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 co-founder 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 confusion. 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

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

2. Acceleration of developments in computers and display devices
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]"

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

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 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

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

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.

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.

(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

[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.

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. nonparametric 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.

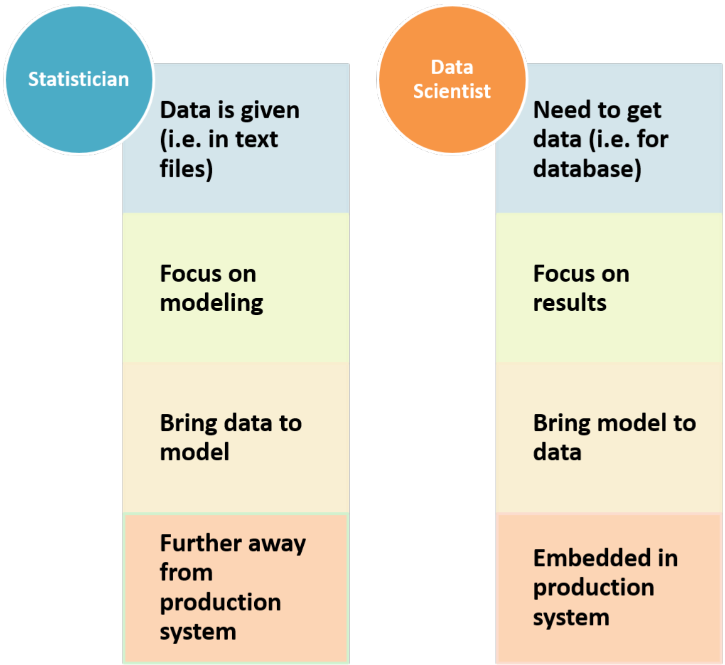
- What will be the temperature tomorrow?
- What is the projected net income for the next season?
- How much inventory should we have?

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 common backgrounds, but there are also some 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 particular power of statistics is that statistician can fit model and make an inference based on limited data. It is quite common that once the data is given and cleaned, the majority of the work is developed different models around the data. Today, data is relatively abundant, and modeling is just part of the overall effort. The focus is to deliver actionable results. Different from statisticians, data scientists, sometimes need to fit model on the cloud instead of reading data in since the data size is too large. 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 Where Does Data Science Team Fits?

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 as mentioned above. To leverage the big data, many companies have established new data science organizations. Companies have gone through different paths to create their data science and machine learning organizations. There are three major formats of data science teams:

(1) independent of any current organizations and the team

report directly to senior leadership;

(2) within each business unit and the team report to business unit leaders;

(3) within in the traditional IT organizations and the team report to IT leaders.

Companies are different in many aspects, but in general, the most efficient option to mine big data is a team of data scientist independent of business units and IT organizations. The independence enables the data science team to collaborate across business units and IT organizations more efficiently and the independence also provides flexibility and potential to solve corporate level strategic big data problems. For each business units, there are many business unit specific data science related problems and embedding data scientist within each business units is also an efficient way to solve business unit specific data science problems. The full cycle of data science projects from data to decision (i.e. Data -> Information -> Knowledge -> Insight -> Decision) is relatively difficult to achieve if the data science team is part of traditional IT organizations.

2.3 Beyond Data and Analytics

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

- a business owner or leader to identify opportunities in business value; program managers to ensure each data science project fit into the overall technical program development;
- data owners and computation resource and infrastructure owners from 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;
- project managers to coordinate all parties to set periodical tasks so that the project meets the preset milestones and delivery results;
- multiple rounds of discussion of resource allocation (i.e. who will 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 will interact with people at various levels ranging from senior leaders who are setting the corporate strategies to front-line employees who are doing the daily work. A data scientist needs to have the capability to view the problem from 10,000 feet above 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.

2.4 Data Scientist as a Leader

During the entire process of data science project defining, planning, executing and implementation, the data scientist lead needs to be involved in every step to ensure the business problem is defined correctly and the business value and success metric are evaluated reasonably. Corporates are investing heavily in data science and machine learning with a very high expectation of big return. There are too many opportunities to introduce unrealistic goal and business impact for a particular data science project. The leading data scientist need to be the leader in these discussions to define the goal backed by data and analytics. Many data science projects over promise in deliverables and too optimistic on the timeline and these projects eventually fail by not delivering the preset business impact within the timeline. As the data scientist in the team, we need to identify these issues early in the stage and communicate to 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 computation infrastructure team to understand the computation resources (i.e. hardware and software) available for the data science project.

2.5 Three Pillars of Knowledge

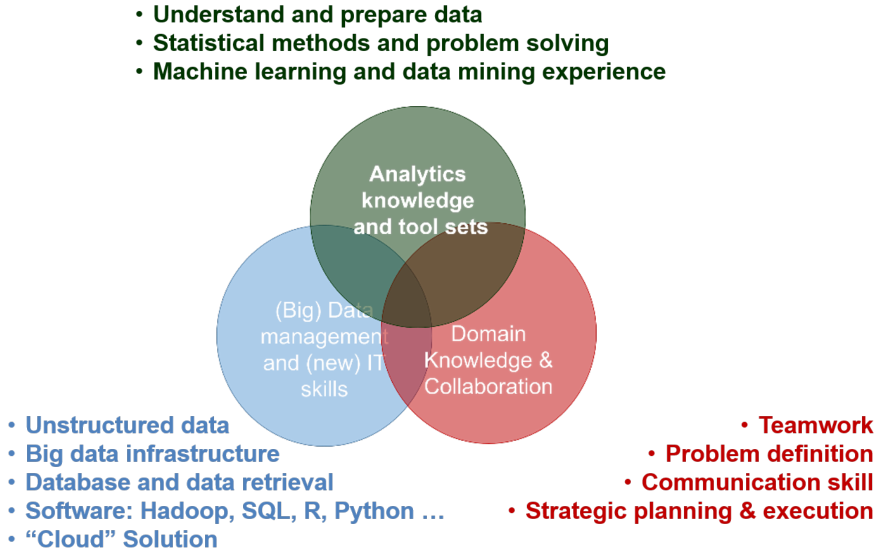
The following picture summarizes the needed three pillars of knowledge to be a successful data scientist.

(1) 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) A successful data scientist needs some domain knowledge to understand 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) The last pillar is about computation environment and model implementation in 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, this barrier is getting easier

for a statistician to overcome and we will discuss in more detail in next chapter.


• 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

2.6 Common Pitfalls of Data Science Projects

Data science projects are usually complicated, and many of these data science projects eventually fail due to various reasons. We will briefly discuss a few common pitfalls in data science projects and how to avoid them.

- **Solve the wrong problem:** data science project usually starts with a very vague description and a few rounds of detailed discussion with all stakeholders involved are needed to define the busses problem. There will be lots of opportunities to introduce misalignment when mapping the business problem into specific data science methods. Especially when the quality and availability of the data are not as good as what is expected at the first place. If not well-communicated during the project, the final data science solution may not be the right one to solve the business problem. As the data scientist (sometimes the only data scientist) in the room, we must understand the business problem thoroughly and communicate regularly to business partners especially there is a change of status to make sure everyone is aligned with the progress and final deliverables.

- **Over promise on business value:** business leaders usually have high expectation on data science projects and the goal of business value and deliverables sometimes are set unrealistic and eventually beyond the scope of available data and computation resource. As the data scientist (sometimes the only data scientist) in the room, we must have our voice heard based on fact (i.e. data, analytics, and resources) instead of wishful thinking. Backed with fact-based evidence, it is easier to communicate what is a realistic goal for the entire team.

- **Too optimistic about the timeline:** there are lots of uncertainties in data science projects such as the data source availability and data quality, computation hardware and software, resource availability in the business team, implementation team and IT department, as well as project direction change which may delay the final delivery date. To have a better-estimated timeline, get as much detail as possible for all the needed tasks and estimated each task individually and reach out to each team member to confirm their availability. Most importantly, communicate with the entire team if there are blocking factors for the project in a prompt way such that everyone aware of the situation and potential impact on the timeline.

- **Too optimistic about data availability and quality:** the most important asset in data science project is data. Even though we are at the big data age, often there is not enough relevant data for the data science projects. The data quality is also a general problem for data science projects. A thorough data availability and quality check are needed at the beginning of the data science project to estimate the needed effort to obtain data as well as data cleaning.

- **Model cannot be scaled:** be careful if you use a subset of data to fit the model and then scale it to the entire dataset. When developing the model using a smaller dataset, we must keep in mind how much computation resources needed for the whole dataset. With limited computation resource, it is important to maximize the computation time in production to a reasonable level based on the business application when fits the model with a sample dataset.

- **Take too long to fail:** data science projects usually are trying to push the boundary of current applications to new territory, people do not expect all data science projects to be successful. Fail fast is good practice such that we can quickly find a better way to solve the problem. A data scientist needs to have an open mindset to not stuck with one idea or one approach for a long time to avoid taking too long to fail.

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:

```r
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:

```r
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 i^{th} farm is generated from a $Bernoulli(1, p_i)$ distribution with p_i being a function of the question answers:

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

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 g^{th} 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:

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

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

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

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

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

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:

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

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("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Dat
# 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

4

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.

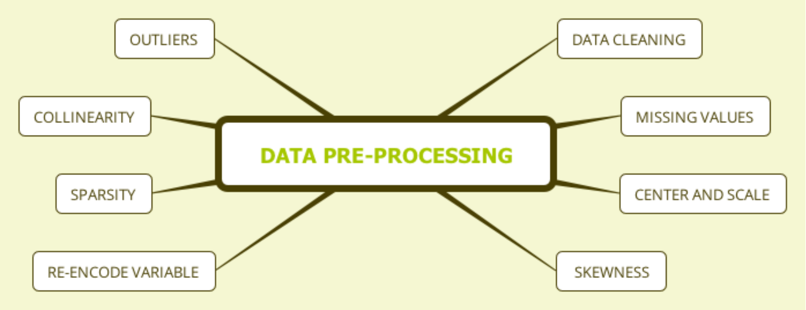


FIGURE 4.1: Data Pre-processing Outline

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:

```
source("https://raw.githubusercontent.com/happyrabbit/CE_JSM2017/master/Rcode/00-course-setup.
```

4.1 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("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/
summary(sim.dat)
```

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 (store_trans) make sense too. Things need 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 will 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","income")))
```

```
      age            income
 Min.   :16.00   Min.   : 41776
 1st Qu.:25.00   1st Qu.: 84930
 Median :33.00   Median : 93023
 Mean   :36.64   Mean   :109968
 3rd Qu.:45.00   3rd Qu.:121535
 Max.   :69.00   Max.   :317476
 NA's   :1
```

Now we will deal with the missing values in the data.

4.2 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).

4.2.1 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:446   Min.   : 41776   No :372   Min.   :  155.8
 1st Qu.:25.00   Male  :326   1st Qu.: 84930   Yes:400   1st Qu.:  202.7
 Median :33.00                Median : 93023             Median :  293.0
 Mean   :36.63                Mean   :109968             Mean   : 1265.4
 3rd Qu.:45.00                3rd Qu.:121535             3rd Qu.:  540.2
 Max.   :69.00                Max.   :317476             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()` function, 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])
```

4.2.2 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)
```

```
Error in `[.data.frame`(old, , non_missing_cols, drop = FALSE) :
  undefined columns selected
```

Now we get an error saying "undefined columns selected." It is because `sim.dat` has non-numeric variables. The `preProcess()` in the first line will automatically ignore non-numeric columns, so there is no error. However, there is a problem when using `predict()` to get the result. Removing those variable will solve the problem.

```
# find factor columns
imp<-preProcess(sim.dat,method="knnImpute",k=5)
idx<-which(lapply(sim.dat,class)=="factor")
demo_imp<-predict(imp,sim.dat[,-idx])
summary(demo_imp[,1:3])
```

`lapply(data,class)` can return a list of column class. Here the data frame is `sim.dat`, and the following code will give the list of column class:

```
# only show the first three elements
lapply(sim.dat,class)[1:3]
```

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)
idx<-which(lapply(temp,class)=="factor")
```

```
demo_imp<-predict(imp,temp[,-idx])
```

```
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:

```
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.

4.2.3 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
```

4.3 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()` in package `caret` 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:

```r
summary(transformed)
```

Sometimes you only need to scale the variable. For example, if the model adds a penalty to the parameter estimates (such as L_2 penalty is ridge regression and L_1 penalty in LASSO), the variables need to have a similar scale to ensure a fair variable selection. I am a heavy user of this kind of penalty-based model in my work, and I used the following quantile transformation:

$$x^*_{ij} = \frac{x_{ij} - quantile(x_{.j}, 0.01)}{quantile(x_{.j} - 0.99) - quantile(x_{-j}, 0.01)}$$

The reason to use 99% and 1% quantile instead of maximum and minimum values is to resist the impact of outliers.

It is easy to write a function to do it:

```r
qscale<-function(dat){
  for (i in 1:ncol(dat)){
    up<-quantile(dat[,i],0.99)
    low<-quantile(dat[,i],0.01)
    diff<-up-low
    dat[,i]<-(dat[,i]-low)/diff
  }
  return(dat)
}
```

In order to illustrate, let's apply it to some variables from ‘demo_imp2:

```r
demo_imp3<-qscale(subset(demo_imp2,select=c("income","store_exp","online_exp")))
summary(demo_imp3)
```

After transformation, most of the variables are between 0-1.

4.4 Resolve Skewness

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

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

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

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

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

```
# skew, fig.cap='Shewed Distribution', out.width='80%', fig.asp=.75, # fig.align='center'
# 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")
```

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)
```

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")))
```

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

```
# bc, fig.cap='Box-Cox Transformation', out.width='80%', fig.asp=.75, # fig.align='center'
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 the transformation, the `stroe_trans` is skewed right. `BoxCoxTrans ()` 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))
```

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

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.

4.5 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`:

```
# scm, fig.cap='Use basic visualization to check outliers', out.width='80%', # fig.asp=.75, fi
# select numerical non-survey data
sdat<-subset(sim.dat,select=c("age","income","store_exp","online_exp","store_trans","online_tr
# 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`:

```r
# 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))
```

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 i^{th} 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:

```
# sst, fig.cap='Spatial sign transformation', out.width='80%', fig.asp=.75,
# fig.align='center'}
# 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.

4.6 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.

```
# corp, fig.cap='Correlation Matrix', out.width='80%', fig.asp=.75,
# fig.align='center'}
# select non-survey numerical variables
sdat<-subset(sim.dat,select=c("age","income","store_exp","online_exp","store_trans","online_tr
# 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 -0.85) 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=.75))
```

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 0.75.

```
# 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 0.75. 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.

4.7 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`:

```
# 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)
```

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

4.8 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)
```

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

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

`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))
```

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.

4.9 Python Computing

Environmental Setup

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import numpy as np
import scipy as sp
import pandas as pd
import math
```

```
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import StandardScaler

from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
```

4.9.1 Data Cleaning

5

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.

5.1 Data Wrangling Using R

5.1.1 Read and write data

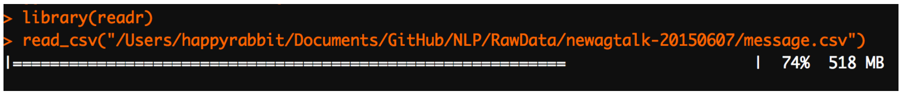
5.1.1.1 `readr`

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:

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

3. `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.
4. 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:

```
library(readr)
sim.dat <- read_csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/
head(sim.dat)
```

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:

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

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

```
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)
```

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)
```

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_name
print(dat)
```

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)
```

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)
```

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)
```

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:

```
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 RPostgreSQL, 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

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

5.1.1.2 `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

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("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/
# subset the first two rows
sim.dat[1:2,]
# subset the first two rows and column 3 and 5
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:

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

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

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

There is no gold standard to choose a specific function to manipulate 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 aggregation, 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 burden 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:

```
# If you haven't install it, use the code to instal
# install.packages("data.table")
# load packagw
library(data.table)
```

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

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

Calculate mean for counts of online transactions:

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

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]
```

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

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

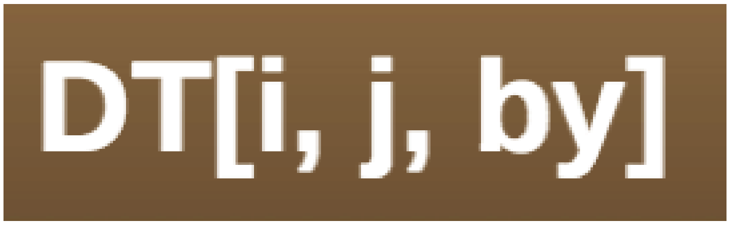
Assign column names for aggregated variables:

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

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

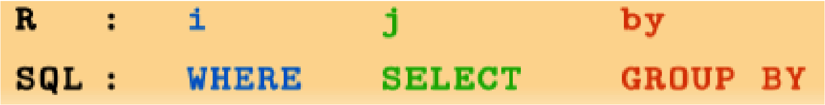
- General setting of `data.table`

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



DT[i, j, by]

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.



R	:	i	j	by
SQL	:	WHERE	SELECT	GROUP BY

Let's review our previous code:

```r
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:

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

is equal to SQL

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

R code

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

is equal to 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

```
# select rows with age<20 and income > 80000
dt[age < 20 & income > 80000]
# select the first two rows
dt[1:2]
```

- 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)
```

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

```
# 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 `.()`:

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

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

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

Delete columns using `-` or `!`:

```
# 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]
```

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

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

Order table:

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

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("https://raw.githubusercontent.com/happyrabbit/DataScientistR/mast
```

```
  user  system elapsed
  4.313   0.027   4.340
```

```
system.time(topic<-readr::read_csv("https://raw.githubusercontent.com/happyrabbit/DataScientis
```

```
   user  system elapsed
  0.267   0.008   0.274
```

```
system.time(topic<-data.table::fread("https://raw.githubusercontent.com/happyrabbit/DataScient
```

```
   user  system elapsed 
  0.217   0.005   0.221 
```

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.

5.1.2 Summarize data

5.1.2.1 `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:

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

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

You can also apply other functions:

```
ma <- matrix(c(1:4, 1, 6:8), nrow = 2)
ma
apply(ma, 1, table)  #--> a list of length 2
apply(ma, 1, stats::quantile) # 5 x n matrix with rownames
```

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.

```
# Read data
sim.dat<-read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/Se
# 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:

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

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()`:

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

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

There are some odd values in store expense. The minimum value is -500 which is a wrong imputation 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:

```r
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)
```

5.1.3 `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.

```
library(dplyr)
```

```
library(dplyr)
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:

```
library(magrittr)
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 "`%>%`":

```
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.

```
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.

```
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))
```

Now, let's peel the onion in order.

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:

```
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))
```

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`:

```
# 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")))
```

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

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

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

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

```
# 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")
```

```
# 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.

5.1.4 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.

5.1.4.1 `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

```
library(reshape2)
(mdat <- melt(sdat, measure.vars=c("store_exp","online_exp"),
              variable.name = "Channel",
              value.name = "Expense"))
```

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)
```

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`" .

5.1.4.2 `tidyr` package

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

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

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

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

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)
```

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

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

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="_")
```

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.

5.1.5 Data Wrangling Using Python

6

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.

6.1 Systematic Error and Random Error

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

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

where ϵ 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:

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

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.

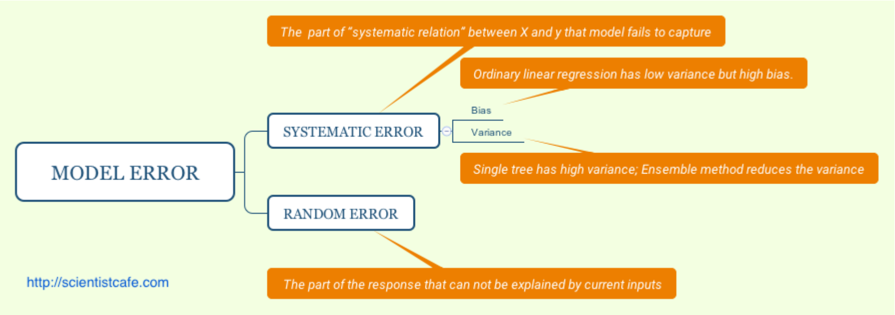

The part of "systematic relation" between X and y that model fails to capture
Ordinary linear regression has low variance but high bias.
Bias
Variance
SYSTEMATIC ERROR
MODEL ERROR
Single tree has high variance; Ensemble method reduces the variance
RANDOM ERROR
http://scientistcafe.com
The part of the response that can not be explained by current inputs

FIGURE 6.1: Types of Model Error

The systematic error can be further decomposed as:

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

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.

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.

```
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:

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

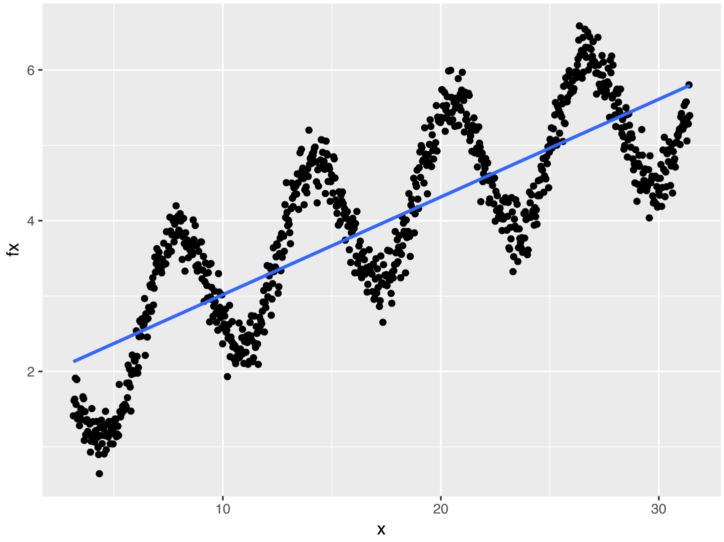


FIGURE 6.2: High bias model

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. 6.2). 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 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)
```

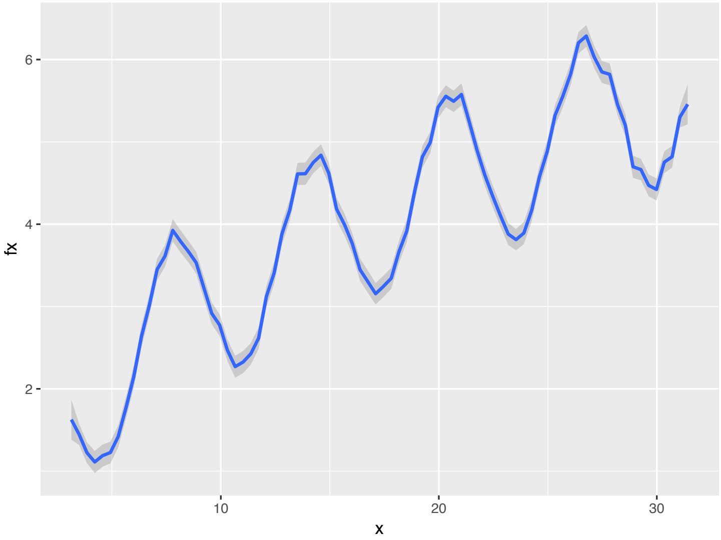
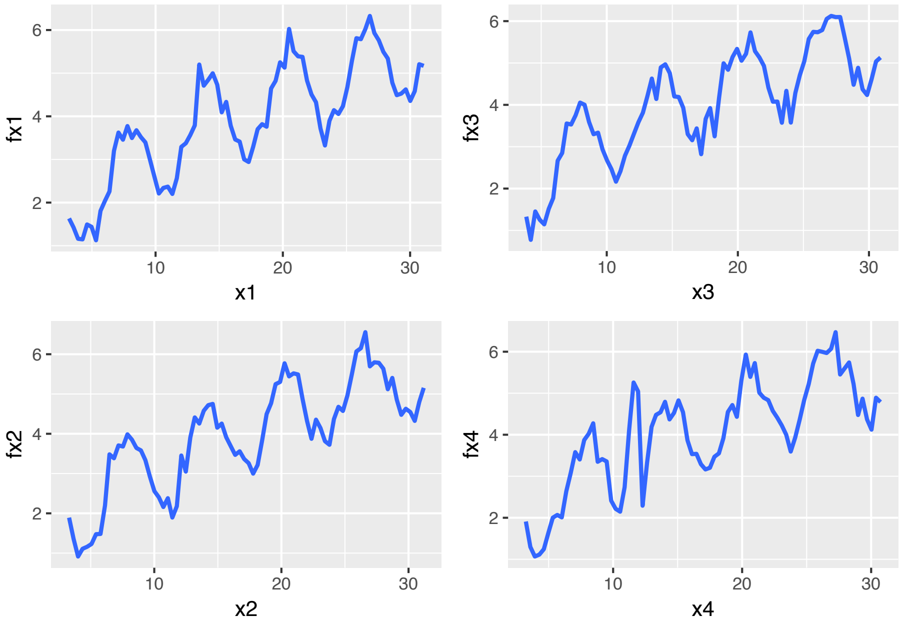


FIGURE 6.3: High variance model

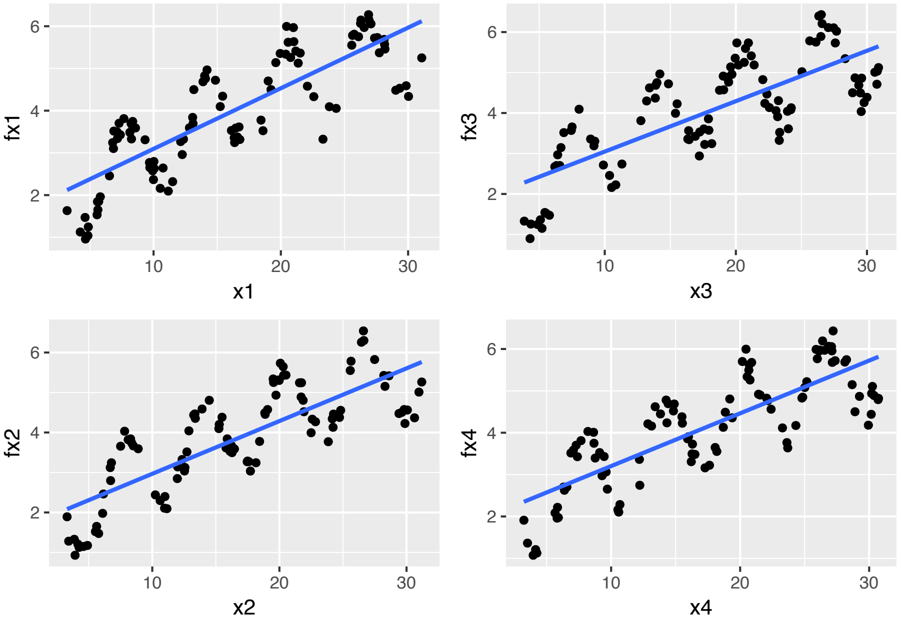
The resulting plot (Fig. 6.3) indicates the smoothing method fit the data much better so it has a much smaller bias. However, this method has a high variance. If we simulate different subsets of the sample, the result curve will change significantly:

```
# 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 $(Var(\hat{f}(\mathbf{X})))$ **increases** and the bias $(Bias(\hat{f}(\mathbf{X})))$ **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 $Bias(\hat{f}(\mathbf{X}))$ decreases is faster than $Var(\hat{f}(\mathbf{X}))$, so the MSE decreases. However, to some degree, higher flexibility has little effect on $Bias(\hat{f}(\mathbf{X}))$ but $Var(\hat{f}(\mathbf{X}))$ increases significantly, so the MSE increases.

6.1.1 Measurement Error in the Response

The measurement error in the response contributes to the random error (ϵ). 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:

```
# load data
sim.dat <- read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/
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`:

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

```
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.

```
# 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 response 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)
}
```

```r
# 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")
```

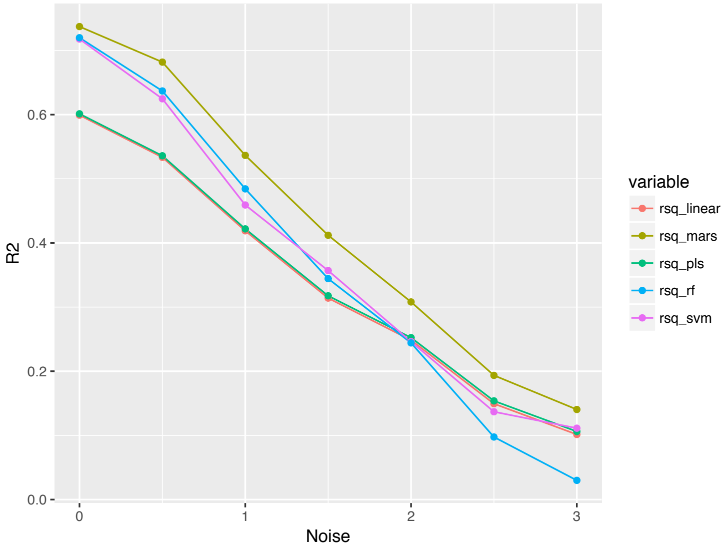


FIGURE 6.4: Test set R^2 profiles for income models when measurement system noise increases. `rsq_linear`: linear regression, `rsq_pls`: Partial Least Square, `rsq_mars`: Multiple Adaptive Regression Spline Regression, `rsq_svm`: Support Vector Machine `rsq_rf`: Random Forest

Fig. 6.4 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 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 simplest model in the case of similar performance. Model evaluation and selection represent the career "maturity" of a data scientist.

6.1.2 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) toonline_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.

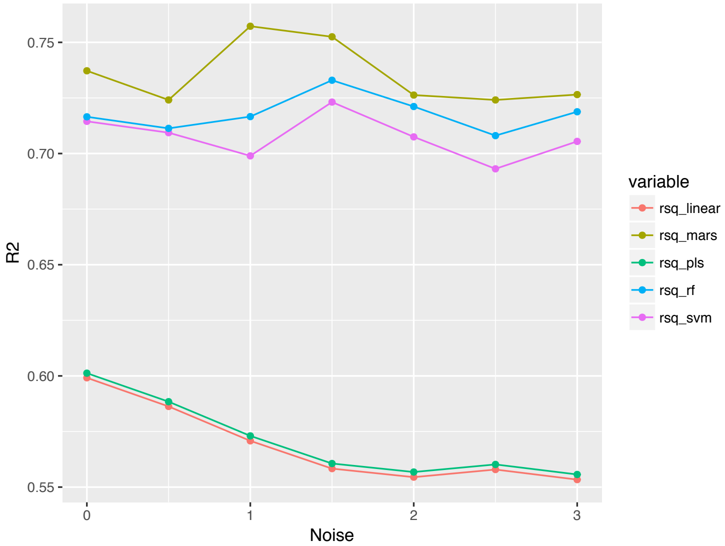


FIGURE 6.5: Test set R^2 profiles for income models when noise in `online_exp` increases. `rsq_linear` : linear regression, `rsq_pls` : Partial Least Square, `rsq_mars`: Multiple Adaptive Regression Spline Regression, `rsq_svm`: Support Vector Machine `rsq_rf`: Random Forest

Comparing Fig. 6.5 and Fig. 6.4, 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. 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.

6.2 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.

6.2.1 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 resampling 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

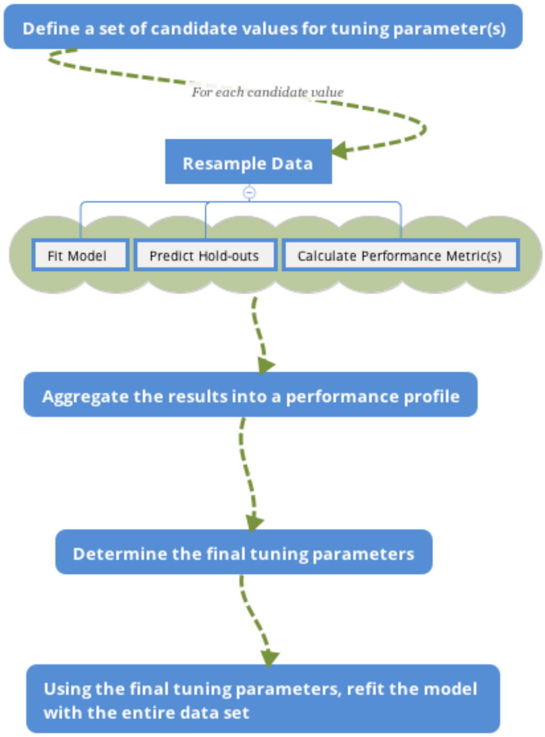

Define a set of candidate values for tuning parameter(s)
For each candidate value
Resample Data
Fit Model
Predict Hold-outs
Calculate Performance Metric(s)
Aggregate the results into a performance profile
Determine the final tuning parameters
Using the final tuning parameters, refit the model with the entire data set

FIGURE 6.6: Parameter Tuning Process

here: (1) sample size; (2) computation intensity. If the sample size is large enough which is the most common situation according to 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 `segment`) 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.

```
# 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.

```r
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`:

```r
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`:

```r
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`:

```r
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)
```

The points from maximum dissimilarity sampling are far away from the initial samples (Fig. 6.7, 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.

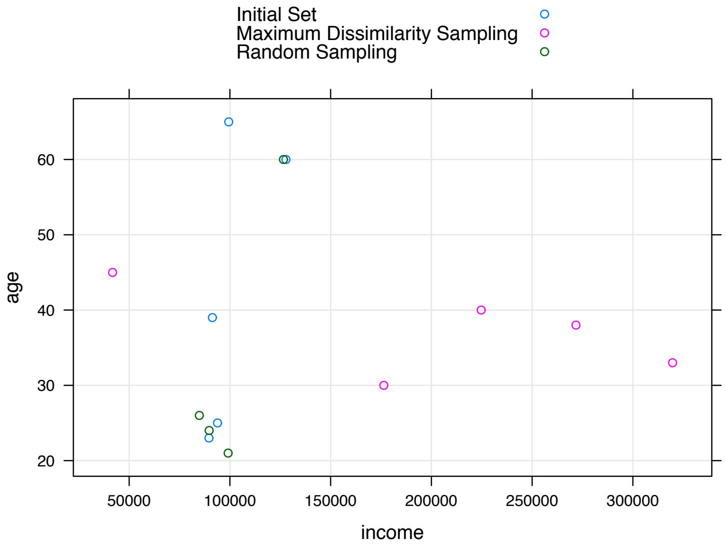


FIGURE 6.7: Compare Maximum Dissimilarity Sampling with Random Sampling

- 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.

```
# 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)))
```

Fig. 6.8 shows 100 simulated time series observation. The goal is to make sure both training and test set to cover the whole period.

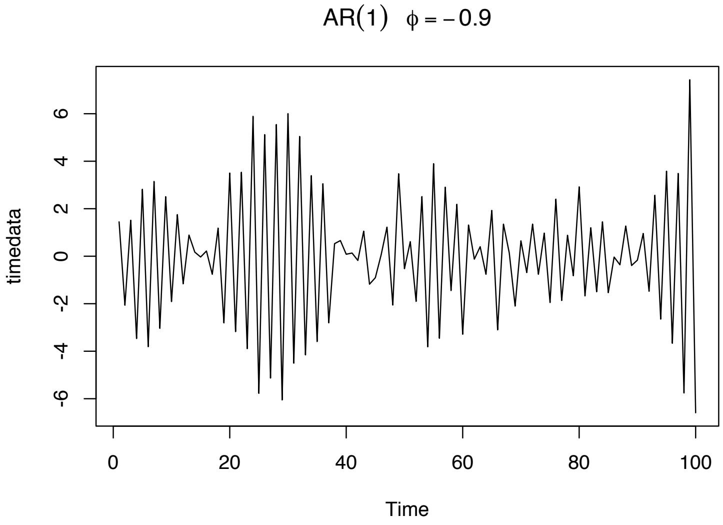


FIGURE 6.8: Divide data according to time

```
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 `fixedWindow`, 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.

6.2.2 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.

6.2.2.1 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 κ^{th} 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 $\kappa = 1k$

• Use data other than fold $\kappa$ to train the model $\hat{f}^{-\kappa}(X)$
• Apply $\hat{f}^{-\kappa}(X)$ to predict fold $\kappa$ to get $\hat{f}^{-\kappa}(x_i^\kappa)$

3. Aggregate the results
        $Er\hat{r}or = \frac{1}{N}\Sigma_{\kappa=1}^k \Sigma_{x_i^\kappa} L(y_i^\kappa, \hat{f}^{-\kappa}(x_i^\kappa))$
```

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.

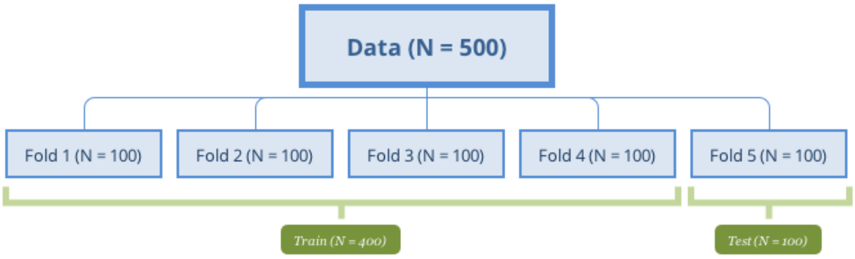


FIGURE 6.9: 5-fold cross-validation

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 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.

6.2.2.2 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.

6.2.2.3 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:

$$(0.632\text{»}\mathit{original\ bootstrap\ estimate})+(0.368\text{»}\mathit{apparent\ error\ rate})$$

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.

7

Measuring Performance

In order to compare different models, we need a way to measure model performance. There are different metrics to measure model performance. Sometimes, it is better to look at models through multiple lens. In this chapter, we will introduce some of the most common measurement metrics.

7.1 Regression Model Performance

7.2 Classification Model Performance

8

Feature Engineering

8.1 Feature Construction

8.2 Feature Extraction

8.3 Feature Selection

8.3.1 Filter Method

8.3.2 Wrapper Method

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

10.1 Ridge Regression

10.2 LASSO

10.3 Elastic Net

10.4 LASSO Generalized Linear Model

11

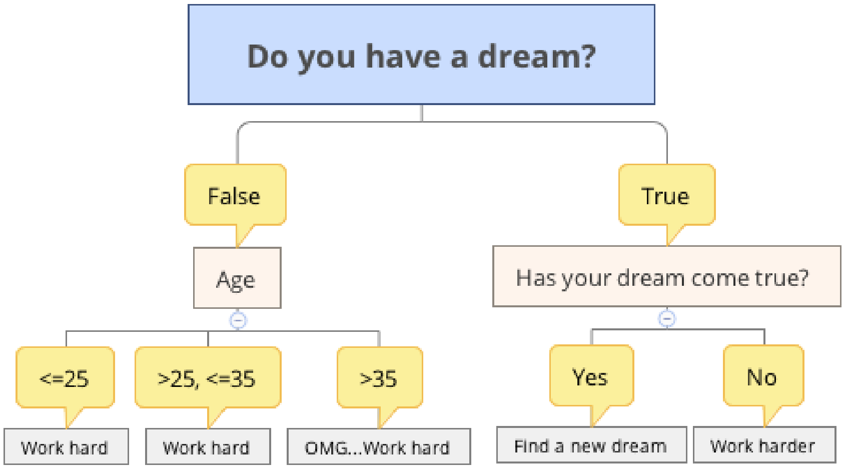
Tree-Based Methods

The tree-based models can be used for regression and classification. They 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 preprocessing 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

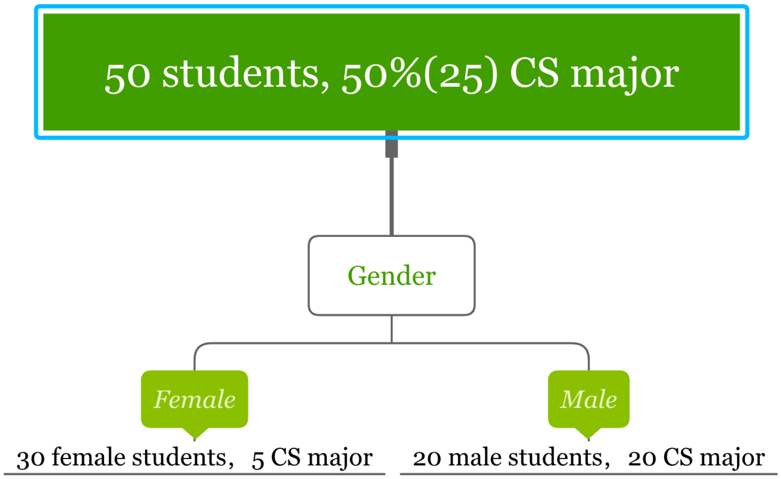
The splitting criteria used by the regression tree and the classification tree are different. Like the regression tree, the goal of the classification tree is to divide the data into smaller, more homogeneous groups. Homogeneity means that most of the samples at each node are from one class. The original 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.


50 students, 50%(25) CS major
Gender
Female
Male
30 female students, 5 CS major
20 male students, 20 CS major

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" $= 0 \times 1 + 1 \times 0 = 0$

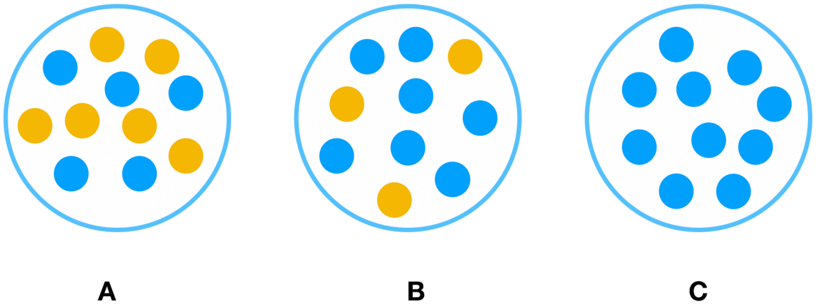
The Gini impurity for the node "Gender" is the following weighted average of the above two scores:

$$\frac{3}{5} \times \frac{5}{18} + \frac{2}{5} \times 0 = \frac{1}{6}$$

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 lower purity.



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.

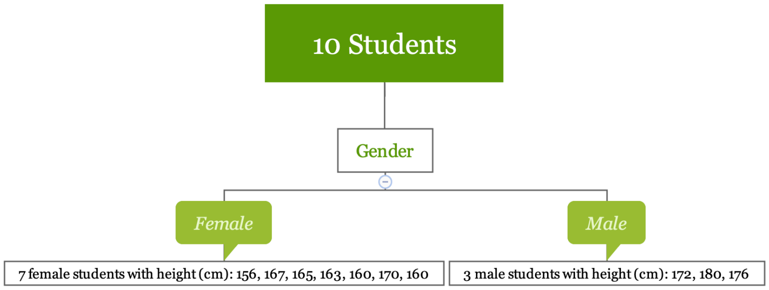
Sum of Square Error (SSE)

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:

$$SSE = \Sigma_{i\in S_1}(y_i - \bar{y}_1)^2 + \Sigma_{i\in S_2}(y_i - \bar{y}_2)^2 \tag{11.1}$$

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:

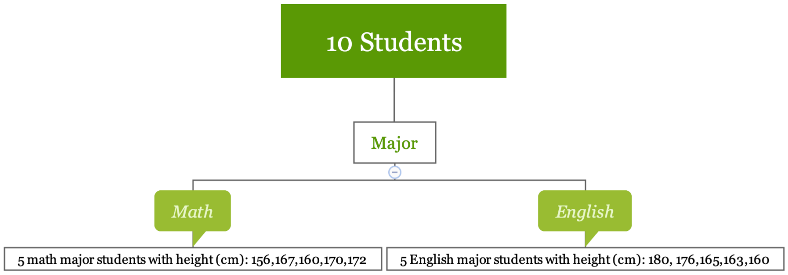

10 Students
Gender
Female
Male
7 female students with height (cm): 156, 167, 165, 163, 160, 170, 160
3 male students with height (cm): 172, 180, 176

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:


10 Students
Major
Math
English
5 math major students with height (cm): 156,167,160,170,172
5 English major students with height (cm): 180, 176,165,163,160

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

11.3 Regression and Decision Tree Basic

11.4 Bagging Tree

11.5 Random Forest

11.6 Gradient Boosted Machine

12

Neural Network

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 $\mathbf{X^T} = (X_1, X_2, \dots, X_p)$ is a vector with p variables. Y is the corresponding response variable. $\omega_{\mathbf{m}}, m = 1, 2, \dots, M$ is parameter vector with p elements.

$$f(\mathbf{X}) = \sum_{m=1}^{M} g_m(\omega_{\mathbf{m}}^{\mathbf{T}}\mathbf{X})$$

The new feature $\mathbf{V_m} = \omega_{\mathbf{m}}^{\mathbf{T}}\mathbf{X}$ is a linear combination of input variables $\mathbf{X}$. The additive model is based on the new features. Here $\omega_{\mathbf{m}}$ is a unit vector, and the new feature $\mathbf{v_m}$ is actually the projection of $\mathbf{X}$ on $\omega_{\mathbf{m}}$. 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.

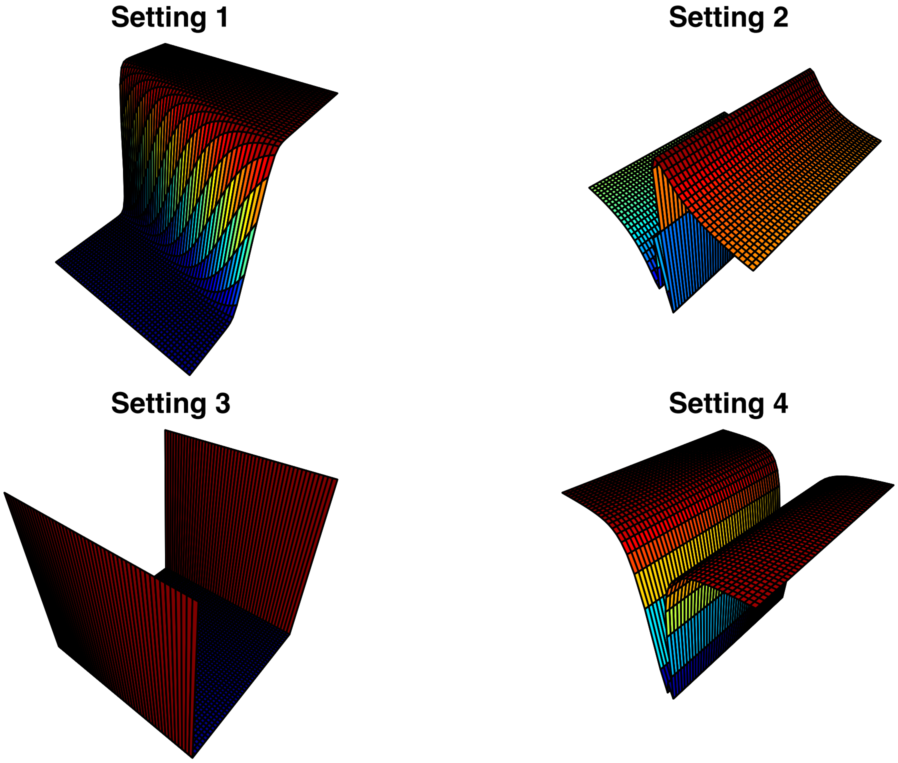
I know it is very abstract. Let's look at some examples. Assume $p = 2$, i.e. there are two variables x_1 and x_2. If $M = 1$, $\omega^{\mathbf{T}} = (\frac{1}{2}, \frac{\sqrt{3}}{2})$, then the corresponding $v = \frac{1}{2}x_1 + \frac{\sqrt{3}}{2}x_2$. Let's try different setings and compare the results:

1. $\omega^{\mathbf{T}} = (\frac{1}{2}, \frac{\sqrt{3}}{2})$, $v = \frac{1}{2}x_1 + \frac{\sqrt{3}}{2}x_2$, $g(v) = \frac{1}{1+e^{-v}}$
2. $\omega^{\mathbf{T}} = (1, 0)$, $v = x_1$, $g(v) = (v+5)sin(\frac{1}{\frac{v}{3}+0.1})$
3. $\omega^{\mathbf{T}} = (0, 1)$, $v = x_2$, $g(v) = e^{\frac{v^2}{5}}$
4. $\omega^{\mathbf{T}} = (1, 0)$, $v = x_1$, $g(v) = (v+0.1)sin(\frac{1}{\frac{v}{3}+0.1})$

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")
```



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_1x_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:

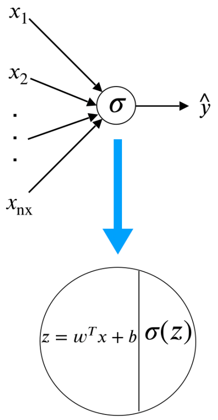
$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(m)} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n_x}^{(1)} & x_{n_x}^{(2)} & \ldots & x_{n_x}^{(m)} \end{bmatrix} \in \mathbb{R}^{n_x \times m} \tag{12.1}$$

$$y = [y^{(1)}, y^{(2)}, \ldots, y^{(m)}] \in \mathbb{R}^{1\times m}$$

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:



To train the model, you need a cost function which is defined as equation (12.2).

$$J(w,b) = \frac{1}{m}\Sigma_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\Sigma_{i=1}^{m}\{ -y^{(i)}log(\hat{y}^{(i)}) - (1-y^{(i)})log(1-\hat{y}^{(i)})\} \tag{12.2}$$

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.

12.2.2 One layer neural network

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

A

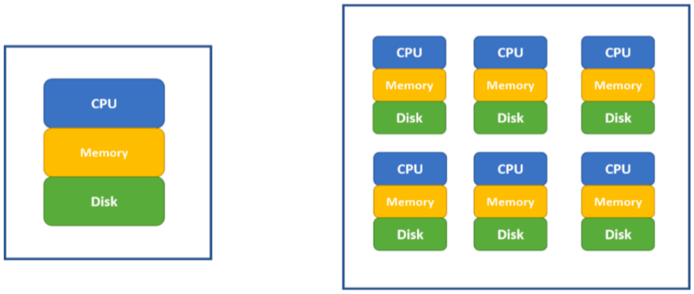
Big Data Cloud Platform

A.1 How Data becomes Science?

Data has been statistician's friend for hundreds of years. Tabulated data are the most familiar format that we use daily. People used to store data on papers, tapes, diskettes, or hard drives. Only recently, with the development of the computer, hardware, software, and algorithms, the volume, variety, and speed of the data suddenly beyond the capacity of a traditional statistician. And data becomes a special science with the very first focus on a fundamental question: with a huge amount of data, how can we store the data and quick access and process the data. In the past a few years, by utilizing commodity hardware and open source software, a big data ecosystem was created for data storage, data retrieval, and parallel computation. Hadoop and Spark have become a popular platform that enables data scientist, statistician, and business analyst to access the data and to build models. Programming skills in the big data platform have been the largest gap for a statistician to become a successful data scientist. However, with the recent wave of cloud computing, this gap is significantly reduced. Many of the technical details have been pushed to the background, and the user interface becomes much easier to learn. Cloud systems also enable quick implementation to the production environment. Now data science is emphasis more on the data itself as well as models and algorithms on top of the data instead of platform and infrastructure.

A.2 Power of Cluster of Computers

We are all familiar with our laptop/desktop computers which contain mainly three components to finish computation with data: (1) Hard disk, (2) Memory, and (3) CPU as shown in Figure 41 left. The data and codes are stored in the hard disk which has certain features such as relatively slow for reading and writes and relatively large capacity of around a few TB in today's market. Memory is relatively fast for reading and writing but relatively small in capacity in the order of a few dozens of GB in today's market. CPU is where all the computation is done.



For statistical software such as R, the amount of data that it can process is limited by the computer's memory. For a typical computer before the year 2000, the memory is less than 1 GB. The memory capacity grows far slower than the availability of the data to analyze. Now it is quite often that we need to analyze data far beyond the capacity of a single computer's memory, especially in an enterprise environment. Meanwhile, the computation time is growing faster than linear to solve the same problem (such as regressions) as the data size increases. Using a cluster of computers become a common way to solve big data problem. In Figure 41 (right), a cluster of computers can be viewed as one powerful machine with total memory, hard disk and CPU equivale to the sum of individual computers. It is common to have thousands of nodes for a cluster.

In the past, to use a cluster of computers, users must write code (such as MPI) to take care of how data is distributed and how the computation is done in a parallel fashion. Luckily with the recent new development, the cloud environment for big data analysis is more user-friendly. As data is typically beyond the size of one hard disk, the dataset itself is stored across different nodes' hard disk (i.e. the Hadoop system mentioned below). When we perform analysis, we can assume the needed data is already distributed across many node's memories in the cluster and algorithm are parallel in nature to leverage corresponding nodes' CPUs to compute (i.e. the Spark system mentioned below).

A.2.1 Evolution of Clustering Computing

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

A.2.2 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 in 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 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 function 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, computation time, and then storing the result onto disk again. So, jobs on top of Hadoop require a lot of disk operation which may slow down the computation process.

A.2.3 Spark

Spark works on top of distributed file system including HDFS with better data and analytics efficiency by leveraging in-memory operations and is more tailored for data processing and analytics. 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 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 R notebook to drive Spark computations.

A.3 Introduction of Cloud Environment

B

Databases and SQL

C

R code for data simulation

C.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
2. Variable distribution parameters: mean and variance
3. 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:

```r
# 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:

```
# 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(
  # 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))
```

C.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
```

```
# 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))
```

```
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")
```

C.3 Swine Disease Breakout Data

```r
# 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)
    }
```

```
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

Box G, C. D. (1964). An analysis of transformations. *Journal of the Royal Statistical Society*, pages 211–252.

Breiman, L. (2001a). Random forests. *Machine Learning*, 45:5–32.

Breiman, L. (2001b). Statistical modeling: The two cultures. *Statistical Science*, 16(3):199231.

de Waal, T., Pannekoek, J., and Scholtus, S. (2011). *Handbook of Statistical Data Editing and Imputation*. John Wiley and Sons.

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, L. B. (1984). *Classification and regression trees*. Number ISBN 978-0412048418. Chapman and Hall/CRC.

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.

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.

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.

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.

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.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *arXiv :1611.03530*.