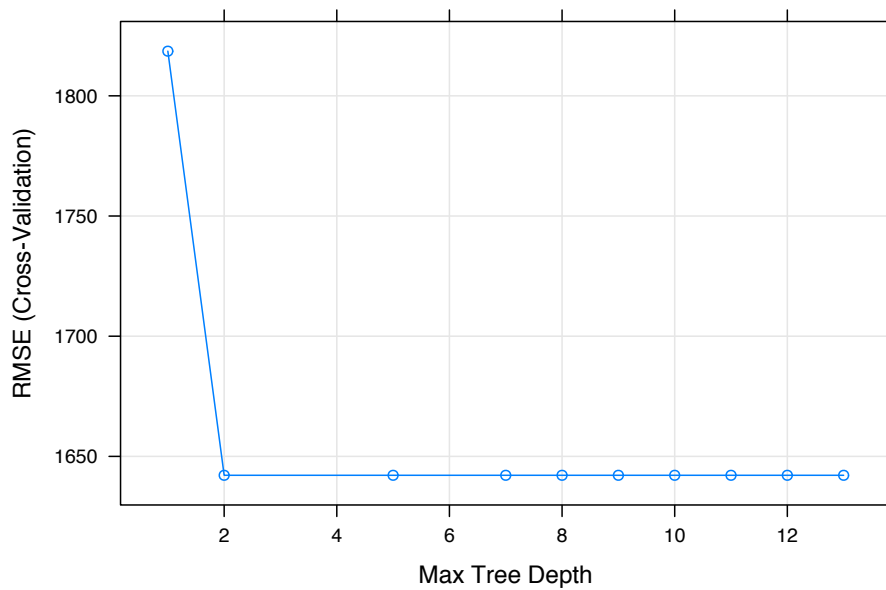

RMSE doesn't change much when the maximum is larger than 2. So we set the maximum depth to be 2 and refit the model:

```
rpartTree <- rpart(trainy ~ ., data = trainx, maxdepth = 2)
```

You can check the result using `print()`:

```
print(rpartTree)
```

```
## n= 999
##
```

```
## node), split, n, deviance, yval
##        * denotes terminal node
##
## 1) root 999 1.581e+10  3479.0
##   2) Q3< 3.5 799 2.374e+09  1819.0
##     4) Q5< 1.5 250 3.534e+06   705.2 *
##     5) Q5>=1.5 549 1.919e+09  2326.0 *
##   3) Q3>=3.5 200 2.436e+09 10110.0 *
```

You can see that the final model picks Q3 and Q5 to predict total expenditure. To visualize the tree, you can convert rpart object to party object using partykit then use plot() function:

```
library(partykit)
rpartTree2 <- as.party(rpartTree)
plot(rpartTree2)
```
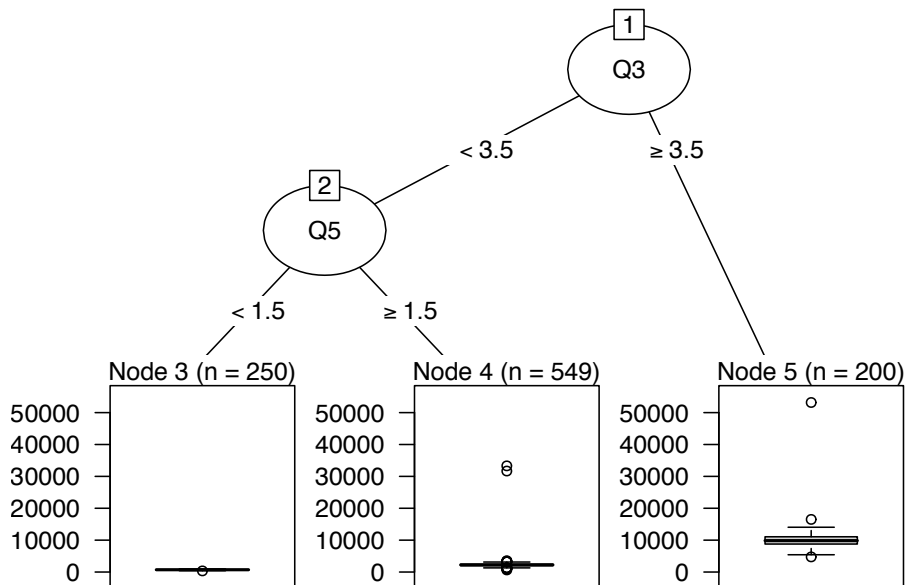
11.3.2 Decision Tree

Similar to a regression tree, the goal of a classification tree is to stratifying the predictor space into a number of sub-regions that are more homogeneous. The difference is that a classification tree is used to predict a categorical response rather than a continuous one. For a classification tree, the prediction is the most commonly occurring class of training observations in the region to which an observation belongs. The splitting criteria for a classification tree are different. The most common criteria are entropy and Gini impurity. CART uses Gini impurity and C4.5 uses entropy.

When the predictor is continuous, the splitting process is straightforward. When the predictor is categorical, the process can take different approaches:

1. Keep the variable as categorical and group some categories on either side of the split. In this way, the model can make more dynamic splits but must treat the categorical predictor as an ordered set of bits.
2. Encode the categorical variable to be a set of binary dummy variables. In this way, the information in the categorical variable is decomposed to independent bits of information. The model considers these dummy variables separately and evaluates for each of these on one split point (because there are only two possible values: 0/1).

When fitting tree models, people need to choose the way to treat categorical predictors. If you know some of the categories have higher predictability, then the first approach may be better. In the rest of this section, we will build tree models using the above two approaches and compare them.

Let's build a classification model to identify the gender of the customer:

```r
library(caret)
library(pROC)
```

```
dat <- read.csv("http://bit.ly/2P5gTw4")
# use the 10 survey questions as predictors
trainx1 <- dat[, grep("Q", names(dat))]
# add a categorical predictor
# use two ways to treat categorical predictor
# trainx1: use approach 1, without encoding
trainx1$segment <- dat$segment

# trainx2: use approach 2, encode it to a set of dummy variables
dumMod<-dummyVars(~.,
                  data=trainx1,
                  # Combine the previous variable name and the level name
                  # as the new dummy variable name
                  levelsOnly=F)
trainx2 <- predict(dumMod,trainx1)
# the response variable is gender
trainy <- dat$gender
```

Here we use `train()` function in `caret` package to call `rpart` to build the model. We can compare the model results from the two approaches:

```
CART

1000 samples
  11 predictor
   2 classes: 'Female', 'Male'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 901, 899, 900, 900, 901, 900, ...
Resampling results across tuning parameters:

  cp       ROC     Sens    Spec
  0.00000  0.6937  0.6517  0.6884
  0.00835  0.7026  0.6119  0.7355
```

```
 0.01670   0.6852   0.5324   0.8205
 0.02505   0.6803   0.5107   0.8498
 0.03340   0.6803   0.5107   0.8498

 ......

 0.23380   0.6341   0.5936   0.6745
 0.24215   0.5556   0.7873   0.3240
```

```
ROC was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.00835.
```

The above keeps the variable as categorical without encoding. Here `cp` is the complexity parameter. It is used to decide when to stop growing the tree. `cp = 0.01` means the algorithm only keeps the split that improves the corresponding metric by more than 0.01. Next, let's encode the categorical variable to be a set of dummy variables and fit the model again:

```
rpartTune2 <- caret::train(trainx2, trainy, method = "rpart",
                        tuneLength = 30,
                        metric = "ROC",
                        trControl = trainControl(method = "cv",
                                                    summaryFunction = twoClassSummary,
                                                    classProbs = TRUE,
                                                    savePredictions = TRUE))
```

Compare the results of the two approaches.

```
rpartRoc <- roc(response = rpartTune1$pred$obs,
                predictor = rpartTune1$pred$Female,
                levels = rev(levels(rpartTune1$pred$obs)))
```

```
## Setting direction: controls < cases
```

```
rpartFactorRoc <- roc(response = rpartTune2$pred$obs,
                      predictor = rpartTune2$pred$Female,
                      levels = rev(levels(rpartTune1$pred$obs)))
```

```
## Setting direction: controls < cases
```

```
plot(rpartRoc, type = "s", print.thres = c(.5),
     print.thres.pch = 3,
     print.thres.pattern = "",
     print.thres.cex = 1.2,
     col = "red", legacy.axes = TRUE,
     print.thres.col = "red")

plot(rpartFactorRoc,
     type = "s",
     add = TRUE,
     print.thres = c(.5),
     print.thres.pch = 16, legacy.axes = TRUE,
     print.thres.pattern = "",
     print.thres.cex = 1.2)
legend(.75, .2,
       c("Grouped Categories", "Independent Categories"),
       lwd = c(1, 1),
       col = c("black", "red"),
       pch = c(16, 3))
```
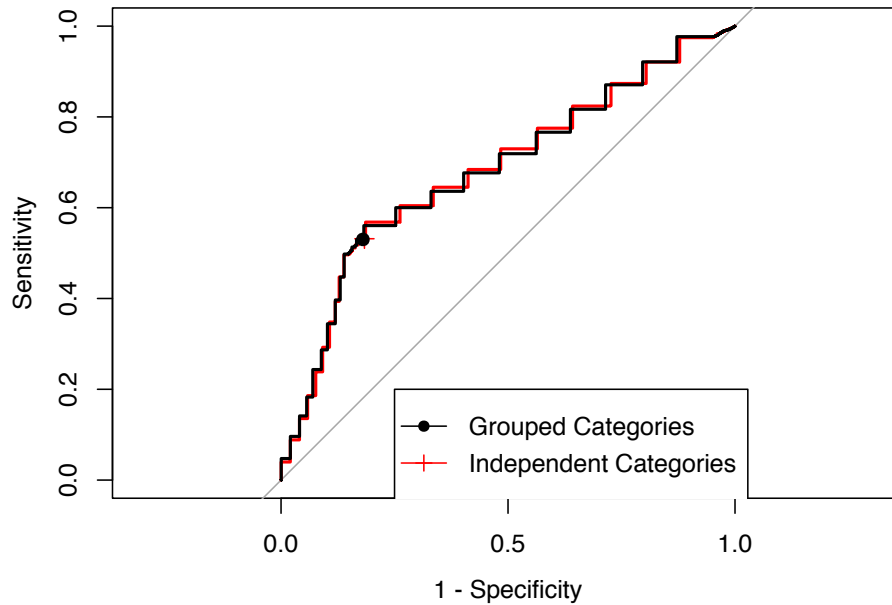
You can see that in this case, the two approaches lead to similar model performance.

Single tree is straightforward and easy to interpret but it has problems:

1. Low accuracy
2. Unstable: little change in the training data leads to very different trees.

One way to overcome those is to use an ensemble of trees. In the rest of this chapter, we will introduce three ensemble methods (combine many models' predictions): bagging tree, random forest, and gradient boosted machine. Those ensemble approaches have significant higher accuracy and stability. However, it comes with the cost of interpretability.

11.4   Bagging Tree

As mentioned before, a single tree is unstable. If you randomly separate the sample to be two parts and fit tree model on each, you can get two very different trees. A stable model should give a similar result on different random samples. Some traditional statistical models have high stability, such as linear regression. *Ensemble methods* appeared in the 1990s which can effectively stabilize the model. *Bootstrapping* is a type of process where you repeated draw samples of the same size from a single original sample with replacement (B and R, 1986). *Bootstrap aggregation* (Bagged) is an ensemble technique proposed by Leo Breiman (Breiman 1996a). It uses bootstrapping in conjunction with any model to construct an ensemble. The process is very straightforward:

1.   Build a model on different bootstrap samples to form an ensemble, say B samples
2.   For a new sample, each model will give a prediction: $\hat{f}^1(x), \hat{f}^2(x) ... , \hat{f}^B(x)$
3.   The bagged model's prediction is the average of all the predictions:

$$\hat{f}_{avg}(x) = \frac{1}{B}\Sigma_{b=1}^{B}\hat{f}^b(x)$$

Assume that there are n independent random variables $Z_1, ... , Z_n$ with variance σ^2. Then the variance of the mean $\bar{Z}$ is $\frac{\sigma^2}{n}$. It is easy to see why bagged models have less variance. Since bootstrapping is to sample with replacement, it means some samples are selected multiple times and some not at all. Those left out samples are called out-of-bag. You can use the out-of-bag sample to access the model performance. For regression, the prediction is a simple average. For classification, the prediction is the category with the most "votes". Here, the number of trees, B is a parameter you need to decide, i.e. tuning parameter. **Bagging is a general approach**

that can be applied to different learners. Here we only discuss in the context of decision trees.

The advantages of bagging tree are:

- Bagging stabilizes the model predictions by averaging the results. If we have 10 bootstrap samples and fit a single tree on each of those, we may get 10 trees with very different structures and leading to different predictions for a new sample. But if we use the average of the 10 predictions as the final prediction, then the result should be much more stable. It means if we have another 10 samples and do it all-over again, we will get very similar averaged prediction.

- Bagging provides more accurate predictions. If the goal is to predict rather than interpret, then the ensemble approach definitely has an advantage, especially for unstable models. However, for stable models (such as regression, MARS), bagging may bring marginal improvement for the model performance.

- Bagging can use out-of-bag samples to evaluate model performance. For each model in the ensemble, we can calculate the value of the model performance metric (you can decide what metric to use). You can use the average of all the out-of-bag performance values to gauge the predictive performance of the entire ensemble. This correlates well with either cross-validation estimates or test set estimates. On average, each tree uses about 2/3 of the samples, and the rest 1/3 is used as out-of-bag. When the number of bootstrap samples is large enough, the out-of-bag performance estimate approximates that from leave one out cross-validation.

You need to choose the number of bootstrap samples. The author of "Applied Predictive Modeling" (Kuhn and Johnston, 2013) points out that often people see an exponential decrease in predictive improvement as the number of iterations increases. Most of the predictive power is from a small portion of the trees. Based on their experience, model performance can have small improvements up to 50 bagging iterations. If it is still not satisfying, they

suggest trying other more powerfully predictive ensemble methods such as random forests and boosting which will be described in the following sections.

The disadvantages of bagging tree are:

- As the number of bootstrap samples increases, the computation and memory requirements increase as well. You can mitigate this disadvantage by parallel computing. Since each bootstrap sample and modeling is independent of any other sample and model, you can easily parallelize the bagging process by building those models separately and bring back the results in the ed to generate the prediction.

- The bagged model is difficult to explain which is common for all ensemble approaches. However, you can still get variable importance by combining measures of importance across the ensemble. For example, we can calculate the RSS decrease for each variable across all trees and use the average as the measurement of the importance.

- Since the bagging tree uses all of the original predictors as everey split of every tree, those trees are related with each other. The tree correlation prevents bagging from optimally reducing the variance of the predicted values. See (Hastie T, 2008) for a mathematical illustration of the tree correlation phenomenon.

Let's look at how to use R to build bagging tree. Get the predictors and response variable first:

```r
dat <- read.csv("http://bit.ly/2P5gTw4")
# use the 10 survey questions as predictors
trainx <- dat[, grep("Q", names(dat))]
# add segment as a predictor
# don't need to encode it to dummy variables
trainx$segment <- dat$segment
# use gender as the response variable
trainy <- as.factor(dat$gender)
```

Then fit the model using train function in caret package. Here we just set the number of trees to be 1000. You can tune that parameter.

```
set.seed(100)
bagTune <- caret::train(trainx, trainy,
                           method = "treebag",
                           nbagg = 2000,
                           metric = "ROC",
                           trControl = trainControl(method = "cv",
                           summaryFunction = twoClassSummary,
                           classProbs = TRUE,
                           savePredictions = TRUE))
```

The model results are:

```
bagTune
```

```
## Bagged CART
##
## 1000 samples
##   11 predictor
##    2 classes: 'Female', 'Male'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 901, 899, 900, 900, 901, 900, ...
## Resampling results:
##
##   ROC     Sens    Spec
##   0.7093  0.6533  0.6774
```

Since we only have a handful of variables in this example, the maximum AUC doesn't improve by using bagging tree. But it makes a difference when we have more predictors.

11.5   Random Forest

Since the tree correlation prevents bagging from optimally reducing the variance of the predicted values, a natural way to improve the model performance is to reduce the correlation among trees. That is what random forest aims to do: improve the performance of bagging by de-correlating trees.

From a statistical perspective, you can de-correlate trees by introducing randomness when you build each tree. One approach (T, 1998; Y and D, 1997) is to randomly choose m variables to use each time you build a tree. Dietterich(T, 2000) came up with the idea of random split selection which is to randomly choose m variables to use at each splitting node. Based on the different generalizations to the original bagging algorithm, Breiman (Breiman, 2001a) came up with a unified algorithm called *random forest.*

When building a tree, the algorithm randomly chooses m variables to use at each splitting node. Then choose the best one out of the m to use at that node. In general, people use $m = \sqrt{p}$. For example, if we use 10 questions from the questionnaire as predictors, then at each node, the algorithm will randomly choose 4 candidate variables. Since those trees in the forest don't always use the same variables, tree correlation is less than that in bagging. It tends to work better when there are more predictors. Since we only have 10 predictors here, the improvement from the random forest is marginal. The number of randomly selected predictors is a tuning parameter in the random forest. Since random forest is computationally intensive, we suggest starting with 5 value around $m = \sqrt{p}$. Another tuning parameter is the number of trees in the forest. You can start with 1000 trees and then increase the number until performance levels off.

The basic random forest is as follows:

1.   Select the number of trees, B
2.   **for** *i=1 to B* **do**

- generate a bootstrap sample of the original data
- train a tree on this sample
    - **for** *each split* **do**
        * randomly select m <p predictors
        * choose the best one out of the m and partition the data
    - **end**
- use typical tree model stopping criteria to determine when a tree is complete without pruning

3. **end**

When $m = p$, random forest is equal to the bagging tree. When the predictors are highly correlated, then smaller m tends to work better. Let's use the `caret` package to train a random forest:

```r
# tune across a list of numbers of predictors
mtryValues <- c(1:5)
set.seed(100)
rfTune <- train(x = trainx,
                y = trainy,
                # set the model to be random forest
                method = "rf",
                ntree = 1000,
                tuneGrid = data.frame(.mtry = mtryValues),
                importance = TRUE,
                metric = "ROC",
                trControl = trainControl(method = "cv",
                            summaryFunction = twoClassSummary,
                            classProbs = TRUE,
                            savePredictions = TRUE))
```

```r
rfTune
```

```
## Random Forest
##
## 1000 samples
##    11 predictor
##     2 classes: 'Female', 'Male'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 899, 900, 900, 899, 899, 901, ...
## Resampling results across tuning parameters:
##
##    mtry  ROC      Sens     Spec
##    1     0.7169   0.5341   0.8205
##    2     0.7137   0.6334   0.7175
##    3     0.7150   0.6478   0.6995
##    4     0.7114   0.6550   0.6950
##    5     0.7092   0.6514   0.6882
##
## ROC was used to select the optimal model using
##  the largest value.
## The final value used for the model was mtry = 1.
```

In this example, since the number of predictors is small, the result of the model indicates that the optimal number of candidate variables at each node is 1. The optimal AUC is not too much higher than that from bagging tree:

If you have selected the values of tuning parameters, you can also use the randomForest package to fit a random forest.

```
library(randomForest)
rfit = randomForest(trainy ~ ., trainx, mtry = 1, ntree = 1000)
```

Since bagging tree is a special case of random forest, you can fit the bagging tree by setting $mtry = p$. Function `importance()` can return the importance of each predictor:

```
importance(rfit)
```

```
##           MeanDecreaseGini
## Q1                  9.056
## Q2                  7.582
## Q3                  7.611
## Q4                 12.308
## Q5                  5.628
## Q6                  9.740
## Q7                  6.638
## Q8                  7.829
## Q9                  5.955
## Q10                 4.781
## segment            11.185
```

You can use varImpPlot() function to visualize the predictor importance:

```
varImpPlot(rfit)
```

rfit

MeanDecreaseGini
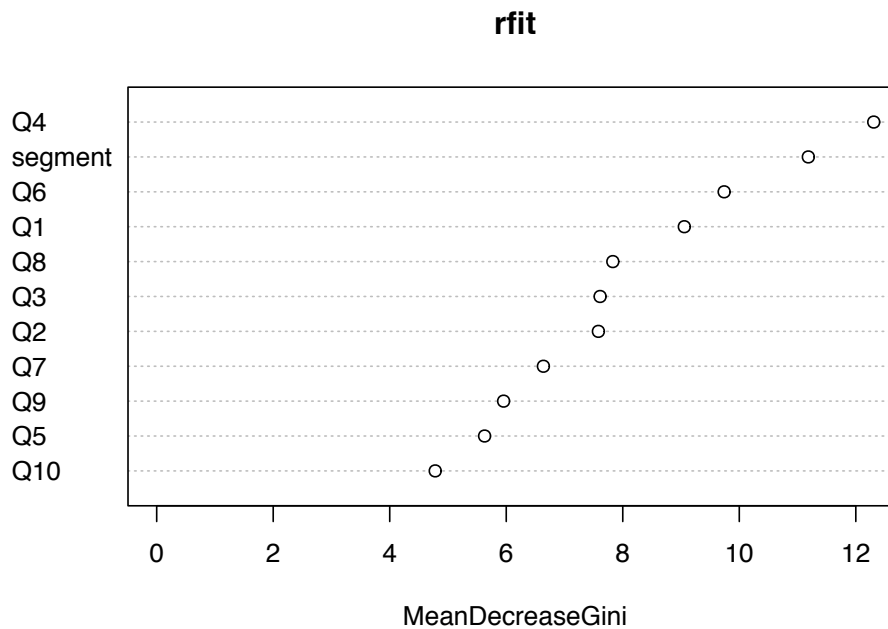
It is easy to see from the plot that `segment` and `Q4` are the top two variables to classify gender.

11.6   Gradient Boosted Machine

Boosting models were developed in the 1980s (L, 1984; M and L, 1989) and were originally for classification problems. Due to the excellent model performance, they were widely used for a variety of applications, such as gene expression (Dudoit S and T, 2002; et. al, 2000), chemical substructure classification (Varmuza K and K, 2003), music classification (et. al, 2006), etc. The first effective implementation of boosting is Adaptive Boosting (AdaBoost) algorithm came up by Yoav Freund and Robert Schapire in 1996 (YFR, 1999). After that, some researchers (Friedman J and R, 2000) started to connect the boosting algorithm with some statistical concepts, such as loss function, additive model, logistic regression. Friedman pointed out that boosting can be considered as a forward stagewise additive model that minimizes exponential loss. The new view of boosting in a statistical framework enabled the method to be extended to regression problems.

The idea is to combine a group of weak learners (a classifier that is marginally better than random guess) to produce a strong learner. Like bagging, boosting is a general approach that can be applied to different learners. Here we focus on the decision tree. Recall that both bagging and random forest create multiple copies of the original training data using the bootstrap, fitting a separate decision tree to each copy and combining all the results to create a single prediction. Boosting also creates different trees but the trees are grown sequentially and each tree is a weak learner. Any modeling technique with tuning parameters can produce a range of learners, from weak to strong. You can easily make a weak learner by restricting the depth of the tree. There are different types of boosting. Here we introduce two main types: adaptive boosting and stochastic gradient boosting.

11.6.1 Adaptive Boosting

Yoav Freund and Robert Schapire (Freund and Schapire, 1997) came up the AdaBoost.M1 algorithm. Consider a binary classification problem where the response variable has two categories $Y \in \{-1, 1\}$. Given predictor matrix, X, construct a classifier $G(X)$ that predicts 1 or -1. The corresponding error rate in the training set is:

$$\bar{err} = \frac{1}{N}\Sigma_{i=1}^{N}I(y_i \neq G(x_i))$$

The algorithm produces a series of classifiers $G_m(x)$, $m = 1, 2, ..., M$ from different iterations. In each iteration, it finds the best classifier based on the current weights. The misclassified samples in the mth iteration will have higher weights in the $m+1^{th}$ iteration and the correctly classified samples will have lower weights. As it moves on, the algorithm will put more effort into the "difficult" samples until it can correctly classify them. So it requires the algorithm to change focus at each iteration. At each iteration, the algorithm will calculate a stage weight based on the error rate. The final prediction is a weighted average of all those weak classifiers using stage weights from all the iterations:

$$G(x) = sign(\Sigma_{m=1}^{M}\alpha_m G_m(x))$$

where $\alpha_1, \alpha_2, ..., \alpha_M$ are the weights from different iterations.

AdaBoost.M1

1. Response variables have two values: +1 and -1

2. Initialize the observation to have the same weights: $w_i = \frac{1}{N}, i = 1, ..., N$

3. for m = 1 to M:

   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i

   (b) Compute the error rate for the mth classifier: $err_m = \frac{\Sigma_{i=1}^{N}w_i I(y_i \neq G_m(x_i))}{\Sigma_{i=1}^{N}w_i}$

     (c) Compute the stage weight for m^{th} iteration: $\alpha_m = log\frac{1-err_m}{err_m}$

     (d) Update $w_i = w_i \cdot exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, ..., N$

4. Calculate the prediction $G(x) = sign[\Sigma_{m=1}^{M}\alpha_m G_m(x)]$ where $sign(\cdot)$ means if $\cdot$ is positive, then the sample is classified as +1, -1 otherwise.

Since the classifier $G_m(x)$ returns discrete value, the AdaBoost.M1 algorithm is known as "Discrete AdaBoost" (Friedman J and R, 2000). You can revise the above algorithm if it returns continuous value, for example, a probability(Friedman J and R, 2000). As mentioned before, boosting is a general approach that can be applied to different learners. Since you can easily create weak learners by limiting the depth of the tree, the boosting tree is a common method. Since the classification tree is a low bias/high variance technique, ensemble decreases model variance and lead to low bias/low variance model. See Breinman(Breiman, 1998) for more explanation about why the boosting tree performs well in general. However, boosting can not significantly improve the low variance model. So applying boosting to Latent Dirichlet Allocation (LDA) or K-Nearest Neighbor (KNN) doesn't lead to as good improvement as applying boosting to statistical learning methods like naive Bayes (E and R, 1999).

11.6.2 Stochastic Gradient Boosting

As mentioned before, Friedman (Friedman J and R, 2000) provided a statistical framework for the AdaBoost algorithm and pointed out that boosting can be considered as a forward stagewise additive model that minimizes exponential loss. The framework led to some generalized algorithms such as Real AdaBoost, Gentle AdaBoost, and LogitBoost. Those algorithms later were unified under a framework called gradient boosting machine. The last section of the chapter illustrates how boosting can be considered as an additive model.

Consider a 2-class classification problem. You have the response $y \in \{0, 1\}$ and the sample proportion of class 1 from the training set is p. $f(x)$ is the model prediction in the range of $[-\infty, +\infty]$ and the predicted event probability is $\hat{p} = \frac{1}{1+exp[-f(x)]}$. The gradient boosting for this problem is as follows:

1. Initialize all predictions to the sample log-odds: $f_i = log\frac{\hat{p}}{1-\hat{p}}$
2. **for** j=1 ... M **do**

   - Compute predicted event probability: $\hat{p}_i = \frac{1}{1+exp[-f_i(x)]}$.
   - Compute the residual (i.e. gradient): $z_i = y_i - \hat{p}_i$
   - Randomly sample the training data
   - Train a tree model on the random subset using the residuals as the outcome
   - Compute the terminal node estimates of the Pearson residuals: $r_i = \frac{1/n\Sigma_i^n(y_i - \hat{p}_i)}{1/n\Sigma_i^n \hat{p}_i(1 - \hat{p}_i)}$
   - Update f $f_i = f_i + \lambda f_i^{(j)}$

3. end

When using the tree as the base learner, basic gradient boosting has two tuning parameters: tree depth and the number of iterations. You can further customize the algorithm by selecting a different loss function and gradient(Hastie T, 2008). The final line of the loop includes a regularization strategy. Instead of adding $f_i^{(j)}$ to the previous iteration's f_i, only a fraction of the value is added. This fraction is called learning rate which is λ in the algorithm. It can take values between 0 and 1 which is another tuning parameter of the model.

The way to calculate variable importance in boosting is similar to a bagging model. You get variable importance by combining measures of importance across the ensemble. For example, we can calculate the Gini index improvement for each variable across all trees and use the average as the measurement of the importance.

Boosting is a very popular method for classification. It is one of the

methods that can be directly applied to the data without requir-
ing a great deal of time-consuming data preprocessing. Applying
boosting on tree models significantly improves predictive accuracy.
Some advantages of trees that are sacrificed by boosting are speed
and interpretability.

Let's look at the R implementation.

```
gbmGrid <- expand.grid(interaction.depth = c(1, 3, 5, 7, 9),
                       n.trees = 1:5,
                       shrinkage = c(.01, .1),
                       n.minobsinnode = c(1:10))


set.seed(100)
gbmTune <- train(x = trainx,
                 y = trainy,
                 method = "gbm",
                 tuneGrid = gbmGrid,
                 metric = "ROC",
                 verbose = FALSE,
                 trControl = trainControl(method = "cv",
                         summaryFunction = twoClassSummary,
                         classProbs = TRUE,
                         savePredictions = TRUE))
```

```
# only show part of the output
gbmTune
```

```
Stochastic Gradient Boosting

1000 samples
  11 predictor
```

```
  2 classes: 'Female', 'Male'


No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 899, 900, 900, 899, 899, 901, ...
Resampling results across tuning parameters:

  shrinkage  interaction.depth  n.minobsinnode  n.trees  ROC     Sens    Spec
  0.01       1                  1               1        0.6821  1.0000  0.000000
  0.01       1                  1               2        0.6882  1.0000  0.000000
  0.01       1                  1               3        0.6936  1.0000  0.000000
  .
  .
  .
  0.01       5                  8               2        0.7127  1.0000  0.000000
  0.01       5                  8               3        0.7148  1.0000  0.000000
  0.01       5                  8               4        0.7096  1.0000  0.000000
  0.01       5                  8               5        0.7100  1.0000  0.000000
  0.01       5                  9               1        0.7006  1.0000  0.000000
  0.01       5                  9               2        0.7055  1.0000  0.000000
 [ reached getOption("max.print") -- omitted 358 rows ]


ROC was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 4, interaction.depth = 3, shrinkage
 = 0.01 and n.minobsinnode = 6.
```

The results show that the tuning parameter settings that lead to the best ROC are n.trees = 4 (number of trees), interaction.depth = 3 (depth of tree), shrinkage = 0.01 (learning rate) and n.minobsinnode = 6 (minimum number of observations in each node).

Now, let's compare the results from the three tree models.

```
treebagRoc <- roc(response = bagTune$pred$obs,
                    predictor = bagTune$pred$Female,
                    levels = rev(levels(bagTune$pred$obs)))
```

```
## Setting direction: controls < cases
```

```
rfRoc <- roc(response = rfTune$pred$obs,
             predictor = rfTune$pred$Female,
             levels = rev(levels(rfTune$pred$obs)))
```

```
## Setting direction: controls < cases
```

```
gbmRoc <- roc(response = gbmTune$pred$obs,
              predictor = gbmTune$pred$Female,
              levels = rev(levels(gbmTune$pred$obs)))
```

```
## Setting direction: controls < cases
```

```
plot(rpartRoc, type = "s", print.thres = c(.5),
     print.thres.pch = 16,
     print.thres.pattern = "",
     print.thres.cex = 1.2,
     col = "black", legacy.axes = TRUE,
     print.thres.col = "black")

plot(treebagRoc, type = "s", add = TRUE, print.thres = c(.5),
     print.thres.pch = 3, legacy.axes = TRUE, print.thres.pattern = "",
     print.thres.cex = 1.2,
     col = "red", print.thres.col = "red")

plot(rfRoc, type = "s", add = TRUE, print.thres = c(.5),
     print.thres.pch = 1, legacy.axes = TRUE, print.thres.pattern = "",
     print.thres.cex = 1.2,
     col = "green", print.thres.col = "green")
```

```
plot(gbmRoc, type = "s", add = TRUE, print.thres = c(.5),
     print.thres.pch = 10, legacy.axes = TRUE, print.thres.pattern = "",
     print.thres.cex = 1.2,
     col = "blue", print.thres.col = "blue")


legend(0.2, 0.5, cex = 0.8,
       c("Single Tree", "Bagged Tree", "Random Forest", "Boosted Tree"),
       lwd = c(1, 1, 1, 1),
       col = c("black", "red", "green", "blue"),
       pch = c(16, 3, 1, 10))
```
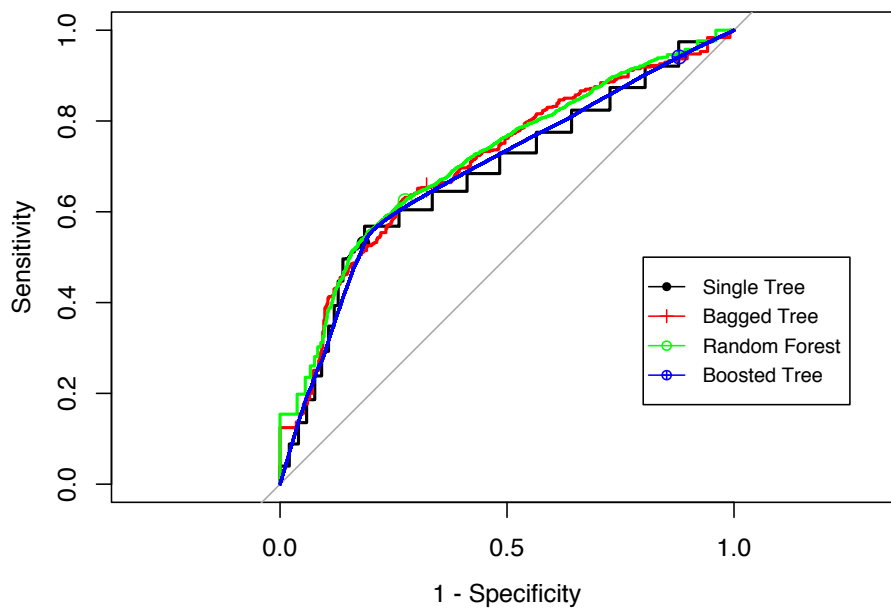

Since the data here doesn't have many variables, we don't see a significant difference among the models. But you can still see those ensemble methods are better than a single tree. In most of the real applications, ensemble methods perform much better. Random forest and boosting trees can be a baseline model. Before exploring different models, you can quickly run a random forest to see the performance and then try to improve that performance. If the performance you got from the random forest is not too much

better than guessing, you should consider collecting more data or reviewing the problem to frame it a different way instead of trying other models. Because it usually means the current data is not enough to solve the problem.

11.6.3 Boosting as Additive Model

This section illustrates how boosting is a forward stagewise additive model that minimizes exponential loss (Friedman J and R, 2000). Many seemingly different models can be represented as a **basis expansion model**. Recall the classifier obtained by the AdaBoost.M1 algorithm:

$$G(x) = sign(\Sigma_{m=1}^{M}\alpha_m G_m(x))$$

The above expression fits in the framework of basis expansion which is as following:

$$f(x) = \Sigma_{m=1}^{M}\beta_m b(x, \gamma_m) \tag{11.2}$$

where β_m, $m = 1, ..., M$ is expansion coefficient and $b(x, \gamma) \in \mathbb{R}$ is basis function. Many of the learning methods fit into this additive framework. Hastie et al discuss basis expansion in detail in Chapter 5 of the book (Hastie T, 2008) and cover the additive expansion for different learning techniques (Hastie T, 2008), such as single-hidden-layer neural networks (Chapter 11) and MARS (Section 9.4). These models are fit by minimizing a designated loss function averaged over the training data:

$$\min_{\{\beta_m,\gamma_m\}_i^M} \Sigma_{i=1}^{N} L\left(y_i, \Sigma_{m=1}^{M}\beta_m b(x_i; \gamma_m)\right) \tag{11.3}$$

Different models have different basis function $b(x_i; \gamma_m)$. You can also customize loss function $L(\cdot)$, such as squared-error loss, or likelihood-based loss. Optimizing the loss function across the whole training set is usually computation costly no matter the choice of basis and loss. The good news is that the problem can be simplified as fitting a single basis function.

$$\min_{\beta,\gamma} = \Sigma_{i=1}^N L(y_i, \beta b(x_i; \gamma))$$

The forward stagewise algorithm approximates the optimal solution of equation (11.3). It adds new basis functions to the expansion without adjusting the previous ones. The forward stagewise additive algorithm is as following (section 10.2, (Hastie T, 2008)):

1. Initialize $f_0(x) = 0$
2. **for** $m = 1, \dots, M$ **do**

   - compute
     $$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{argmin} \Sigma_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$

   - set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

At iteration m, it will search for the optimal $b(x; \gamma_m)$ and β_m based on the previous basis function $f_{m-1}(x)$. And then add the new basis $b(x; \gamma_m)\beta_m$ to the previous basis function to get a new basis function $f_m(x)$ without changing any parameters from previous steps. Assume we use squared-error loss:

$$L(y, f(x)) = (y - f(x))^2$$

Then we have:

$$L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2$$

where $y_i - f_{m-1}(x_i)$ is the residual for sample i based on the previous model. That is to say, it is fitting the new basis using the residual of the previous step.

However, the squared-error loss is generally not a good choice. For a regression problem, it is susceptible to outliers. It doesn't fit the classification problem since the response is categorical. Hence we often consider other loss functions.

Now let is go back to the AdaBoost.M1 algorithm. It is actually

a special case of the above forward stagewise model when the loss function is:

$$L(y, f(x)) = exp(-yf(x))$$

In AdaBoost.M1, the basis function is the classifier from each iteration $G_m(x) \in \{-1, 1\}$. If we use the exponential loss, the optimization problem is

$$
\begin{aligned}
(\beta_m, G_m) &= \underset{\beta,G}{argmin} \Sigma_{i=1}^N exp[-y_i(f_{m-1}(x_i) + \beta G(x_i))] \\
&= \underset{\beta,G}{argmin} \Sigma_{i=1}^N exp[-y_i \beta G(x_i)] \cdot exp[-y_i f_{m-1}(x_i)] \\
&= \underset{\beta,G}{argmin} \Sigma_{i=1}^N w_i^m exp[-y_i \beta G(x_i)]
\end{aligned}
$$
(11.4)

where $w_i^m = exp[-y_i f_{m-1}(x_i)]$. It does not depend on β and $G(x)$. So we can consider it as the weight for each sample. Since the weight is related to $f_{m-1}(x_i)$, it changes each iteration. We can further decompose equation (11.4):

$$
\begin{aligned}
(\beta_m, G_m) &= \underset{\beta,G}{argmin} \Sigma_{i=1}^N w_i^m exp[-y_i \beta G(x_i)] \\
&= \underset{\beta,G}{argmin} \Sigma_{i=1}^N \left\{ w_i^m e^{-\beta} I(y_i = G(x)) + w_i^m e^{\beta} I(y_i \neq G(x)) \right\} \\
&= \underset{\beta,G}{argmin} \Sigma_{i=1}^N \left\{ w_i^m e^{-\beta} [1 - I(y_i \neq G(x))] + w_i^m e^{\beta} I(y_i \neq G(x)) \right\} \\
&= \underset{\beta,G}{argmin} \left\{ (e^{\beta} - e^{-\beta}) \cdot \Sigma_{i=1}^N w_i^m I(y_i \neq G(x_i)) + e^{-\beta} \cdot \Sigma_{i=1}^N w_i^m \right\}
\end{aligned}
$$
(11.5)

when $\beta > 0$, the solution for equation (11.5) is:

$$G_m = \underset{G}{argmin} \Sigma_{i=1}^N w_i^m I(y_i \neq G(x))$$

It is the classifier that minimizes the weighted error. Plug the above G_m into equation (11.5). Take the derivative with respect to β and set it to be 0 to solve the optimal β_m:

$$\beta_m = \frac{1}{2} ln \frac{1 - err_m}{err_m}$$

where

$$err_m = \frac{\Sigma_{i=1}^{N} w_i^m I(y_i \neq G_m(x_i))}{\Sigma_{i=1}^{N} w_i^m}$$

According to the forward stagewise algorithm, the result is updated as:

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$$

We can go ahead and get the weight for the next iteration:

$$
\begin{aligned}
w_i^{m+1} &= exp[-y_i f_m(x_i)] \\
&= exp[-y_i f_{m-1}(x) - y_i \beta_m G_m(x)] \\
&= w_i^m \cdot exp[-\beta_m y_i G_m(x_i)]
\end{aligned}
\tag{11.6}
$$

Since $-y_i G_m(x_i) = 2 \cdot I(y_i \neq G_m(x_i)) - 1$, equation (11.5) can be written as:

$$w_i^{m+1} = w_i^m \cdot exp[\alpha_m I(y_i \neq G_m(x_i))] \cdot exp[-\beta_m]$$

where $\alpha_m = 2\beta_m = log\frac{1-err_m}{err_m}$ is the same with the α_m in AdaBoost.M1 algorithm we showed before. So AdaBoost.M1 is a special case of a forward stagewise additive model using exponential loss. For comparing and selecting different loss functions, you can refer to section 10.5 and 10.6 in (Hastie T, 2008).

12

Deep Learning

12.1 Projection Pursuit Regression

Before moving onto neural networks, let us start with a broader framework, Projection Pursuit Regression (PPR). It has a form of **additive model** of the derived features rather than the inputs themselves. Another widely used algorithm, AdaBoost, also fits an additive model in a base learner.

Assume $\mathbf{X^T} = (X_1, X_2, \ldots, X_p)$ is a vector with p variables. Y is the corresponding response variable. $\boldsymbol{\omega_m}, m = 1, 2, \ldots, M$ is parameter vector with p elements.

$$f(\mathbf{X}) = \sum_{m=1}^{M} g_m(\boldsymbol{\omega_m^T}\mathbf{X})$$

The new feature $\mathbf{V_m} = \boldsymbol{\omega_m^T}\mathbf{X}$ is a linear combination of input variables $\mathbf{X}$. The additive model is based on the new features. Here $\boldsymbol{\omega_m}$ is a unit vector, and the new feature $\mathbf{v_m}$ is actually the projection of $\mathbf{X}$ on $\boldsymbol{\omega_m}$. It projects the p-dimensional independent variable space onto the new M-dimensional feature space. This is similar to the principal component analysis except that the principal component is orthogonal projection but it is not necessarily orthogonal here.

I know it is very abstract. Let's look at some examples. Assume $p = 2$, i.e. there are two variables x_1 and x_2. If $M = 1$, $\boldsymbol{\omega^T} = (\frac{1}{2}, \frac{\sqrt{3}}{2})$, then the corresponding $v = \frac{1}{2}x_1 + \frac{\sqrt{3}}{2}x_2$. Let's try different setings and compare the results:

1. $\mathbf{T} = (\frac{1}{2}, \frac{\sqrt{3}}{2})$, $v = \frac{1}{2}x_1 + \frac{\sqrt{3}}{2}x_2$, $g(v) = \frac{1}{1+e^{-v}}$

2. $\mathbf{T} = (1, 0)$, $v = x_1$, $g(v) = (v+5)sin(\frac{1}{\frac{v}{3}+0.1})$

3. $\mathbf{T} = (0, 1)$, $v = x_2$, $g(v) = e^{\frac{v^2}{5}}$

4. $\mathbf{T} = (1, 0)$, $v = x_1$, $g(v) = (v+0.1)sin(\frac{1}{\frac{v}{3}+0.1})$

Here is how you can simulate the data and plot it using R:

```r
# use plot3D package to generate 3D plot
library(plot3D)
# get x1 and x2 note here x1 and x2 need to be matrix if you check the
# two objects, you will find: columns in x1 are identical rows in x2
# are identical mesh() is funciton from plot3D package you may need to
# think a little here
M <- mesh(seq(-13.2, 13.2, length.out = 50), seq(-37.4, 37.4, length.out = 50))
x1 <- M$x
x2 <- M$y
## setting 1 map X using w to get v
v <- (1/2) * x1 + (sqrt(3)/2) * x2
# apply g() on v
g1 <- 1/(1 + exp(-v))
par(mfrow = c(2, 2), mar = c(0, 0, 1, 0))
surf3D(x1, x2, g1, colvar = g1, border = "black", colkey = FALSE, box = FALSE,
    main = "Setting 1")
## setting 2
v <- x1
g2 <- (v + 5) * sin(1/(v/3 + 0.1))
surf3D(x1, x2, g2, colvar = g2, border = "black", colkey = FALSE, box = FALSE,
    main = "Setting 2")
## setting 3
v <- x2
g3 <- exp(v^2/5)
surf3D(x1, x2, g3, colvar = g3, border = "black", colkey = FALSE, box = FALSE,
    main = "Setting 3")
## setting 4
```
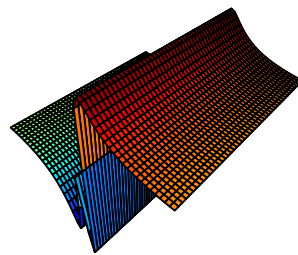
```
v <- x1
g4 <- (v + 0.1) * sin(1/(v/3 + 0.1))
surf3D(x1, x2, g4, colvar = g4, border = "black", colkey = FALSE, box = FALSE,
    main = "Setting 4")
```
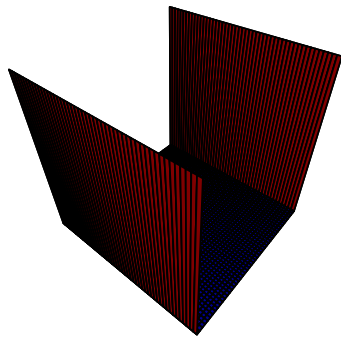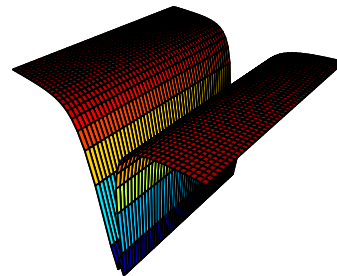
Setting 1
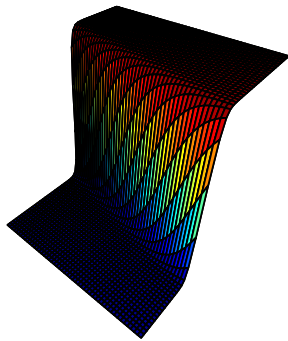
Setting 2

Setting 3

Setting 4

You can see that this framework is very flexible. In essence, it is to do a non-linear transformation of the linear combination. You can use this way to capture varies of relationships. For example,$x_1 x_2$ can be written as $\frac{(x_1+x_2)^2-(x_1-x_2)^2}{4}$, where $M = 2$. All the higher order factors of x_1 and x_2 can be represented similarly. If M is large enough, this framework can approximate any continuous function on $\mathbb{R}^p$. So the model family covers a board area, but with a price. That is the interpretability. Because the number of parameters increases with M and the mode is nested.

PPR in 1981 was a new idea then which lead to the debut of

the neural network model. The basic technical idea behind deep learning has been around for decades. However, why did they take off in recent years? Here are some main drivers behind the rise.

First, thanks to the digitalization where lots of human activities are now in the digital realm and captured as data. So in the last 10 year, for many problems, we went from having a relatively small amount of data to accumulating a large amount of data. The traditional learning algorithms, like Logistic Regression, Support Vector Machine, Random Forest cannot effectively take advantage of such big data. Second, the increasing computation power enables us to train a large neural network either on a CPU or GPU using big data. The scale of data and computation ability lead to much progress, but tremendous algorithmic innovation is also an important driver. Many of the innovations are about speeding up the optimization of neural network. One of the examples is to use ReLU as intermediate layer activation function instead of the previous sigmoid function. The change has made the optimization process much faster because the previous sigmoid function suffers from vanishing gradient. We will talk more about that in the following sections. Here we just want to show an example of the impact of algorithmic innovation.

12.2 Standard Neural Network

12.2.1 Logistic Regression as Neural Network

Let's look at logistic regression from the lens of neural network. For a binary classification problem, for example spam classifier, given m samples $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$, we need to use the input feature $x^{(i)}$ (they may be the frequency of various words such as "money", special characters like dollar signs, and the use of capital letters in the message etc.) to predict the output $y^{(i)}$ (if it is a spam email). Assume that for each sample i, there are n_x input features. Then we have:

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(m)} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n_x}^{(1)} & x_{n_x}^{(2)} & \cdots & x_{n_x}^{(m)} \end{bmatrix} \in \mathbb{R}^{n_x \times m} \qquad (12.1)$$

$$y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}] \in \mathbb{R}^{1 \times m}$$

To predict if sample i is a spam email, we first get the inactivated **neuro** $z^{(i)}$ by a linear transformation of the input $x^{(i)}$, which is $z^{(i)} = w^T x^{(i)} + b$. Then we apply a function to "activate" the neuro $z^{(i)}$ and we call it "activation function". In logistic regression, the activation function is sigmoid function and the "activated" $z^{(i)}$ is the prediction:

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$. The following figure summarizes the process:
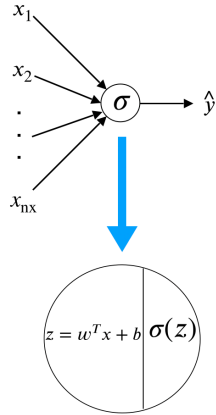
There are two types of layers. The last layer connects directly to the output. All the rest are *intermediate layers*. Depending on your definition, we call it "0-layer neural network" where the layer count only considers *intermediate layers*. To train the model, you need a cost function which is defined as equation (12.2).

$$J(w, b) = \frac{1}{m} \Sigma_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \Sigma_{i=1}^m \{ -y^{(i)} log(\hat{y}^{(i)}) - (1 - y^{(i)}) log(1 - \hat{y}^{(i)}) \}$$
$$(12.2)$$

The general approach to minimize $J(w, b)$ is by gradient descent, also known as *back-propagation*. In logistic regression, it is easy to calculate the gradient w.r.t the parameters (w, b) using the chain rule for differentiation. The optimization process is a forward and backward sweep over the network. Let's look at the gradient descent for logistic regression across m sample. The non-vectorized process is as follows.
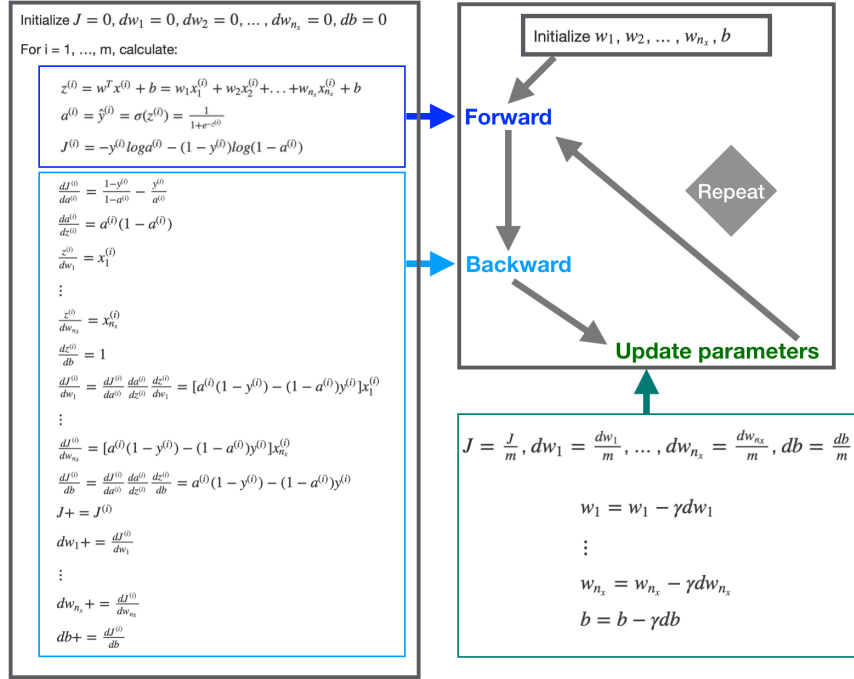
First initialize w_1, w_2, ... , w_{n_x}, and b. Then plug in the initialized value to the forward and backward propagation. The forward propagation takes the current weights and calculates the prediction $\hat{h}^{(i)}$ and cost $J^{(i)}$. The backward propagation calculates the gradient descent for the parameters. After iterating through all m samples, you can calculate gradient descent for the parameters. Then update the parameter by:
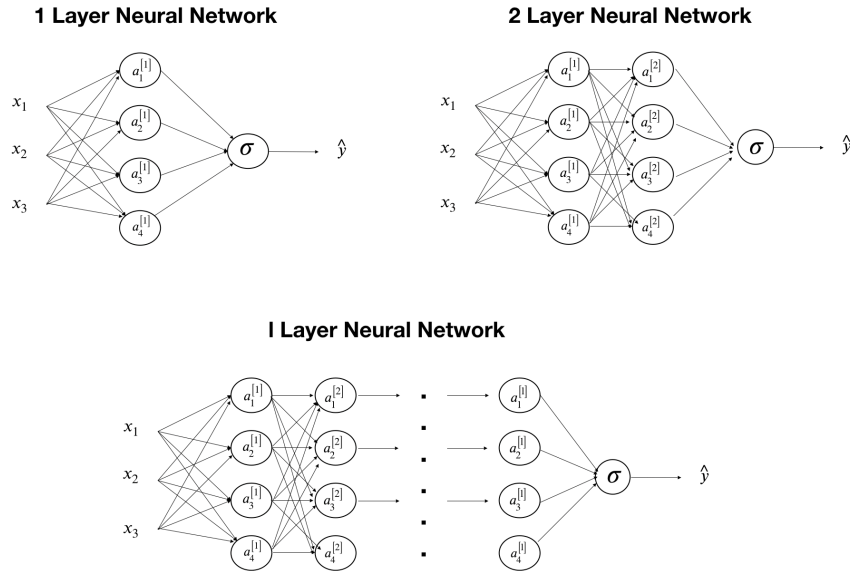
$$w := w - \gamma \frac{\partial J}{\partial w}$$

$$b := b - \gamma \frac{\partial J}{\partial b}$$

Repeat the progapation process using the updated parameter until the cost J stabilizes.

12.2.2 Deep Neural Network

Before people coined the term *deep learning*, a neural network refers to *single hidden layer network*. Neural networks with more than one layers are called *deep learning*.

12.2.3 Activation Function

You need to choose activation function in the hidden layers. Logistic regression only uses sigmoid activation function. But there are other choices. Intermediate layers usually use different activation function than the last layer. Let's first look at some of the common options other than sigmoid.

12.3   Convolutional Neural Network

12.4   Recurrent Neural Network

12.4.1

A

R code for data simulation

A.1 Customer Data for Clothing Company

The simulation is not very straightforward and we will break it into three parts: 1. Define data structure: variable names, variable distribution, customer segment names, segment size 1. Variable distribution parameters: mean and variance 1. Iterate across segments and variables. Simulate data according to specific parameters assigned

By organizing code this way, it makes easy for us to change specific parts of the simulation. For example, if we want to change the distribution of one variable, we can just change the corresponding part of the code.

Here is code to define data structure:

```
# set a random number seed to make the process repeatable
set.seed(12345)
# define the number of observations
ncust<-1000
# create a data frmae for simulated data
seg_dat<-data.frame(id=as.factor(c(1:ncust)))
# assign the variable names
vars<-c("age","gender","income","house","store_exp","online_exp","store_trans","online_trans")
# assign distribution for each variable
vartype<-c("norm","binom","norm","binom","norm","norm","pois","pois")
# names of 4 segments
group_name<-c("Price","Conspicuous","Quality","Style")
```

```
# size of each segments
group_size<-c(250,200,200,350)
```

The next step is to define variable distribution parameters. There
are 4 segments of customers and 8 parameters. Different segments
correspond to different parameters. Let's store the parameters in
a 4×8 matrix:

```
# matrix for mean
mus <- matrix( c(
  # Price
  60, 0.5, 120000,0.9, 500,200,5,2,
  # Conspicuous
  40, 0.7, 200000,0.9, 5000,5000,10,10,
  # Quality
  36, 0.5, 70000, 0.4, 300, 2000,2,15,
  # Style
  25, 0.2, 90000, 0.2, 200, 2000,2,20), ncol=length(vars), byrow=TRUE)
```

```
# matrix for variance
sds<- matrix( c(
  # Price
  3,NA,8000,NA,100,50,NA,NA,
  # Conspicuous
  5,NA,50000,NA,1000,1500,NA,NA,
  # Quality
  7,NA,10000,NA,50,200,NA,NA,
  # Style
  2,NA,5000,NA,10,500,NA,NA), ncol=length(vars), byrow=TRUE)
```

Now we are ready to simulate data using the parameters defined
above:

```r
# simulate non-survey data
sim.dat<-NULL
set.seed(2016)
# loop on customer segment (i)
 for (i in seq_along(group_name)){

   # add this line in order to moniter the process
   cat (i, group_name[i],"\n")

  # create an empty matrix to store relevent data
  seg<-data.frame(matrix(NA,nrow=group_size[i], ncol=length(vars)))

  # Simulate data within segment i
  for (j in seq_along(vars)){

    # loop on every variable (j)
    if (vartype[j]=="norm"){
      # simulate normal distribution
      seg[,j]<-rnorm(group_size[i], mean=mus[i,j], sd=sds[i,j])
    } else if (vartype[j]=="pois") {
      # simulate poisson distribution
      seg[,j]<-rpois(group_size[i], lambda=mus[i,j])
    } else if (vartype[j]=="binom"){
      # simulate binomial distribution
      seg[,j]<-rbinom(group_size[i],size=1,prob=mus[i,j])
    } else{
      # if the distribution name is not one of the above, stop and return a message
      stop ("Don't have type:",vartype[j])
    }
  }
  sim.dat<-rbind(sim.dat,seg)
 }
```

Now let's edit the data we just simulated a little by adding tags to 0/1 binomial variables:

```r
# assign variable names
names(sim.dat)<-vars
# assign factor levels to segment variable
sim.dat$segment<-factor(rep(group_name,times=group_size))
# recode gender and house variable
sim.dat$gender<-factor(sim.dat$gender, labels=c("Female","Male"))
sim.dat$house<-factor(sim.dat$house, labels=c("No","Yes"))
# store_trans and online_trans are at least 1
sim.dat$store_trans<-sim.dat$store_trans+1
sim.dat$online_trans<-sim.dat$online_trans+1
# age is integer
sim.dat$age<-floor(sim.dat$age)
```

In the real world, the data always includes some noise such as missing, wrong imputation. So we will add some noise to the data:

```r
# add missing values
idxm <- as.logical(rbinom(ncust, size=1, prob=sim.dat$age/200))
sim.dat$income[idxm]<-NA
# add wrong imputations and outliers
set.seed(123)
idx<-sample(1:ncust,5)
sim.dat$age[idx[1]]<-300
sim.dat$store_exp[idx[2]]<- -500
sim.dat$store_exp[idx[3:5]]<-c(50000,30000,30000)
```

So far we have created part of the data. You can check it using `summary(sim.dat)`. Next, we will move on to simulate survey data.

```r
# number of survey questions
nq<-10
# mean matrix for different segments
mus2 <- matrix( c(
```

```r
  # Price
 5,2,1,3,1,4,1,4,2,4,
  # Conspicuous
 1,4,5,4,4,4,4,1,4,2,
  # Quality
 5,2,3,4,3,2,4,2,3,3,
  # Style
 3,1,1,2,4,1,5,3,4,2), ncol=nq, byrow=TRUE)

# assume the variance is 0.2 for all
sd2<-0.2
sim.dat2<-NULL
set.seed(1000)
# loop for customer segment (i)
for (i in seq_along(group_name)){
  # the following line is used for checking the progress
  # cat (i, group_name[i],"\n")
  # create an empty data frame to store data
  seg<-data.frame(matrix(NA,nrow=group_size[i], ncol=nq))
  # simulate data within segment
  for (j in 1:nq){
    # simulate normal distribution
    res<-rnorm(group_size[i], mean=mus2[i,j], sd=sd2)
    # set upper and lower limit
    res[res>5]<-5
    res[res<1]<-1
    # convert continuous values to discrete integers
    seg[,j]<-floor(res)
  }
  sim.dat2<-rbind(sim.dat2,seg)
}

names(sim.dat2)<-paste("Q",1:10,sep="")
sim.dat<-cbind(sim.dat,sim.dat2)
sim.dat$segment<-factor(rep(group_name,times=group_size))
```

A.2   Customer Satisfaction Survey Data from Airline Company

```
# Create a matrix of factor loadings
# This pattern is called bifactor because it has a general factor for separate components.
# For example, "Ease of making reservation" has general factor loading 0.33, specific factor l
# The outcome variables are formed as combinations of these general and specific factors
loadings <- matrix(c (
  # Ticketing
  .33, .58, .00, .00,  # Ease of making reservation
  .35, .55, .00, .00,  # Availability of preferred seats
  .30, .52, .00, .00,  # Variety of flight options
  .40, .50, .00, .00,  # Ticket prices
  # Aircraft
  .50, .00, .55, .00,  # Seat comfort
  .41, .00, .51, .00,  # Roominess of seat area
  .45, .00, .57, .00,  # Availability of Overhead
  .32, .00, .54, .00,  # Cleanliness of aircraft
  # Service
  .35, .00, .00, .50,  # Courtesy of flight attendant
  .38, .00, .00, .57,  # Friendliness
  .60, .00, .00, .50,  # Helpfulness
  .52, .00, .00, .58,  # Food and drinks
  # General
  .43, .10, .30, .30,  # Overall satisfaction
  .35, .50, .40, .20,  # Purchase again
  .25, .50, .50, .20), # Willingness to recommend
  nrow=15,ncol=4, byrow=TRUE)

# Matrix multiplication produces the correlation matrix except for the diagonal
cor_matrix<-loadings %*% t(loadings)
# Diagonal set to ones
diag(cor_matrix)<-1
```

```r
# use the mvtnorm package to randomly generate a data set with a given correlation pattern

library(mvtnorm)
# mean vectors of the 3 airline companies
mu1=c(5,6,5,6, 7,8,6,7, 5,5,5,5, 6,6,6)
mu2=c(3,3,2,3, 5,4,5,6, 8,8,8,8, 3,3,3)
mu3=c(2,2,2,2, 8,8,8,8, 8,8,8,8, 8,8,8)

# set random seed
set.seed(123456)
# respondent ID
resp.id <- 1:1000

library(MASS)
rating1 <- mvrnorm(length(resp.id),
                    mu=mu1,
                     Sigma=cor_matrix)
rating2 <- mvrnorm(length(resp.id),
                   mu=mu2,
                   Sigma=cor_matrix)
rating3 <- mvrnorm(length(resp.id),
                   mu=mu3,
                   Sigma=cor_matrix)


# truncates scale to be between 1 and 9
rating1[rating1>9]<-9
rating1[rating1<1]<-1
rating2[rating2>9]<-9
rating2[rating2<1]<-1
rating3[rating3>9]<-9
rating3[rating3<1]<-1

# Round to single digit
rating1<-data.frame(round(rating1,0))
```

```r
rating2<-data.frame(round(rating2,0))
rating3<-data.frame(round(rating3,0))
rating1$ID<-resp.id
rating2$ID<-resp.id
rating3$ID<-resp.id
rating1$Airline<-rep("AirlineCo.1",length(resp.id))
rating2$Airline<-rep("AirlineCo.2",length(resp.id))
rating3$Airline<-rep("AirlineCo.3",length(resp.id))
rating<-rbind(rating1,rating2,rating3)

# assign names to the variables in the data frame
names(rating)<-c(
  "Easy_Reservation",
  "Preferred_Seats",
  "Flight_Options",
  "Ticket_Prices",
  "Seat_Comfort",
  "Seat_Roominess",
  "Overhead_Storage",
  "Clean_Aircraft",
  "Courtesy",
  "Friendliness",
  "Helpfulness",
  "Service",
  "Satisfaction",
  "Fly_Again",
  "Recommend",
  "ID",
  "Airline")
```

A.3   Swine Disease Breakout Data

```
# sim1_da1.csv the 1st simulated data similar sim1_da2 and sim1_da3
# sim1.csv simulated data, the first simulation dummy.sim1.csv dummy
# variables for the first simulated data with all the baseline code for
# simulation

# setwd(dirname(file.choose())) library(grplasso)

nf <- 800
for (j in 1:20) {
    set.seed(19870 + j)
    x <- c("A", "B", "C")
    sim.da1 <- NULL
    for (i in 1:nf) {
        # sample(x, 120, replace=TRUE)->sam
        sim.da1 <- rbind(sim.da1, sample(x, 120, replace = TRUE))
    }

    sim.da1 <- data.frame(sim.da1)
    col <- paste("Q", 1:120, sep = "")
    row <- paste("Farm", 1:nf, sep = "")
    colnames(sim.da1) <- col
    rownames(sim.da1) <- row

    # use class.ind() function in nnet package to encode dummy variables
    library(nnet)
    dummy.sim1 <- NULL
    for (k in 1:ncol(sim.da1)) {
        tmp = class.ind(sim.da1[, k])
        colnames(tmp) = paste(col[k], colnames(tmp))
        dummy.sim1 = cbind(dummy.sim1, tmp)
    }
```

```r
dummy.sim1 <- data.frame(dummy.sim1)

# set 'C' as the baseline delete baseline dummy variable

base.idx <- 3 * c(1:120)
dummy1 <- dummy.sim1[, -base.idx]

# simulate independent variable for different values of r simulate
# based on one value of r each time r=0.1, get the link function
s1 <- c(rep(c(1/10, 0, -1/10), 40), rep(c(1/10, 0, 0), 40), rep(c(0,
    0, 0), 40))
link1 <- as.matrix(dummy.sim1) %*% s1 - 40/3/10

# r=0.25
# c(rep(c(1/4,0,-1/4),40),rep(c(1/4,0,0),40),rep(c(0,0,0),40))->s1
# as.matrix(dummy.sim1)%*%s1-40/3/4->link1

# r=0.5
# c(rep(c(1/2,0,-1/2),40),rep(c(1/2,0,0),40),rep(c(0,0,0),40))->s1
# as.matrix(dummy.sim1)%*%s1-40/3/2->link1

# r=1 c(rep(c(1,0,-1),40),rep(c(1,0,0),40),rep(c(0,0,0),40))->s1
# as.matrix(dummy.sim1)%*%s1-40/3->link1

# r=2 c(rep(c(2,0,-2),40),rep(c(2,0,0),40),rep(c(0,0,0),40))->s1
# as.matrix(dummy.sim1)%*%s1-40/3/0.5->link1


# calculate the outbreak probability
hp1 <- exp(link1)/(exp(link1) + 1)

# based on the probability hp1, simulate response variable: res
res <- rep(9, nf)
for (i in 1:nf) {
    res[i] <- sample(c(1, 0), 1, prob = c(hp1[i], 1 - hp1[i]))
}
```

```r
    # da1 with response variable, without group indicator da2 without
    # response variable, with group indicator da3 without response
    # variable, without group indicator

    dummy1$y <- res
    da1 <- dummy1
    y <- da1$y
    ind <- NULL
    for (i in 1:120) {
        ind <- c(ind, rep(i, 2))
    }

    da2 <- rbind(da1[, 1:240], ind)
    da3 <- da1[, 1:240]

    # save simulated data
    write.csv(da1, paste("sim", j, "_da", 1, ".csv", sep = ""), row.names = F)
    write.csv(da2, paste("sim", j, "_da", 2, ".csv", sep = ""), row.names = F)
    write.csv(da3, paste("sim", j, "_da", 3, ".csv", sep = ""), row.names = F)
    write.csv(sim.da1, paste("sim", j, ".csv", sep = ""), row.names = F)
    write.csv(dummy.sim1, paste("dummy.sim", j, ".csv", sep = ""), row.names = F)
}
```

Bibliography

A, A. and J, A. A. (1984). On the existence of the maximum likelihood estimates in logistic regression models. *Biometrika*, 71(1):1–10.

B, E. and R, T. (1986). Bootstrap methods for standard errors, con- fidence intervals, and other measures of statistical accuracy. *Statistical Science*, pages 54–75.

Box G, C. D. (1964). An analysis of transformations. *Journal of the Royal Statistical Society*, pages 211–252.

Breiman, L. (1998). Arcing classifiers. *The Annals of Statistics*, 26:123–140.

Breiman, L. (2001a). Random forests. *Machine Learning*, 45:5–32.

Breiman, L. (2001b). Statistical modeling: The two cultures. *Statistical Science*, 16(3):199231.

Cestnik, B. and Bratko, I. (1991). Estimating probabilities in tree pruning. *EWSL*, pages 138–150.

Chun, H. and Keleş, S. (2010). Sparse partial least squares regression for simultaneous dimension reduction and variable selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(1):3–25.

de Waal, T., Pannekoek, J., and Scholtus, S. (2011). *Handbook of Statistical Data Editing and Imputation*. John Wiley and Sons.

Dudoit S, F. J. and T, S. (2002). Comparison of discrimination meth- ods for the classification of tumors using gene expression data. *Journal of the American Statistical Association*, 97(457):77–87.

E, B. and R, K. (1999). An empirical comparison of voting classifica- tion algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–142.

Efron, B. (1983). Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, pages 316–331.

Efron, B. and Tibshirani, R. (1986). Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, pages 54–75.

Efron, B. and Tibshirani, R. (1997). Improvements on cross-validation: The 632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560.

et. al, B. A. (2000). Tissue classification with gene expression profiles. *Journal of Computational Biology*, 7(3):559–583.

et. al, B. J. (2006). Aggregate features and adaboost for music classification. *Machine Learning*, 65:473–484.

et. al, L. B. (1984). *Classification and regression trees*. Number ISBN 978-0412048418. Chapman and Hall/CRC.

F. Espoito, D. M. and Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491.

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139.

Friedman J, H. T. and R, T. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 38:337–374.

Gareth James, Daniela Witten, T. H. and Tibshirani, R. (2015). *An Introduction to Statistical Learning*. Springer, 6th edition.

Geladi P, K. B. (1986). Partial least squares regression: A tutorial. *Analytica Chimica Acta*, (185):1–17.

Hastie T, Tibshirani R, F. J. (2008). *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer, 2nd edition.

Hoerl, A. and Kennard, R. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Hyndman, R. and Athanasopoulos, G. (2013). *Forecasting: Principles and Practice*, volume Section 2/5. OTect: Melbourne, Australia.

Iglewicz, B. and Hoaglin, D. (1993). How to detect and handle outliers. *The ASQC Basic References in Quality Control: Statistical Techniques*, 16.

Jolliffe, I. (2002). *Principla component analysis.* Springer, 2nd edition.

Kuhn, M. and Johnston, K. (2013). *Applied Predictive Modeling.* Springer.

L, B. (1966a). Bagging predictors. *Machine Learning*, 24(2):123–140.

L, V. (1984). A theory of the learnable. *Communications of the ACM*, 27:1134–1142.

L Meier, S. v. d. G. and Buhlmann, P. (2008). The group lasso for logistic regression. *J. R. Stat. Soc. Ser. B Stat. Methodol*, 70:53–71.

Line Clemmensen, Trevor Hastie, D. W. and Ersbøll, B. (2011). Sparse discriminant analysis. *Technometrics*, 53(4):406–413.

M, K. and L, V. (1989). Cryptographic limitations on learning boolean formulae and finite automata. In *In "Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing.*

M, S. T. and F, P. (2007b). Handling missing values when applying classification models. *Journal of Machine Learning Research*, 8:1625–1657.

Mulaik, S. (2009). *Foundations of factor analysis.* Chapman Hall/CRC, 2nd edition.

Patel, N. and Upadhyay, S. (2012). Study of various decision tree pruning methods with their empirical comparison in weka. *International Journal of Computer Applications*, 60(12).

Quinlan, J. (1999). Simplifying decision trees. *International Journal of Human-Computer Studies*, 61(2).

R, T. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B (Methodological)*, 58(1):267–288.

Serneels S, Nolf ED, E. P. (2006). Spatial sign preprocessing: A simple way to impart moderate robustness to multivariate estimators. *Journal of Chemical Information and Modeling*, 46(3):1402–1409.

T, D. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40:139–158.

T, H. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:340–354.

Varmuza K, H. P. and K, F. (2003). Boosting applied to classification of mass spectral data. *Journal of Data Science*, 1(391–404).

Wedderburn, R. W. M. (1976). On the existence and uniqueness of the maximum likelihood estimates for certain generalized linear models. *Biometrika*, 63:27–32.

Willett, P. (2004). Dissimilarity-based algorithms for selecting structurally diverse sets of compounds. *Journal of Computational Biology*, 6(3-4)(doi:10.1089/106652799318382):447–457.

Y, A. and D, G. (1997). Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588.

Y. Kim, J. K. and Kim, Y. (2006). Blockwise sparse regression. *Statist. Sin*, 16:375–390.

YFR, S. (1999). Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103.

Yuan, M. and Lin, Y. (2007). Model selection and estimation in regression with grouped variables. *J. R. Stat. Soc. Ser. B Stat. Methodol*, 68:49–67.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *arXiv :1611.03530*.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B,*, 67(2):301–320.