STA551 E-Pack: Foundations of Data Science

Cheng Peng

West Chester University

Contents

Chapter 1

Introduction

This *E-coursepack* is a self-contained homegrown eBook that contains all topics covered in current STA551 at WCU.

Data science is a young discipline, but it is becoming more and more significant in both academia and industry. It is not a sub-field of any existing well-developed disciplines. Its foundation is built on theories and techniques and drawn from applied statistics and mathematics, machine learning, database and information technology, communication and related domain fields. Proficiency in programming with languages such as Python, SQL, R, etc. is essential to perform any data science tasks.

Data science has an interdisciplinary nature, collectively uses existing tool from different fields, and creates new knowledge and tools to overcome new challenges . There are debates on the definition of data science and validity of naming the emerging interdisciplinary field as **data science**.

1.1 The Origin of Data Science

The idea of expanding the horizon of classical statistics can be traced back to John Tukey's paper *The future of data analysis* (1962), Although not mentioning the term *data science*, Tukey urged statisticians to reduce their focus on statistical theory and engage with the entire data-analysis process: procedures for analyzing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analyzing data: These are foundational components of present new field of **data science**.

John Tukey was a chemist, topologist, educator, consultant, information scientist, researcher, statistician, data analyst, and corporate executive. At the end

of World War II, he began a joint industrial and academic career at Bell Telephone Laboratories and at Princeton University (1946-1985) before he retired.

30 years after the Tukey's prophesied "new science", John Chambers, Tukey's colleague at Bell and a co-inventor of the S language (R and S-Plus are both implementations of S), published an article *Greater or lesser statistics: a choice for future research* in which he thought statistics research was on crossroad and need to rethink whether the field needs to be expanded. Chambers came up with two contrast views of statistics: `Lesser Statistics` (the classical statistics built based on the probability theory) and `Greater Statistics` (the data-driven research which is the part of Tukey's unnamed *New Science*). He thinks the if statisticians remain aloof, others will act. Statistics will lose; in addition, I believe science and society will lose also since statistician's mental attitude at its best provides qualities likely to be missing otherwise.

The term **data science** was first used in scientific community in 1968 by Peter Naur (an astronomer and computer scientist/professor), in an unpublished text in which he defined **data science** to be *'The science of dealing with data, once they have been established, while the relation of the data to what they represent is delegated to other fields and sciences.'*.Apparently, Naur's "data science" was not meant a discipline.

Jeff Wu, an established statistics professor, first used the term **data science** in a lecture given in 1985 as an alternative of statistics. Later, in an inaugural lecture at the University Michigan in 1997, he called for statistics to be renamed data science and statisticians to be renamed data scientists.

William Cleveland, a statistics professor (at Purdue, 2004 - present) and industrial statistician and researcher (at Bell Labs, 1972-2003), defined **data science** as an expanded technical area in the field of statistics. According to Cleveland's proposal, the technical areas of **data science** consist of 6 components with corresponding suggested allocations apply to universities' curriculum.

- (25%) **Multidisciplinary Investigations**: data analysis collaborations in a collection of subject matter areas.
- (20%) **Models and Methods for Data**: statistical models; methods of model building; methods of estimation and distribution based on probabilistic inference.
- (15%) **Computing with Data**: hardware systems; software systems; computational algorithms.
- (15%) **Pedagogy**: curriculum planning and approaches to teaching for elementary school, secondary school, college, graduate school, continuing education, and corporate training.
- (5%) **Tool Evaluation**: surveys of tools in use in practice, surveys of perceived needs for new tools, and studies of the processes for developing new tools.

- (20%) **Theory**: foundations of data science; general approaches to models and methods, computing with data, teaching, and tool evaluation; mathematical investigations of models and methods, computing with data, teaching, and evaluation.

Cleveland's **data science curriculum** proposal has a balanced components of classical statistical and mathematical foundations for data science, computational and software tools and technologies, and domain knowledge. This article as republished in 2014 in Wiley's journal *Statistical Analysis and Data Mining* together with his another article in which he listed the 6 key technical areas of data science are those that have an impact on how a data analyst analyses data in practice:

- Statistical theory;
- Statistical models;
- Statistical and machine learning methods;
- Algorithms for statistical and machine learning methods, and optimization;
- Computational environments for data analysis;
- Live analyses of data where results are judged by the findings, not the methodology and systems that where used.

By that time many universities and colleges including some key universities such as started offered undergraduate majors in data science and master's program in data sciences. These college degree programs in data science are usually houses in mathematics and statistics, computer science and engineering, or business and information systems departments. Some of the key universities expand their statistics departments and rename it as *Statistics and Data Science*. All these movements in academia indicate that the Tukey's prophesied `new science` has been growing to the field of **data science**.

1.2 The Coverage of the First Data Science Course

The technical foundation of data science is built partly on applied statistics and mathematics, computer science and information system, and domain knowledge. Many universities started DS programs at graduate level degrees or certificates from different traditional disciplines using the existing resources and faculty expertise. Because of these limitations, different programs deliver slightly different curricula with different emphases but teach commonalities to build DS foundation.

For the first DS course, different programs also deliver the content in different ways depending on faculty expertise. Most of the current data science foundation courses are methodological emphasis.

We designed this course as a project-based course. We teach the **data science**

process not the isolated models or algorithms.

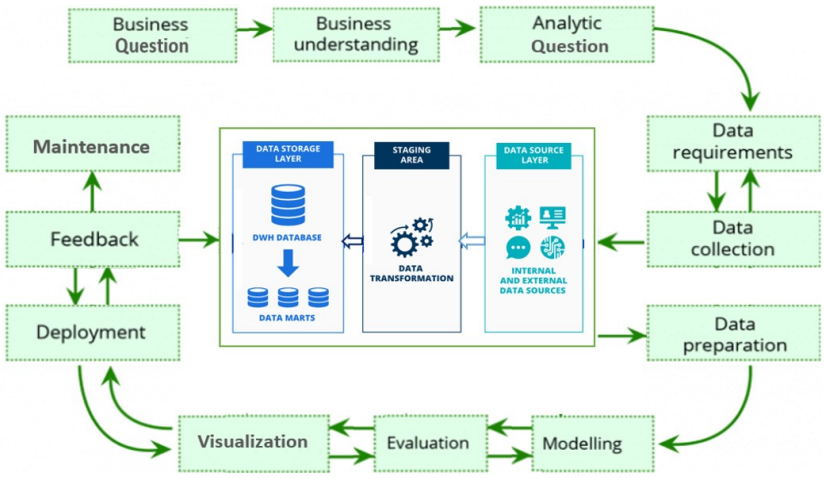


Figure 1.1: Data science process workflow

We will guide students to complete an end-to-end data science project that involves question formulation, data extraction and transformation, identification of models and algorithms, model deployment and monitoring, and effective communication. The final DS consulting project will be conducted in a workplace like environment and with real-world problems and using data sources. Through doing the project, students will

- understand the big picture of DS and its capacity of solving practical problems;

- sharpen their to skills in programming and information extraction and transforming;
- enhance their understanding and effective utilization of models and algorithms;
- improve their effective communications skills, particularly the information visualization;
- learn how to extract **actionable** information and effectively implement the DS product.

1.3 Tentative Topics

We will cover the following major topics in this course.

- Data science process
- Formulating analytic question from business questions
- Data source identification, collection, and processing
- EDA and Visualization in basic feature engineering

- Statistical models for data science
- Performance measures in predictive analytics
- Training, testing and cross validation - data-driven methods
- Survey of supervised machine learning algorithms and models
- Unsupervised algorithms
- Algorithm-based Feature engineering methods
- Model / algorithm deployment and updating (including real-time predictive analytics)
- Four-four workshop on a real-world data science consulting team project.

Chapter 2

Data Science - A Big Picture

This note introduces the big picture of data science and tools and infrastructure needed for perform data science tasks. These include technical tools such as models and algorithms and data storage and extraction tools, etc.

2.1 Data Science Process

Although there are different versions of definition of data science, but the fundamental technical components have been well established for the field by the academic community. There is a consensus that the four pillars of data science are depicted in the following figure.

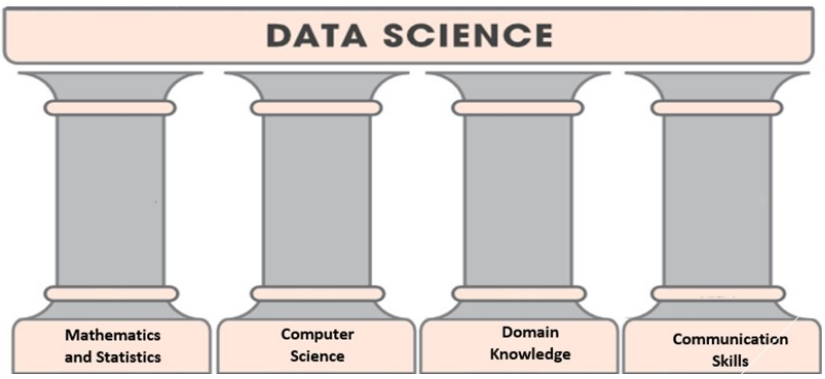


Figure 2.1: Four pillars of data science.

This is pretty much consistent with Tukey's *new science* and Cleveland's pro-

posal of college DS curriculum.

2.1.1 What is Data?

According to the definition in *Wikipedia*,

> Data is a set of values of qualitative or quantitative variables; restated, pieces of data are individual pieces of information. Data is measured, collected, and reported, and analyzed, whereupon it can be visualized using graphs or images. Data as a general concept refers to the fact that some existing information or knowledge is represented or coded in some form suitable for better usage or processing.

The data world of data science, **data is all recordable information** such as images, audios, and videos.

2.1.2 Data Storage and Retrieval

Once a business question is translated to an analytic questions, the next step is to identify data sources. This includes finding data sources in existing data warehouses in an organization or collecting data based on protocols or using external data sources. This stage involves data storage and retrieval. The following figure shows the simplest architecture of a data science professional will work with.

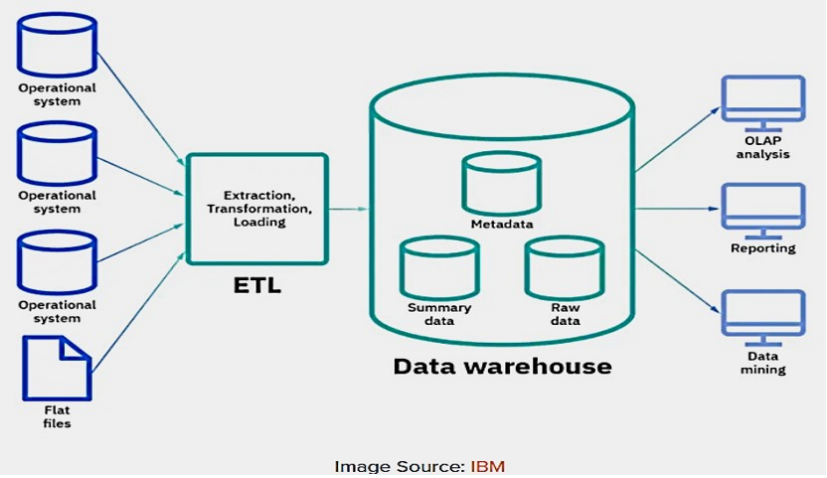


Figure 2.2: Basic architechute associated with data storage and retrieval.

2.1.3 Different DA Roles in Industry

There are different types of data science roles in industry. Depending on the specific job role and organizations, Different job titles were used for different job role.

Mathematical and Statistical analysis - statistician: This role uses Statistics Methods such as Descriptive statistics, EDA, regression analysis, time series and nonparametric analysis, etc. Sometimes, mathematical methods such mixture modeling and various optimization algorithms are also used when performing certain tasks.

Machine Learning Algorithms - Machine Learning Engineer: This role uses machine learning algorithms routinely in the work. Typical algorithms include *classification algorithms* (Rule-based, Tree-based methods, Kernel method-KNN, Naive Bayes, Support vector machine-SVM, Neural networks, etc.), Clustering Algorithms (K-mean, Hierarchical clustering) and Anomaly detection algorithms, etc.

Programming - Software Engineer: Database query languages (SQL and NoSQL), Script languages (such as Python, R, Julia, SAS, etc.), Data wrangling and cleaning using different software tools, creating visuals via coding, Basic software development skills (writing APIs), Writing industry standard production code.

Technology - IT and Data Engineer: This role uses database technology, big data technology, data science platforms, software programs, collaboration tools, communication tools.

Business Intelligence - Business analyst: This role requires a business mind set, project management skills and communication skills.

2.1.4 Cloud Computing

Cloud computing is the on-demand availability of application and infrastructure computing resources such as servers, storage, databases, networking, software, analytics, and intelligence — over the internet, without direct active management by the customer.

The key characteristics of cloud computing are

- **On-demand self-services**: Users monitor and manage computing resources as necessary without the assistance of human administrators.
- **Broad network access**: A wide range of hardware and established networks are usually used to deliver computing services.
- **Rapid elasticity**: Resources for the computing services can be scaled up and down quickly as required.

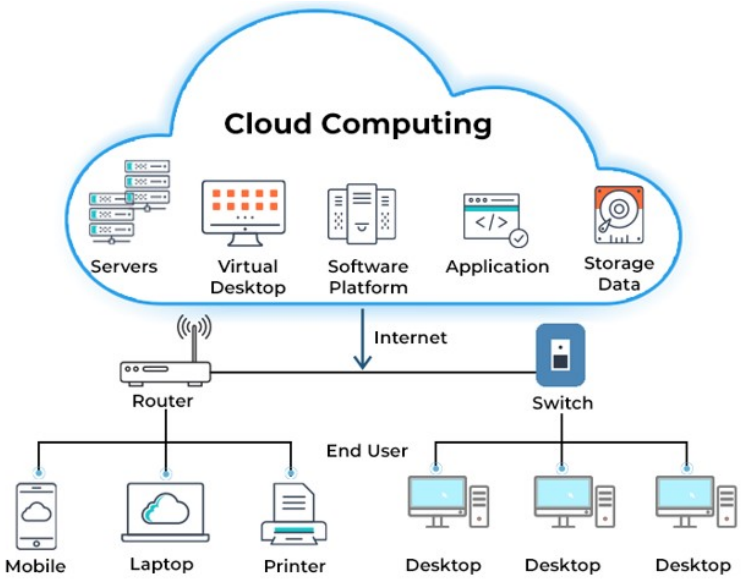


Figure 2.3: The architecture of cloud computing.

- **Resource pooling**: Ad hoc sharing of networks, servers, storage, apps, and services by several users and applications.
- **Resilient computing**: computing services are ensured with high uptime and dependability.
- **Flexible pricing structures**: including pay-per-use, subscription-based, and spot pricing.
- **Security**: To safeguard the privacy of sensitive data and their users' data.
- **Automation**: Cloud computing services feature a high level of automation with little to no manual input.

2.2 From Business Questions to Analytic Question

Business questions (if any) are usually ambiguous. Quite often in practice, there is no business question but a description of business issues or goals. As an example, let's assume a credit card company has been suffering fraud loss and decided to create a fraud detection team to reduce the current fraud loss.

2.2.1 Business Goal

Business Goal: *The business goal of the executive team* is to cut the current fraud loss by half within one year. The business goal is clear and achievable based on the current fraud loss in the credit card industry. The question is that the business goal does not operational information for the analytic and data science team.

In order to appropriately formulate the corresponding analytic question, we need to understand the business logic of this business goal - how credit card transactions are processed?

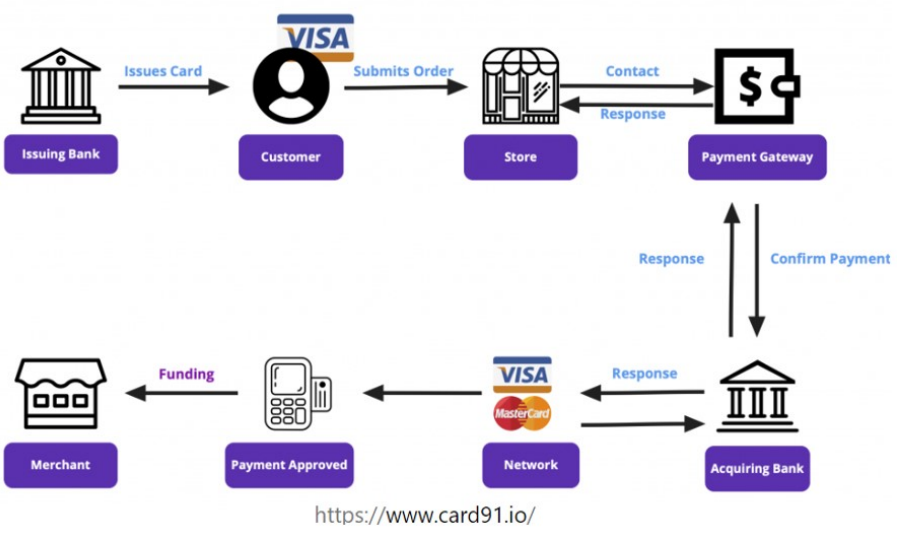


Figure 2.4: Credit card transaction processing workflow.

The electronic credit card payment process may be difficult to understand at first. The diagram above illustrates the workflow of payment process.

1. The **cardholder** swipes the card at the merchant POS in exchange for goods or services.

2. The **merchant** sends a request for payment authorization to their **payment processor**.

3. The **payment processor** submits transactions to the appropriate card association, eventually reaching the **issuing bank**.

4. **Authorization** requests are made to the issuing bank, including parameters such as CVV (card verification value), expiration date, etc. to validate the incoming transaction request.

5. The **issuing bank** approves or declines the request. The transaction can be declined in case of insufficient funds.

6. The **issuing bank** then sends the approval (or denial) statement back along the line to the **card association**, **merchant bank**, and finally to the **merchant**.

7. **Merchants** send batches of authorized transactions to their **payment processor**.

8. The **payment processor** passes transaction details to the card associations that communicate the appropriate debits with the **issuing bank** in their network.

9. The **issuing bank** charges the cardholder's account for the amount of the transactions,

10. The **issuing bank** then transfers the appropriate amount for the transactions to the **merchant bank**, minus the interchange fees.

11. The **merchant bank** deposits funds into the merchant account.

If a fraudulent transaction was detected during the authorization process (the above steps 1- 6), there will be zero loss to the company. If the fraudulent transaction escaped from the authorization process, the company would lose at least one transaction, in general more if there is no way to identify them.

2.2.2 Analytic Question

Analytic Question: *The high-level analytic question* is how to identify fraudulent transactions and stop them? - This high-level analytic question is not operational since we cannot be based on this question to identify data. We still need to drill down to gather some granular information in order to develop actionable plans for data collection and analysis planning.

- There are different types of fraud, different fraud has different patterns that require different pieces of information to identify.
  - Identity theft fraud?
  - Lost card fraud?
  - Application fraud?
  - Account takeover fraud?
- Fraud interception at authorization process - proactive action for zero loss.
- If the initial fraudulent transaction escaped from the fraud check during the authorization process, fraudsters would continue stealing, what analytic action we should take.

Based on the above sub-analytic questions, we make analytic plans and create a set of sub-tasks to identify data sources and models and algorithms to build a detection system. This means, we will not work with one model or algorithm,

we need a set of potentially very different models and algorithms to tackle the seemingly simple business question.

Types of Models/Algorithms and Required Data Sources: We will use two of the above sub-analysis questions to illustrate the potential models/algorithms and relevant information needed to build identification systems.

- **Proactive Fraud Detection**: This detection is most favorable is it intercepts the fraud before the transaction is complete (i.e., the transaction will be declined). Some of the information such as geolocation of the merchant site, time and previous transaction site and time, card verification value (CVV), expiration, credit limit, etc. in the initial fraud check. This types of information can be used to develop business rules (expert system). Of cause, we can also build predictive models using this information.
- **Application Fraud**: A fraudster can use fake information to apply for credit and then use the credit and discard the card. The first transaction is not easy to catch, what analytic models and algorithms can do is to detect the card and suspend it as soon as possible. The more effective way to stop application fraud is to build models to prevent potential application fraud during card application. The information needed to build the model can be extracted from the application and personal credit information and other relevant information from third-party.
- **Identity Theft Fraud**: Identity fraud is common where criminals make purchases or obtain cash advances in the victim's name. This can be with an existing account, via theft of victim's physical credit card or victim's account numbers and PINs, or by opening new credit card accounts in victim's name. Because the fraudsters information is not available, we can use customers' proxy information on spending patterns to build models.

The types of algorithms and models for fraud identification will be discussed in more detail in the subsequent chapters.

2.3 Concepts of Relational Databases and SQL

Every data science professional should have basic knowledge of **databases** and **data warehouse** to accomplish all data science and other analytic projects.

2.3.1 What is Database?

A **database** is a collection of *related data* set collected from the real world. It is designed to be built and populated with data for a specific task. It is also a building block of data solutions. A database

- offers the security of data and its access;
- offers a variety of techniques to store and retrieve data;

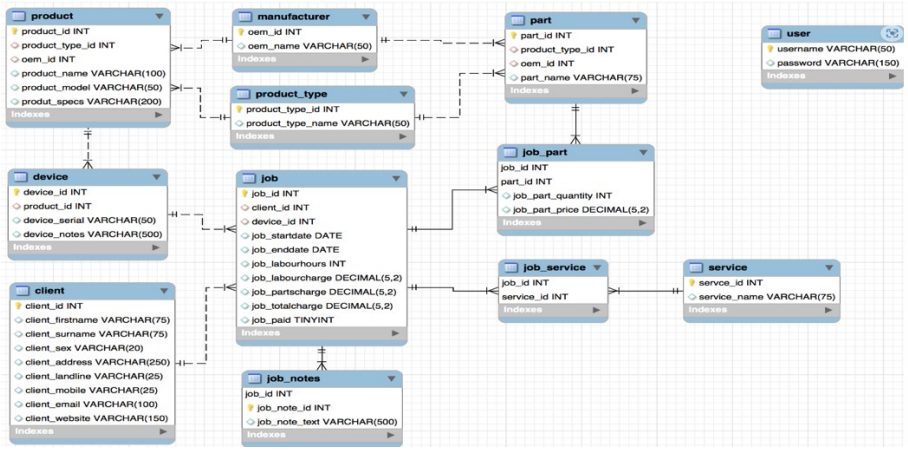


Figure 2.5: Relational datatables in a relational database.

- acts as an efficient handler to balance the requirement of multiple applications using the same data;
- A DBMS offers integrity constraints to get a high level of protection to prevent access to prohibited data;
- allows users to access concurrent data in such a way that only a single user can access the same data at a time.

2.3.2 What is a Data Warehouse?

A **data warehouse** is an information system which stores historical and commutative data from single or multiple sources. It is designed to *analyze, report, and integrate transaction data* from different sources.

A basic structure of an organization's data warehouse is depicted in the following.

A data warehouse.

- helps business users to access critical data from some sources all in one place;
- provides consistent information on various cross-functional activities;
- helps you to integrate many sources of data to reduce stress on the production system.
- helps you to reduce TAT (total turnaround time) for analysis and reporting.
- helps users to access critical data from different sources in a single place so, it saves user's time of retrieving data information from multiple sources. You can also access data from the cloud easily.
- allows you to store a large amount of historical data to analyze different periods and trends to make future predictions.

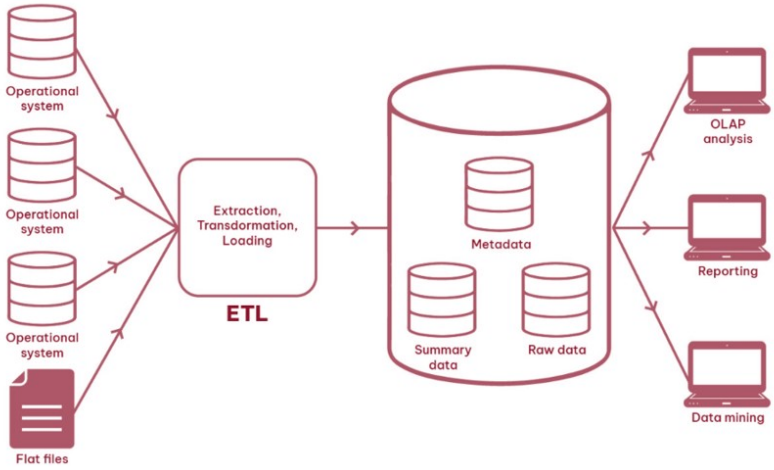


Figure 2.6: Basic structure of data warehouse.

- enhances the value of operational business applications and customer relationship management systems
- separates analytics processing from transactional databases, improving the performance of both systems
- provides more accurate reports.

2.3.3 Difference between Database and Data Warehouse

Both databases and data warehouses use relational data structures. But they are different. A data warehouse exists as a layer on top of another database or databases (usually OLTP databases). The following table lists the difference between two systems.

2.3.4 Database Management System (DBMS)

Database Management Systems (DBMS) are software systems used to store, retrieve, and run queries on data. DBMS manages the data, the database engine, and the database schema, allowing for data to be manipulated or extracted by users and other programs. This helps provide data security, data integrity, concurrency, and uniform data administration procedures.

The following figure depicts the structure of a simple DBMS.

Structured query language (SQL) is a programming language for storing and processing information in a relational database. A relational database stores information in tabular form, with rows and columns representing different data attributes and the various relationships between the data values. We can use SQL statements to store, update, remove, search, and retrieve information from

Parameter	Database	Data Warehouse
Purpose	Is designed to record	Is designed to analyze
Processing Method	The database uses the Online Transactional Processing (OLTP)	Data warehouse uses Online Analytical Processing (OLAP).
Usage	The database helps to perform fundamental operations for your business	Data warehouse allows you to analyze your business.
Tables and Joins	Tables and joins of a database are complex as they are normalized.	Table and joins are simple in a data warehouse because they are denormalized.
Orientation	Is an application-oriented collection of data	It is a subject-oriented collection of data
Storage limit	Generally limited to a single application	Stores data from any number of applications
Availability	Data is available real-time	Data is refreshed from source systems as and when needed
Usage	ER modeling techniques are used for designing.	Data modeling techniques are used for designing.
Technique	Capture data	Analyze data
Data Type	Data stored in the Database is up to date.	Current and Historical Data is stored in Data Warehouse. May not be up to date.
Storage of data	Flat Relational Approach method is used for data storage.	Data Warehouse uses a dimensional and normalized approach for data structure. Example: Star and snowflake schema.
Query Type	Simple transaction queries are used.	Complex queries are used for analysis purpose.
Data Summary	Detailed Data is stored in a database.	It stores highly summarized data.

Figure 2.7: Comparison between database and data warehouse.

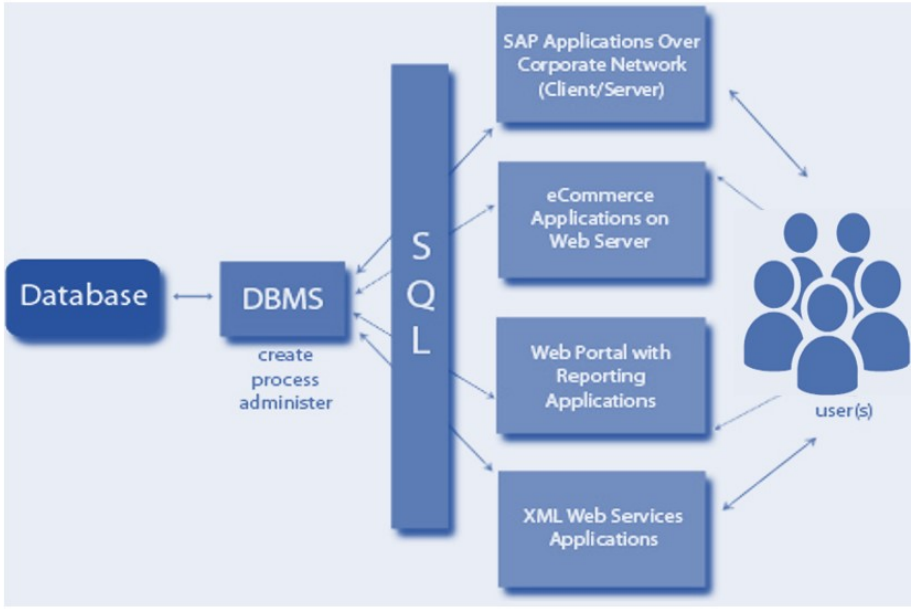


Figure 2.8: Comparison between database and data warehouse.

the database. We can also use SQL to maintain and optimize database performance.

2.3.5 Some Definitions and Notations of Relational Tables

- **Names Associated with Relational Tables**

Name – each relation in a relational database should have a name that is unique among other relations.

Attribute – each column in a relation.

The degree of the relation – the total number of attributes for a relation.

Tuple – each row in a relation.

The cardinality of the relation – the total number of rows in a relation.

- **Operations among Relational Tables**

In a relational database, we can define several **operations** to create new relations out of the existing ones.

- **Basic operations** are
  - Unary Operation: Insert, Delete, Update, Select, Project,
  - Binary Operation: Join, Union, Intersection, Difference.

Chapter 3

Running SQL in SAS and R

Although R and Python has different data wrangling libraries we can use to connect to databases for information extraction. Structural Query Language (SQL) is native to different DBMS. Some degree of proficiency in SQL crucial for every data professional.

This note use SAS PROX SQL and several R packages to run SQL in SAS and R.

3.1 Running SQL in SAS

The PROC SQL in SAS is powerful. We can run authentic SQL script with in SAS SQL. One can use free SAS Studio via SAS on-Demand. We will use three relational data tables in the following SAS code (URLs are also in the SAS code).

In order to make the code workable in SAS environment, we simply copy and paste the code to include it in this note. NOte that, SAS converted imported relational tables to SAS data sets. The resulting tables in PROC SAS are still SAS tables stored in the designated library. If a library reference is not given, the resulting SAS data set will be saved in the temporary library.

3.1.1 Loading Data

```
LIBNAME sql "/home/u50445699/STA551";

filename dat01 url 'https://pengdsci.github.io/datasets/AnimalSurvey/plots.csv';
filename dat02 url 'https://pengdsci.github.io/datasets/AnimalSurvey/species.csv';
filename dat03 url 'https://pengdsci.github.io/datasets/AnimalSurvey/surveys.csv';

*write a macro to load multiple data files;
```

```
%MACRO IMPORTCSV(in_csv, out_sas);
PROC IMPORT DATAFILE=&in_csv
   DBMS=csv
   OUT=sql.&out_sas
   REPLACE;
RUN;
%MEND;

*call macros to load data files;
%IMPORTCSV(dat01, plots);
%IMPORTCSV(dat02, species);
%IMPORTCSV(dat03, survey);
```

3.1.2 Basic SQL Syntax and Clauses

####. First query with clause SELECT

(a). Create a table view

(b). Create a new table (SAS data set)

```
*Table view;
PROC SQL;
/** The actual SQL code starts here **/
SELECT year
FROM sql.survey;
/***/
QUIT;


* create a new table;
PROC SQL;

CREATE TABLE sql.YMD AS
SELECT year,
       month,
       day
FROM sql.survey;

QUIT;

* create a new table: select all variables;
PROC SQL;

CREATE TABLE sql.survey_all AS
SELECT *
FROM sql.survey;
```

```
QUIT;
```

CAUTION: SQL uses "CREATE VIEW name_of_table AS" to create a view table that is saved in the database and can be used to look at, filter, and even update information.

3.1.2.1 Select unique values

```
* one variable;
PROC SQL;
SELECT DISTINCT year
FROM sql.survey;
QUIT;

* two variable;
PROC SQL;
SELECT DISTINCT year,
                species_id
FROM sql.survey;
QUIT;
```

3.1.2.2 Calculated values

```
* Create a view;
PROC SQL;
SELECT year,
       month,
       day,
       INPUT(weight, best.)/1000.0 AS wgt
FROM sql.survey;
QUIT;

* Create a table and define a new variable
  based on the calculated values;
PROC SQL;
CREATE TABLE sql.Add_new_var AS
SELECT year,
       month,
       day,
       INPUT(weight, best.)/1000.0 AS wt_kilo
FROM sql.survey;
QUIT;


proc contents data = sql.surveys; run;
```

```
PROC SQL;
SELECT plot_id,
       species_id,
       sex,
       weight,
       ROUND(INPUT(weight, best.)/ 1000.0, 0.01) /* ROUND() is a SAS function! */
FROM sql.survey;
QUIT;
```

3.1.2.3 Filtering

```
* subsetting by filtering - WHERE statement;
* Single condition;
PROC SQL;
SELECT *
FROM sql.survey
WHERE species_id='DM';
QUIT;


* Multiple conditions: AND;
PROC SQL;
SELECT *
FROM sql.survey
WHERE (year >= 2000) AND (species_id = 'DM');
QUIT;

* Multiple conditions: OR;
PROC SQL;
SELECT *
FROM sql.survey
WHERE (species_id = 'DM') OR (species_id = 'DO') OR (species_id = 'DS');
QUIT;
```

3.1.2.4 Special Keywords simplify WHERE statement

```
* use of keyword IN;
PROC SQL;
SELECT *
FROM sql.survey
WHERE (year >= 2000) AND (species_id IN ('DM', 'DO', 'DS'));
QUIT;
```

3.1.2.5 Sorting Variables

```
* Ascending ordering - default;
```

```
PROC SQL;
SELECT *
FROM sql.species
ORDER BY taxa ASC;
QUIT;

* descending ordering;
PROC SQL;
SELECT *
FROM sql.species
ORDER BY taxa DESC;
QUIT;

* sorting multiple variables- nest sorting;
PROC SQL;
SELECT *
FROM sql.species
ORDER BY genus ASC, species ASC;
QUIT;
```

3.1.2.6 Order of Execution

```
*Clauses are written in a fixed order: SELECT, FROM, WHERE, then ORDER BY. ;
PROC SQL;
SELECT genus, species
FROM sql.species
WHERE taxa = 'Bird'
ORDER BY species_id ASC;
QUIT;
```

3.1.2.7 Summary Statistics with groups

```
*no group - Using the wildcard simply counts the number of records (rows);
PROC SQL;
SELECT COUNT(*)
FROM sql.survey;
QUIT;

* calculate the sum of a numerical variable;
PROC SQL;
SELECT COUNT(*) ,
       SUM(INPUT(weight, best.))
FROM sql.survey;
QUIT;

* calculate the sum of a numerical variable - adding names of summary statistics;
```

```
PROC SQL;
SELECT COUNT(*) AS sample_size,
       SUM(weight) AS total_weight
FROM sql.survey;
QUIT;

*summary statistics within subgroups - GROUP BY clause;
* This is equalent to a univeriable frequency table;
PROC SQL;
SELECT species_id,
       COUNT(*)
FROM sql.survey
GROUP BY species_id;
QUIT;
```

3.1.2.8 HAVING clause based on aggregated variables

```
* conditioning on species size;
PROC SQL;
SELECT species_id,
       COUNT(species_id)
FROM sql.survey
GROUP BY species_id
HAVING COUNT(species_id) > 10;
QUIT;

* conditioning on species size - a better code;
PROC SQL;
SELECT species_id,
       COUNT(species_id) AS species_size
FROM sql.survey
GROUP BY species_id
HAVING species_size > 10;
QUIT;
```

3.1.2.9 Ordering aggregated results.

```
* sorting aggregated variables -- This DOES NOT work!
* Summary functions are restricted to the SELECT and HAVING clauses only;
PROC SQL;
SELECT species_id,
       COUNT(*)
FROM sql.survey
GROUP BY species_id
ORDER BY COUNT(species_id);
QUIT;
```

```
* sorting aggregated variables;
* use the new name in the ORDER BY clause;
PROC SQL;
SELECT species_id AS subtotal,
       COUNT(*)
FROM sql.survey
GROUP BY species_id
ORDER BY subtotal;
QUIT;
```

3.1.2.10 NULL - Working with Missing values

```
* keyword IS;
PROC SQL;
SELECT *
FROM sql.survey
WHERE species_id IS NULL;
QUIT;

* keyword IS NOT;
PROC SQL;
SELECT *
FROM sql.survey
WHERE species_id IS NOT NULL;
QUIT;

* for non-missing values, we use IS/IS NOT or =/!=;
* CAUTION: "=" and "==" both work in SQL, However, "==" does NOT in SAS.;
PROC SQL;
SELECT SUM(INPUT(weight, best.)),
       COUNT(*),
       SUM(INPUT(weight, best.))/COUNT(*)
FROM sql.survey
WHERE species_id = 'PE';
QUIT;

* for non-missing values, we use IS/IS NOT or =/!=;
* CAUTION: "=" and "==" both work in SQL, However, "==" does NOT in SAS.;
* != works in SQL, but not in SAS. ^= works in SAS;
PROC SQL;
SELECT SUM(INPUT(weight, best.)),
       COUNT(*),
       SUM(INPUT(weight, best.))/COUNT(*)
FROM sql.survey
WHERE species_id ^= 'PE';
```

```
QUIT;
```

3.1.2.11 Working with multiple tables: JOIN

```
*INNER JOIN;
*Need to use ALIAS to rename/name the data set since
 the files are stored in the SAS permanent libaray;
PROC SQL;
SELECT *
FROM sql.survey AS surveys
JOIN sql.species AS species
ON surveys.species_id = species.species_id;
QUIT;


* we can simply rename the tables as A and B using alias;
PROC SQL;
CREATE TABLE sql.INNERJOIN AS
SELECT *
FROM sql.survey AS A
JOIN sql.species AS B
ON A.species_id = B.species_id;
QUIT;

*left join;
PROC SQL;
CREATE TABLE sql.LEFTJOIN AS
SELECT *
FROM sql.survey AS A
LEFT JOIN sql.species AS B
ON A.species_id = B.species_id;
QUIT;

*right join;
PROC SQL;
CREATE TABLE sql.RIGHTJOIN AS
SELECT *
FROM sql.survey AS A
RIGHT JOIN sql.species AS B
ON A.species_id = B.species_id;
QUIT;

*full join;
PROC SQL;
CREATE TABLE sql.FULLTJOIN AS
SELECT *
```

```
FROM sql.survey AS A
FULL JOIN sql.species AS B
ON A.species_id = B.species_id;
QUIT;

* We can select some variables from individual tables
* then join the two sub tables.;
* CAUTION: We will NOT select any variables in species table
*  to include in the new table;
PROC SQL;
SELECT A.species_id,
       A.sex,
       AVG(INPUT(a.hindfoot_length, best.)) as mean_foot_length
FROM sql.survey  AS A
JOIN sql.species AS B
ON A.species_id=B.species_id
WHERE taxa = 'Rodent' AND A.sex IS NOT NULL
GROUP BY A.species_id, A.sex;
QUIT;
```

3.1.2.12 Creatng New Variables

The following code defines new variables using string functions in SQL. This method is sometimes used to define composite keys

```
PROC SQL;
SELECT *,
       species_id||'-'||sex AS newKey
FROM sql.surveys;
QUIT;
```

3.1.2.13 Nest Queries

```
* nest queries;
* The SELECT ... FROM in the denominator is self-contained
* It is NOT affected by GROUP BY statement. The COUNT() function
* returns the total size of the data;
PROC SQL;
SELECT B.taxa,
       100.0*COUNT(*)/(SELECT COUNT(*) FROM sql.survey)  AS Percentage
FROM sql.survey AS A
JOIN sql.species AS B
ON A.species_id = B.species_id
GROUP BY taxa;
QUIT;
```

3.2 Running SAS in R

To run SQL clauses in R, we need to use several R libraries (installed and loaded in the above R setup code chunk). There are different ways to run SQL query in R. We only introduce one methods that is close to the authentic SQL code that can be run a DBMS.

3.2.1 Connect R to Existing Database

If there is an existing database, the following code connects R to the database.

```
con <- DBI::dbConnect(drv = odbc::odbc(),
                      Driver = "driver_name",
                      Server = "server_url",
                      Database = "database_name",
                      user = "user", #optional
                      password = "password") #optional
```

This section shows the three basic steps to run SQL in R using R Markdown starting with a set of relational tables.

1. Load relational data tables as usual to R.
2. Create a SQLite (relational) database that contain these relational table.
3. Create R code chunk and connect to the created database using Chunk options.

3.2.2 Create SQLite Database with R

If modeling requires a data set that contains information from multiple relational data tables, we need to perform data management to aggregate the required information from different data tables. We can load the different data sets in different formats using appropriate R functions.

As an example, We use three ecological survey data sets to create a database.

```
#Load the sample data
plots <- read.csv("https://pengdsci.github.io/datasets/AnimalSurvey/plots.csv")
species <- read.csv("https://pengdsci.github.io/datasets/AnimalSurvey/species.csv")
surveys <- read.csv("https://pengdsci.github.io/datasets/AnimalSurvey/surveys.csv")
```

Next, we create a SQLit database using several R libraries.

```
#Create database
con <- dbConnect(drv = SQLite(),
                 dbname = ":memory:")

#store sample data in database
dbWriteTable(conn = con,
```

```
             name = "plots",
             value = plots)

dbWriteTable(conn = con,
             name = "species",
             value = species)

dbWriteTable(conn = con,
             name = "surveys",
             value = surveys)

#remove the local data from the environment
rm(plots, species, surveys)
```

We can use table view function `tbl()` to explore the information of relational data tables in the database. Note that, we

```
tbl(src = con, #the source if the database connection profile
    c("surveys")) #the name of the table to preview
```

```
## # Source:   table<surveys> [?? x 10]
## # Database: sqlite 3.41.2 [:memory:]
##        X record_id month   day  year plot_id species_id sex   hindfoot_length weight
##    <int>     <int> <int> <int> <int>   <int> <chr>      <chr>           <int>  <int>
##  1     1         1     7    16  1977       2 NL         M                  32     NA
##  2     2         2     7    16  1977       3 NL         M                  33     NA
##  3     3         3     7    16  1977       2 DM         F                  37     NA
##  4     4         4     7    16  1977       7 DM         M                  36     NA
##  5     5         5     7    16  1977       3 DM         M                  35     NA
##  6     6         6     7    16  1977       1 PF         M                  14     NA
##  7     7         7     7    16  1977       2 PE         F                  NA     NA
##  8     8         8     7    16  1977       1 DM         M                  37     NA
##  9     9         9     7    16  1977       1 DM         F                  34     NA
## 10    10        10     7    16  1977       6 PF         F                  20     NA
## # i more rows
```

3.2.3 Running SQL Queries in R Code chunks

To use SQL in RMarkdown, we need the following chunk options:

1. sql
2. connection = “database-name”
3. output.var = “output-dataset-name”

If we create a data view only, we simply ignore option `output.var =`

Following are few examples of SQL queries based on the animal survey data tales in the database.

3.2.3.1 Subsetting and Duplicating Data

1. Extract year, month and day from `survey` table

```
SELECT
  surveys.year, surveys.month, surveys.Day
FROM
 surveys /* pointer is not needed since it is in the database */
WHERE
  surveys.species_id IN ('NL', 'DM') AND
  surveys.sex = 'M'
```

2. Duplicate a data and rename it

```
SELECT
  surveys.*
FROM
 surveys
```

3. Create a table view (i.e., no data set will be created and saved)

```
SELECT
  surveys.year, surveys.month, surveys.Day
FROM
 surveys
WHERE
  surveys.species_id = 'NL' AND
  surveys.sex = 'M'
```

3.2.3.2 Define A New Variable

1. Define a new variable with simple arithmetic operations

```
SELECT
    surveys.plot_id,
    surveys.species_id,
    surveys.sex,
    surveys.weight,
    surveys.weight/100 AS wt_kilo  /*should not the pointer in front of the name of th
FROM
   surveys
```

2. Define new variables using string functions in SQL

```
SELECT surveys.*,
       surveys.species_id||'-'||surveys.sex AS newKey
FROM surveys
```

3. Define new variables with aggregated information

```
SELECT surveys.species_id,
       COUNT(surveys.species_id) AS species_ctr
FROM surveys
GROUP BY surveys.species_id
HAVING species_ctr > 10
```

3.2.3.3 Sorting Variables

1. Sort data based on the summarized statistics of a variable

Summary functions are restricted to the SELECT and HAVING clauses only;

```
SELECT surveys.species_id
FROM surveys
GROUP BY surveys.species_id
ORDER BY COUNT(surveys.species_id);
```

2. Sort data based on a new variable defined using summarized statistics of a variable.

```
/* create a table view*/
SELECT surveys.species_id AS subtotal,
       COUNT(*)
FROM surveys
GROUP BY surveys.species_id
ORDER BY subtotal;
```

3.2.3.4 Join Tables

This section introduce commonly used join operation to merge tables using the common key(s).

1. Inner Join

```
SELECT *
FROM surveys AS A
JOIN species AS B
ON A.species_id = B.species_id;
```

2. Left Join

```
SELECT *
FROM surveys AS A
LEFT JOIN species AS B
ON A.species_id = B.species_id;
```

3. Right Join

```
SELECT *
FROM surveys AS A
```

```
RIGHT JOIN species AS B
ON A.species_id = B.species_id;
```

4. Full Join

```
SELECT *
FROM surveys AS A
FULL JOIN species AS B
ON A.species_id = B.species_id;
```

5. Join sub-tables

```
SELECT A.species_id,
       A.sex,
       AVG(A.weight) as mean_wgt
FROM surveys AS A
JOIN species AS B
ON A.species_id=B.species_id
WHERE taxa = 'Rodent' AND A.sex IS NOT NULL
GROUP BY A.species_id, A.sex;
```

3.2.3.5 Subqueries

1. Sample size

```
SELECT COUNT(*)
FROM surveys
```

2. Relative Frequency with sub-query

```
SELECT B.taxa,
       100.0*COUNT(*)/(SELECT COUNT(*) FROM surveys)  AS Percentage
FROM surveys AS A
JOIN species AS B
ON A.species_id = B.species_id
GROUP BY taxa;
```

Chapter 4

Exploratory Data Analysis (EDA)

The US National Institute of Standards and Technology (NIST) defines EDA as:

> An approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to maximize insight into a data set, uncover underlying structure, extract important variables, detect outliers and anomalies, test underlying assumptions, develop parsimonious models, and determine optimal factor settings.

The term EDA was coined by John Tukey in the 1970s. According to Tukey: "It (EDA) is important to understand what you CAN DO before you learn to measure how WELL you seem to have DONE it… Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone – as the first step.

Tukey clearly explains the purpose of EDA. In classical statistics, EDA has been primarily used to inspect the distribution of variables and observe patterns to make hypotheses and test (validating). To be more specific, EDA is for

1. inspecting the distribution of variables,
2. detecting (and/or removing) outliers,
3. examining the trend of variables
4. assess the associations between variables

The general tools used for EDA in classical statistics are numerical descriptive statistics with basic graphics such as histograms and scatter plots, etc. A cautionary note about EDA is its descriptive nature. EDA is NOT an inferential method.

In Data Science, more EDA tools will be used for feature engineering in order to improve the performance of underlying models and algorithms. This note will systematically outline EDA tools and their applications in both classical statistics and data science.

Working Data Set

For convenience, we use a data set to illustrate the concepts and methods as we proceed. The data set can be found at https://pengdsci.github.io/datasets/MelbourneHousingMarket/MelbourneHousing.csv

```
MelbourneHousePrice = read.csv("https://pengdsci.github.io/datasets/MelbourneHousingMa
```

4.1 Tools of EDA and Applications

This section summarizes the tools of EDA and their applications in both classical statistics and data science.

4.1.1 Descriptive Statistics Approach

This approach uses tables and summarized statistics to uncover the pattern in the data. These patterns include the distribution of feature variables, the correlation between variables, missing values proportions, outliers, etc. Measures such five number summary, quartiles, IQR, and standardization of numerical variables.

R has a powerful function `summary()` that produces summarized descriptive statistics for every variable in the data set.

```
summary(MelbourneHousePrice)
```

```
##     Suburb            Address              Rooms            Type              Price
##  Length:34857       Length:34857       Min.   : 1.000   Length:34857       Min.   :
##  Class :character   Class :character   1st Qu.: 2.000   Class :character   1st Qu.:
##  Mode  :character   Mode  :character   Median : 3.000   Mode  :character   Median :
##                                        Mean   : 3.031                      Mean   :
##                                        3rd Qu.: 4.000                      3rd Qu.:
##                                        Max.   :16.000                      Max.   :
##                                                                            NA's   :
##    SellerG              Date              Distance           Postcode            Bedr
##  Length:34857       Length:34857       Length:34857       Length:34857       Min.
##  Class :character   Class :character   Class :character   Class :character   1st Qu
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character   Median
##                                                                               Mean
##                                                                               3rd Qu
##                                                                               Max.
##                                                                               NA's
##       Car            Landsize         BuildingArea       YearBuilt     CouncilArea
```

```
##  Min.   : 0.000   Min.   :     0.0   Min.   :    0.0   Min.   :1196    Length:34857       Min.
##  1st Qu.: 1.000   1st Qu.:   224.0   1st Qu.:  102.0   1st Qu.:1940    Class :character   1st
##  Median : 2.000   Median :   521.0   Median :  136.0   Median :1970    Mode  :character   Medi
##  Mean   : 1.729   Mean   :   593.6   Mean   :  160.3   Mean   :1965                       Mean
##  3rd Qu.: 2.000   3rd Qu.:   670.0   3rd Qu.:  188.0   3rd Qu.:2000                       3rd
##  Max.   :26.000   Max.   :433014.0   Max.   :44515.0   Max.   :2106                       Max.
##  NA's   :8728     NA's   :11810      NA's   :21115     NA's   :19306                      NA's
##    Longtitude     Regionname        Propertycount
##  Min.   :144.4   Length:34857       Length:34857
##  1st Qu.:144.9   Class :character   Class :character
##  Median :145.0   Mode  :character   Mode  :character
##  Mean   :145.0
##  3rd Qu.:145.1
##  Max.   :145.5
##  NA's   :7976
```

We observe from the above summary tables that (1) most of the numeric variables have missing values; (2) The distribution of some of these numeric variables is skewed. We will discuss how to use these observations in feature engineering later.

Remarks: Handling missing values in classical statistics is crucial particularly when the sample size is mall. In data science, most of the projects are based on large data sets. Furthermore, the sample is usually not the ransom sample taken from a well-defined population. Therefore, imputing missing values is less important in many data science projects are less important (usually assume missing at random). Next, we delete all records with missing components.

```
HousePrice = na.omit(MelbourneHousePrice)
```

For a categorical variable, we can use a frequency table to display its distribution. For example,

```
table(HousePrice$Bedroom2)
```

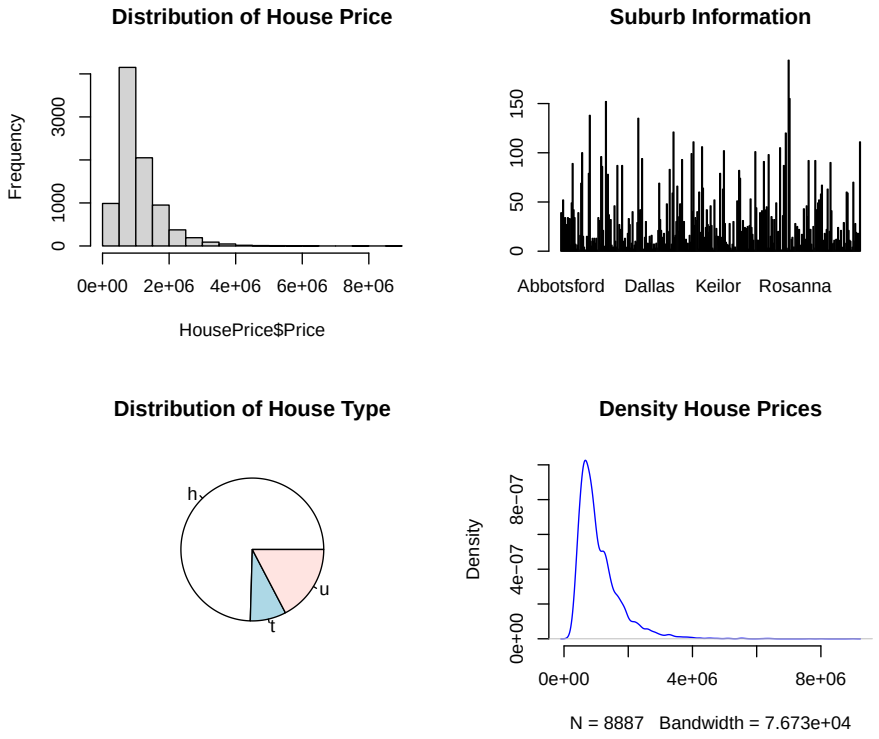
```
##
##    0    1    2    3    4    5    6    7    8    9   10   12
##    5  348 1965 3837 2183  487   50    5    2    3    1    1
```

4.1.2 Graphical Approach

This approach uses basic statistical graphics to visualize the shape of the data to discover the distributional information of variables from the data and the potential relationships between variables. Graphics that are commonly used are histograms, box plots, serial plots, etc.

```
par(mfrow = c(2,2))
hist(HousePrice$Price, main = "Distribution of House Price")
Suburb = table(HousePrice$Suburb)
```

```
barplot(Suburb, main="Suburb Information")
Type = table(HousePrice$Type)
pie(Type, main="Distribution of House Type")
den <- density(HousePrice$Price)
plot(den, frame = FALSE, col = "blue",main = "Density House Prices")
```



We can see We will discuss how to use these observed patterns in feature engineering to yield better results later.

4.1.3 Algorithm-based Method

If there exist some groups (data points clustered), we may want to assign an ID for each group to reduce the overall variations of the data. Including this cluster ID will improve the performance of the underlying model. The clustering algorithm uses a lot of computing resources. As an example, we use the well-known iris data set based on the 4 numerical variables.

```
iris0 = iris[,-5]
res.hc <- eclust(iris0, "hclust", k = 3)
#fviz_dend(res.hc)              # dendrogam
```

```
 fviz_cluster(res.hc)       # scatter plot


NewIris = iris
NewIris$Cluster = res.hc$cluster
```

4.2 Visual Techniques of EDA

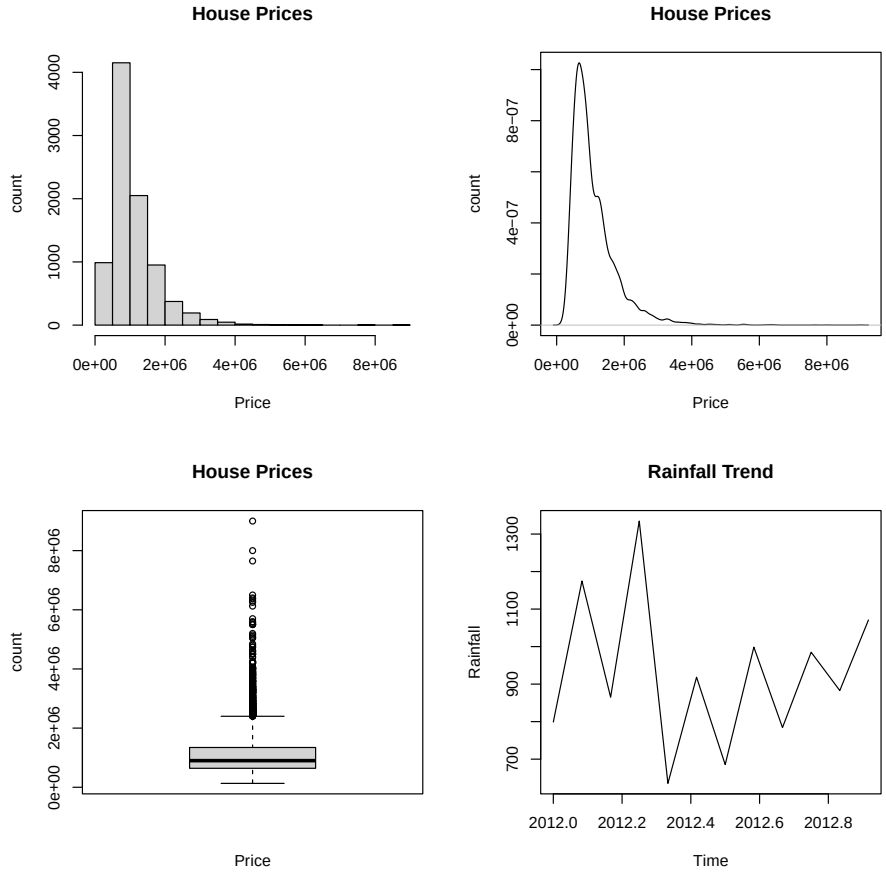
EDA is particularly effective for low-dimensional data. The following discussion will be based on the number and type of variables.

4.2.1 Univariate EDA

- Numerical Variable

The commonly used visual techniques for numerical variables are histograms, density curves, box-plots, serial plots, etc.

```
par(mfrow = c(2,2))
hist(HousePrice$Price, xlab = "Price", ylab = "count", main = "House Prices")
den=density(HousePrice$Price)
plot(den, xlab = "Price", ylab = "count", main = "House Prices")
##
boxplot(HousePrice$Price, xlab = "Price", ylab = "count", main = "House Prices")
##
# Get the data points in the form of an R vector.
rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
# Convert it to a time series object.
rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)
# Plot a graph of the time series.
plot(rainfall.timeseries, ylab = "Rainfall", main = "Rainfall Trend")
```



One can also create a frequency table to look at the distribution.

```
options(digits = 7)
bound = round(seq(100000,9000000, length=15),1)
as.data.frame(table(cut(HousePrice$Price, breaks=bound)))
```

```
##                   Var1 Freq
## 1     (1e+05,7.36e+05] 3101
## 2  (7.36e+05,1.37e+06] 3669
## 3  (1.37e+06,2.01e+06] 1374
## 4  (2.01e+06,2.64e+06]  442
## 5  (2.64e+06,3.28e+06]  172
## 6  (3.28e+06,3.91e+06]   77
## 7  (3.91e+06,4.55e+06]   24
## 8  (4.55e+06,5.19e+06]   11
## 9  (5.19e+06,5.82e+06]    8
## 10 (5.82e+06,6.46e+06]    5
```

```
## 11 (6.46e+06,7.09e+06]    1
## 12 (7.09e+06,7.73e+06]    1
## 13 (7.73e+06,8.36e+06]    1
## 14    (8.36e+06,9e+06]    1
```

The above frequency table gives a similar distribution as shown in the histogram and the density curve.

- Categorical Variable

The commonly used visual techniques for numerical variables are bar charts and pie charts.

```
par(mfrow=c(1,2))
freq.tbl = table(HousePrice$Bedroom2)
barplot(freq.tbl, xlab="Number of Bedrooms", ylab = "Counts", main="Distribution of number of Bed
pie(freq.tbl, xlab="Number of Bedrooms", ylab = "Counts", main="Distribution of number of Bedroom
```


```
kable(as.data.frame(table(HousePrice$Bedroom2)))
```

Var1	Freq
0	5
1	348
2	1965
3	3837
4	2183
5	487
6	50
7	5
8	2
9	3
10	1
12	1

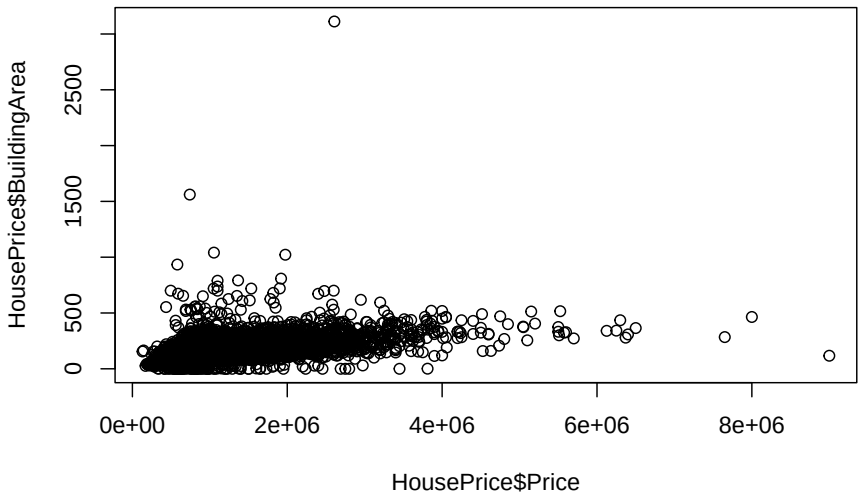
4.2.2 Two Variables

Three different cases involve two variables.

4.2.2.1 Two Numeric Variables

In the case of two numeric variables, the key interest is to look at the potential association between the two. The most effective visual representation is a scatter plot.

```
plot(HousePrice$Price, HousePrice$BuildingArea)
```



The above scatter plot indicates a linear trend between the house price and the building area.

4.2.2.2 Two Categorical Variable

For given two categorical variables, we may be interested in exploring whether they are independent. The two-way table and be used to visualize the potential relationship between the two categorical variables.

```
ftable(HousePrice$Bathroom, HousePrice$Bedroom2)
```

```
##       0    1    2    3    4    5    6    7    8    9   10   12
##
## 1     2  345 1667 1921  255   10    1    0    0    0    0    0
## 2     3    2  295 1788 1495  203   13    1    0    0    0    0
## 3     0    1    3  124  394  206   26    2    1    0    0    0
## 4     0    0    0    4   36   45    9    2    1    0    0    0
## 5     0    0    0    0    3   22    0    0    0    0    0    1
## 6     0    0    0    0    0    1    1    0    0    1    0    0
## 7     0    0    0    0    0    0    0    0    0    1    0    0
## 8     0    0    0    0    0    0    0    0    0    1    0    0
## 9     0    0    0    0    0    0    0    0    0    0    1    0
```

```
chisq.test(HousePrice$Bathroom, HousePrice$Bedroom2)
```

```
## Warning in chisq.test(HousePrice$Bathroom, HousePrice$Bedroom2): Chi-squared
```

```
##
##  Pearson's Chi-squared test
##
## data:  HousePrice$Bathroom and HousePrice$Bedroom2
## X-squared = 20915, df = 88, p-value < 2.2e-16
```

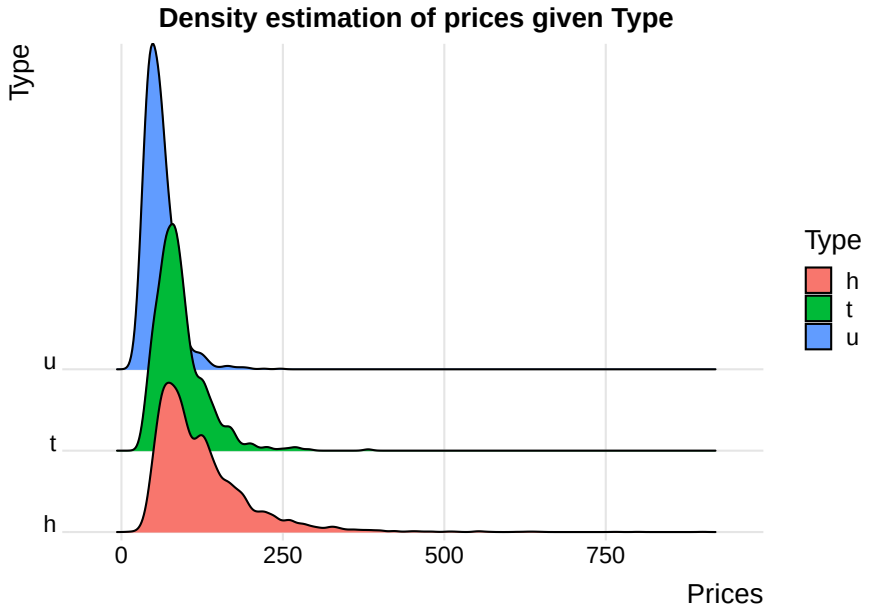
Note that χ^2 test is sometimes used in EDA.

4.2.2.3 One Numeric Variable and One Categorical Variable

From the modeling point of view, there are two different ways to assess the relationship between a categorical variable and a numerical variable. For example, a ridge plot can be used to visualize the distribution of house prices across the Type of houses.

```
ggplot(HousePrice, aes(x=Price/10000,y=Type,fill=Type))+
  geom_density_ridges_gradient(scale = 4) + theme_ridges() +
  scale_y_discrete(expand = c(0.01, 0)) +
  scale_x_continuous(expand = c(0.08, 0)) +
  labs(x = "Prices",y = "Type") +
  ggtitle("Density estimation of prices given Type") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## Picking joint bandwidth of 6.65
```



The ridge plot is a visual representation of ANOVA.

4.2.3 Three or More Variables

Visualizing the relationship between three or more variables can be challenging. One has to use visual design elements such as line, shape, negative/white space, volume, value, color, and texture — to represent the values of variables.

4.2.3.1 Use of Colors, Movement, and Point-size

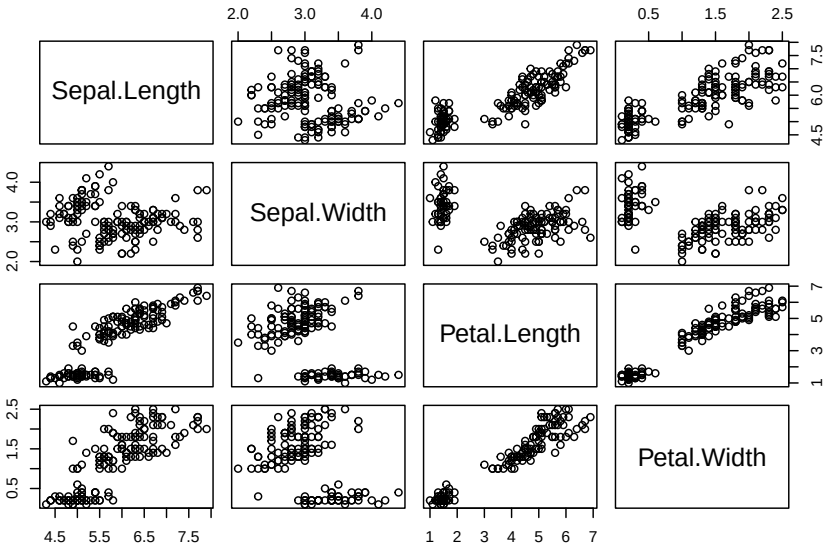
In the following example, color, movement, and point size represent `continent`, `time`, and `population size`, respectively. Therefore, it represents the complete relationship of 5 variables.

```
knitr::include_url("https://flo.uri.sh/visualisation/11871870/embed?auto=1")
```

4.2.3.2 Pairewised Relationship Between Variables

The pair-wise scatter plot **numerical variables** is the most commonly used in practice. We use the `Iris` data set as an example to show the pair-wise plot in the following.

```
pairs(iris[,-5])
```



The above enhanced pair-wise scatter plot provides a pair-wise comparison between the four numerical variables across the three species (categorical variable).

4.3 Roles of Visualization in EDA

Information visualization displays information in a visual format that makes insights easier to understand for human users. The information in data is usually visualized in a pictorial or graphical form such as charts, graphs, lists, maps, and comprehensive dashboards that combine these multiple formats.

4.3.1 Data Visualization

The primary objective of **data visualization** is to clearly communicate what the data says, help explain trends and statistics, and show patterns that would otherwise be impossible to see. **Data visualization** is used to make consuming,

interpreting, and understanding data as simple as possible, and to make it easier to derive insights from data.

4.3.2 Visual Analytics

Visual analytics is an emerging area in analytics. It is more than visualization. **Interactive** exploration and **automatic** visual manipulation play a central role in visual analytics.

Visual analytics does the **heavy lifting*"** with data, by using a variety of tools and technologies — machine learning and mathematical algorithms, statistical models, cutting-edge software programs, etc — to identify and reveal patterns and trends. It prepares the data for the process of data visualization, thereby enabling users to examine data, understand what it means, interpret the patterns it highlights, and help them find meaning and gain useful insights from complex data sets.

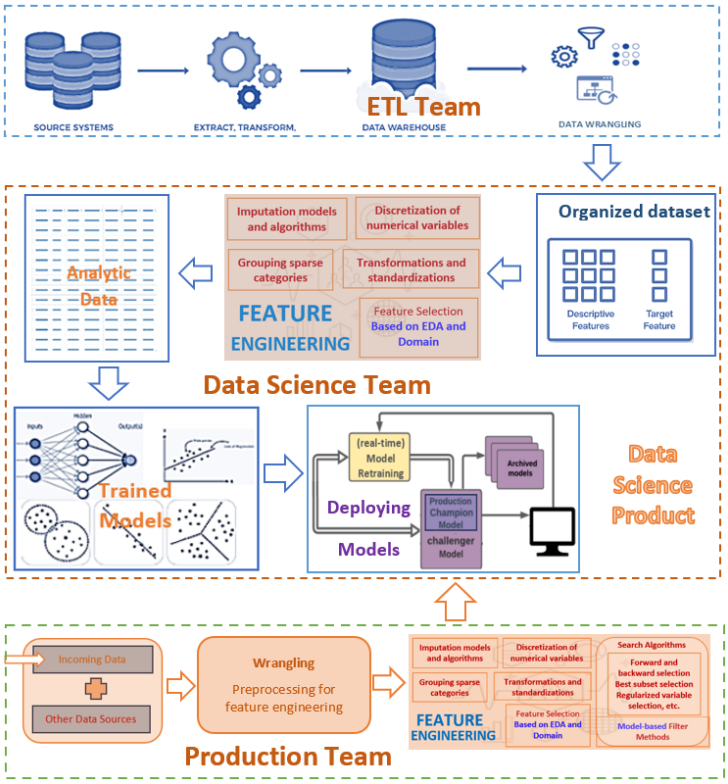
In other words, using visual analytic methods and techniques can enhance (data) visualization and improve the performance of analysis and modeling. Interactive visualization technology enables the exploration of data via the manipulation of chart images, with the color, brightness, size, shape, and motion of visual objects representing aspects of the data set being analyzed. The following is such an example (https://vizhub.healthdata.org/cod/).

```
knitr::include_app("https://vizhub.healthdata.org/cod/")
```

Chapter 5

EDA for Feature Engineering

Recall the following workflow of the data science project.



We have introduced the techniques and tools for exploratory data analysis. This note provides an example to show how to use EDA based on the data set used in the example.

5.1 Description of Data

A population of women who were at least 21 years old, of Pima Indian heritage, and living near Phoenix, Arizona, was tested for diabetes according to World Health Organization criteria. The data were collected by the US National Institute of Diabetes and Digestive and Kidney. The objective of the data set is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the data set. Several constraints were placed on the selection of these instances from a larger database.

There are two versions of the data available in the public domain. This case study uses the version that contains the missing values. The total number of records in this data set is 768. The data set consists of 9 variables including the response variable with the name `diabetes`. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on. A detailed description of the variables is given below

`pregnant`: Number of times pregnant

`glucose`: Plasma glucose concentration 2 hours in an oral glucose tolerance test

`pressure`: Diastolic blood pressure (mm Hg)

`triceps`: Triceps skin fold thickness (mm)

`insulin`: 2-Hour serum insulin (mu U/ml)

`mass`: Body mass index (weight in kg/(height in m)^2)

`pedigree`: Diabetes pedigree function

`age`: Age (years)

`diabetes`: outcome class variable ('neg' or 'pos')

A copy of this publicly available data is stored at https://pengdsci.github.io/datasets/PimaDiabetes/PimaIndiansDiabetes2.csv.

```
PimaDiabetes = read.csv("https://pengdsci.github.io/datasets/PimaDiabetes/PimaIndiansDi
```

5.2 EDA for Feature Engineering

We have introduced several techniques and methods in an earlier note. We will use some EDA techniques and methods as needed for this data and subsequent

modeling.

We first scan the entire data set and determine the EDA tools to use for feature engineering.

```
summary(PimaDiabetes)
```

```
##     pregnant         glucose         pressure         triceps         insulin            mass
##  Min.   : 0.000   Min.   : 44.0   Min.   : 24.00   Min.   : 7.00   Min.   : 14.00   Min.   :18
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.:22.00   1st Qu.: 76.25   1st Qu.:27
##  Median : 3.000   Median :117.0   Median : 72.00   Median :29.00   Median :125.00   Median :32
##  Mean   : 3.845   Mean   :121.7   Mean   : 72.41   Mean   :29.15   Mean   :155.55   Mean   :32
##  3rd Qu.: 6.000   3rd Qu.:141.0   3rd Qu.: 80.00   3rd Qu.:36.00   3rd Qu.:190.00   3rd Qu.:36
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00   Max.   :846.00   Max.   :67
##                   NA's   :5       NA's   :35       NA's   :227     NA's   :374      NA's   :11
##       age          diabetes
##  Min.   :21.00   Length:768
##  1st Qu.:24.00   Class :character
##  Median :29.00   Mode  :character
##  Mean   :33.24
##  3rd Qu.:41.00
##  Max.   :81.00
##
```

5.2.1 Missing Values - Imputation

The above summary table indicates that feature variables `glucose`, `pressure`, `triceps`, `insulin`, and `mass` have missing values. `insulin` has nearly 50% missing values. `triceps` has 227 missing values. The other three variables have a very low percentage of missing values.

5.2.1.1 Missing Value vs No Value

Missing value means that the information is available but not collected while no value means that the value does not exist.

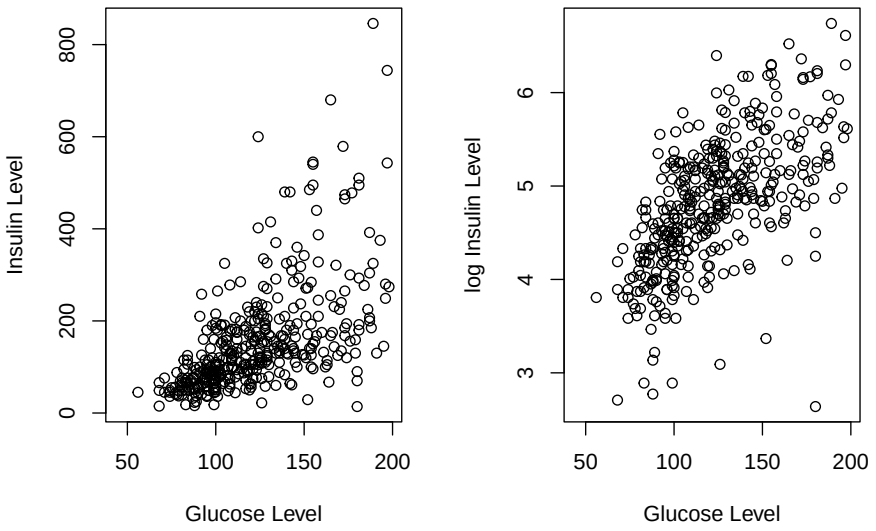
Replacing the missing values with proxy values (imputation) or deleting them from the data are the ways of handling missing values. Most software programs *automatically delete all records with missing components* from the data before modeling if the missing value issue is not handled.

`no-value` should be never imputed in the data processing. The ways of handling `no value` is to either drop all records with `no value` components or the feature variables that have `no values`. The former will change the study population and the latter will lead to a loss of information.

5.2.1.2 Glucose Tolerance (glucose) vs 2-Hour Serum Insulin (insulin)

Both `fasting insulin test` and `glucose tolerance test` are used in diabetes diagnosis, therefore, variables `glucose` and `insulin` are correlated. Since nearly 50% of patients did not do the insulin test. Therefore, we can use `glucose` to impute the missing values in `insulin`. We first look at the correlation between the two variables based on the complete data.

```
par(mfrow = c(1,2))
plot(PimaDiabetes$glucose, PimaDiabetes$insulin, xlab = "Glucose Level", ylab = "Insuli
plot(PimaDiabetes$glucose, log(PimaDiabetes$insulin), xlab = "Glucose Level", ylab = "
```



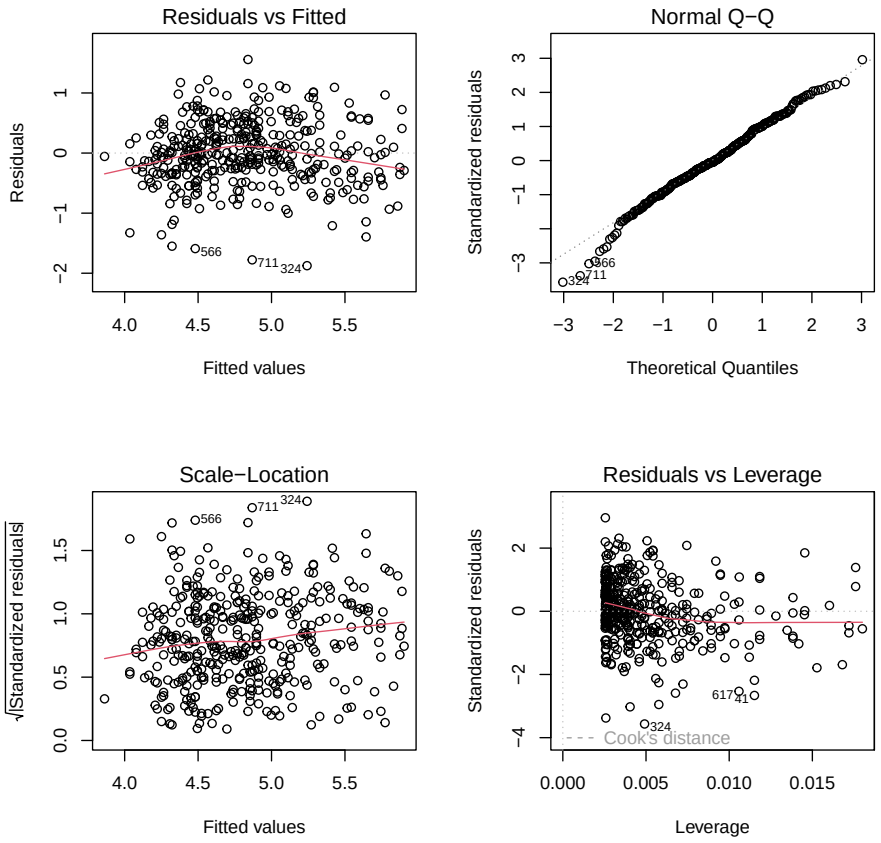
The scatter plot shows that the logarithm of the insulin level and the glucose level are highly linearly correlated. We can use this relationship to impute the logarithm of insulin level based on the no-missing glucose level. Since we will use this data set to build predictive models, the logarithm of insulin will be used directly in the subsequent models and algorithms.

```
impute.insulin.lm = lm(log(insulin[-446]) ~ glucose[-446], data = PimaDiabetes)
summary(impute.insulin.lm)
```

```
##
## Call:
## lm(formula = log(insulin[-446]) ~ glucose[-446], data = PimaDiabetes)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.87469 -0.31478 -0.02521  0.33826  1.55711
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.0588150  0.1095012   27.93   <2e-16 ***
## glucose[-446] 0.0143630  0.0008673   16.56   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5269 on 390 degrees of freedom
##   (   375    )
## Multiple R-squared:  0.4129, Adjusted R-squared:  0.4114
## F-statistic: 274.3 on 1 and 390 DF,  p-value: < 2.2e-16
```

```
par(mfrow = c(2,2))
plot(impute.insulin.lm)
```



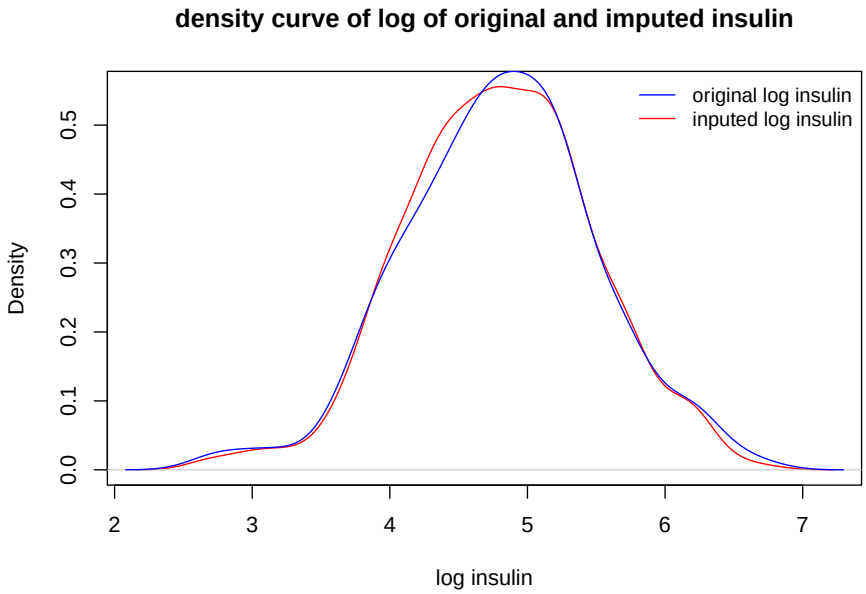
Next, we use the following linear regression to impute the missing values in `insulin`.

```
glucose = PimaDiabetes$glucose
impute.log.insulin = log(PimaDiabetes$insulin)
n=length(impute.log.insulin)
for (i in 1:n){
  if (is.na(impute.log.insulin[i]) == TRUE && is.na(glucose[i]) == FALSE) impute.log.in
}
```

Visual comparison of the distribution between the original `insulin` and the imputed `insulin`.

```
den.orig.insulin = density(na.omit(log(PimaDiabetes$insulin)))
den.impute.insulin = density(na.omit(impute.log.insulin))
plot(den.impute.insulin, xlab="log insulin", main = "density curve of log of original a
     col = "red")
```

```
lines(den.orig.insulin, col = "blue")
legend("topright", c("original log insulin", "inputed log insulin"), col = c("blue", "red"), lty
```



The above density curves show that distributions of the imputed log insulin and original log insulin levels are close to each other.

```
PimaDiabetes$impute.log.insulin = impute.log.insulin
```

5.2.1.3 Triceps Skinfold Thickness (`triceps`) vs Body Mass Index (`mass`)

Clinical variables `triceps` (triceps skin-fold thickness, see the following figure to see how it is measured) and `mass` (body mass index) are clinically correlated.

`triceps` has nearly 30% missing values and `mass` has a few missing values. We can use the information in `mass` to impute the missing values in triceps - single imputation with a linear regression model. To perform imputation,

1. fit a linear regression model with `triceps` being the response and `mass` as the predictor.
2. use the above fitted regression to predict `triceps` on non-missing `mass`.

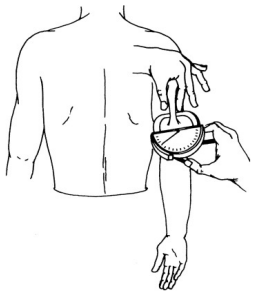


Figure 5.1: Figure 1. Measurement of triceps skinfold using a Lange caliper. With the subject's arm in a relaxed position, the skinfold is picked with thumb and index fingers at the midpoint of the arm.

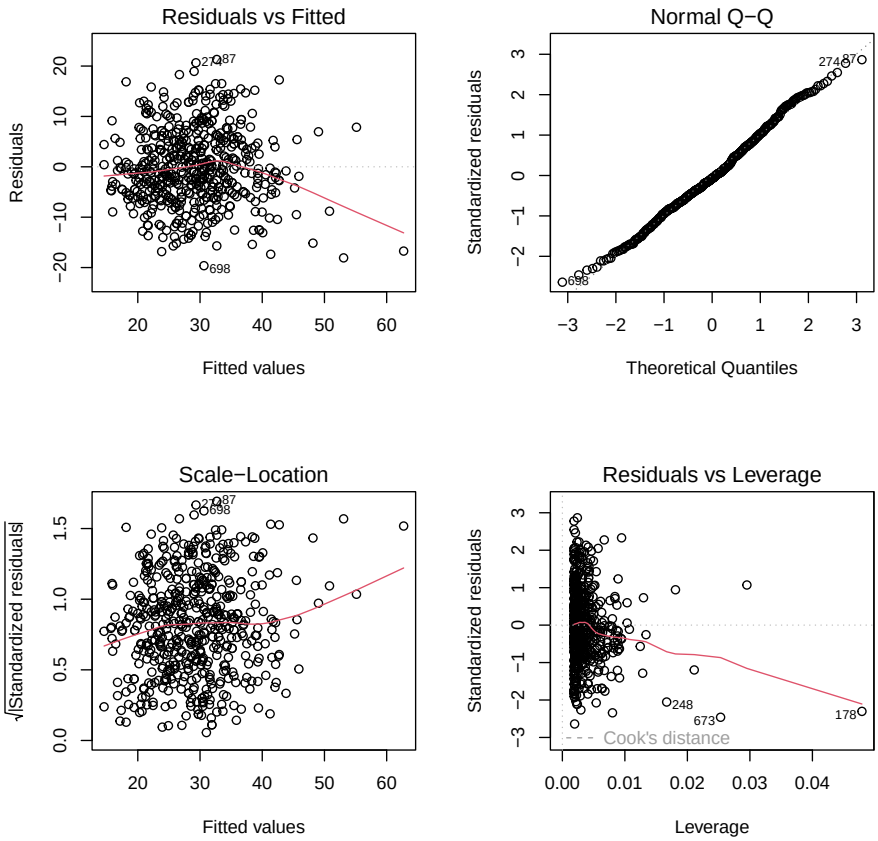
3. impute the missing value in `triceps` with the *predicted* `triceps`.

Note that in R, records with missing components will be automatically deleted in the modeling process.

```
impute.lm = lm(triceps[-580] ~ mass[-580], data = PimaDiabetes)
summary(impute.lm)
```

```
##
## Call:
## lm(formula = triceps[-580] ~ mass[-580], data = PimaDiabetes)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.6294  -4.9225  -0.4862   5.0930  21.3029
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.34070    1.56901  -2.129   0.0337 *
## mass[-580]   0.98464    0.04669  21.087   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.442 on 536 degrees of freedom
##   (   229    )
## Multiple R-squared:  0.4534, Adjusted R-squared:  0.4524
## F-statistic: 444.7 on 1 and 536 DF,  p-value: < 2.2e-16
```

```
par(mfrow = c(2,2))
plot(impute.lm)
```



The above fitted regression line will be used to **impute** the missing values in `triceps` in the following.

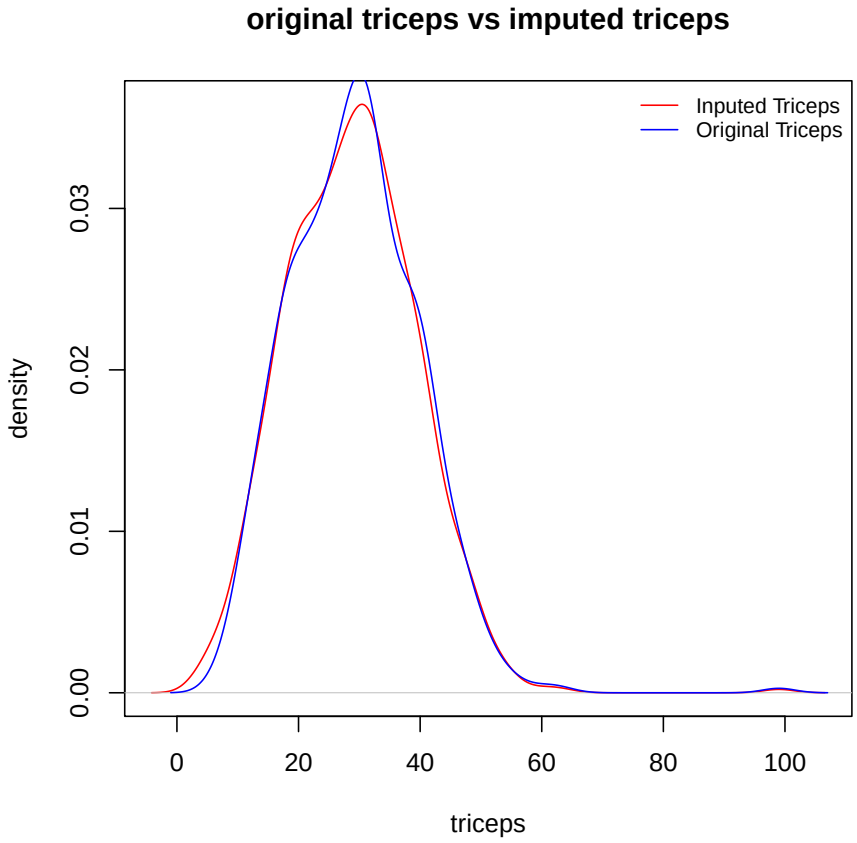
```
mass = PimaDiabetes$mass
impute.triceps = PimaDiabetes$triceps
n=length(impute.triceps)
for (i in 1:n){
  if (is.na(impute.triceps[i]) == TRUE && is.na(mass[i]) == FALSE) impute.triceps[i] = sum(coef(i
}
```

```
PimaDiabetes$impute.triceps = impute.triceps
```

Next, we check whether the missing values in `triceps` were appropriately imputed.

We look at the density curves of `impute.triceps` and the original `triceps` to see the performance of the imputation and whether a discretization is needed.

```
den.tri = density(na.omit(PimaDiabetes$triceps))
den.imput.tri = density(na.omit(PimaDiabetes$impute.triceps))
plot(den.imput.tri, col = "red", xlab = "triceps", ylab = "density", main = "original t
lines(den.tri, col = "blue")
legend("topright", c("Inputed Triceps", "Original Triceps"), col=c("red", "blue"), lty
```



The above density curves indicate that

- the two distributions are almost identical, and
- both distributions are almost symmetric (except for one outlier in the original data).

Since the missing values in `triceps` were appropriately imputed, we next add the `impute.triceps` to the original data frame and drop the original `triceps`.

To close this imputation section, we re-organize the data set by dropping the

original variables and keeping the imputed variables. At the same time, we also delete all records with missing components.

```
PimaDiabetes = na.omit(PimaDiabetes[, c("pregnant", "glucose", "pressure", "mass" , "pedigree" ,
```

5.2.2 Assess Distributions

This subsection focuses on the potential discretization of continuous variables and grouping sparse categories of category variables based on their distribution.

5.2.2.1 Discretizing Continuous Variables

The above pairwise scatter plot shows that `glucose`, `pressure`(diastolic reading), and `age` are usually discretized in the clinical study. We will use the clinical standard and practices to discretize these variables

According to Medical News Today (https://www.medicalnewstoday.com/articles/a1c-chart-diabetes-numbers#a-1-c-chart). The glucose levels < 117, $[117, 137]$, > 137 indicate normal, pre-diabetes, and diabetes.

According to National Diabetes Statistics Report (https://www.cdc.gov/media/releases/2017/p0718-diabetes-report.html#:~:text=Rates%20of%20diagnosed%20diabetes%20increased,older%2C%2025%20percent%20had%20diabetes), rates of diagnosed diabetes increased with age. Among adults ages 18-44, 4 percent had diabetes. Among those ages 45-64 years, 17 percent had diabetes. And among those ages 65 years and older, 25 percent had diabetes.

According to The Seventh Report of the Joint National Committee on Prevention, Detection, Evaluation, and Treatment of High Blood Pressure (2003 Guideline, https://www.nhlbi.nih.gov/files/docs/guidelines/express.pdf), The normal diastolic pressure is less than 80 mm Hg, at risk diastolic reading is between 80 mm Hg and 90 mm Hg, abnormal (hypertension) diastolic reading is higher than 90 mm Hg.

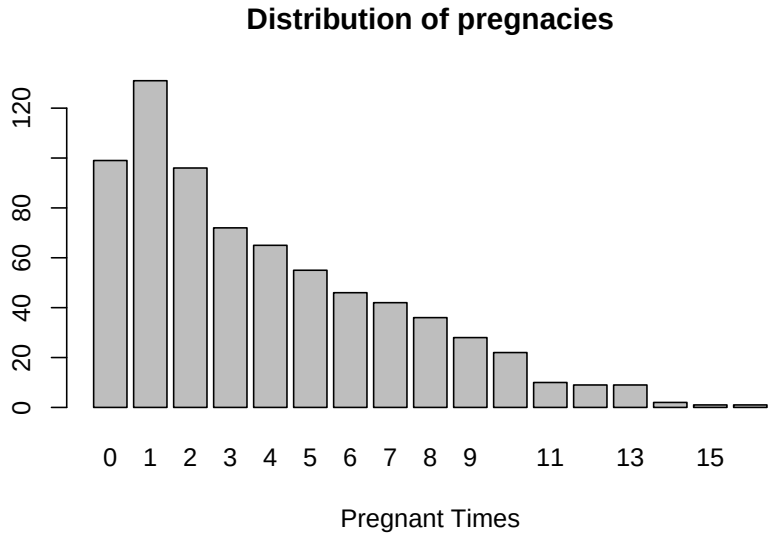
We will discretize these three variables for future models and algorithms.

```
PimaDiabetes$grp.glucose <- ifelse(PimaDiabetes$glucose < 117, '(0, 117)',
               ifelse(PimaDiabetes$glucose > 137, '> 137', '[117,137]'))

PimaDiabetes$grp.diastolic <- ifelse(PimaDiabetes$pressure < 80, '(0, 80)',
               ifelse(PimaDiabetes$pressure > 90, '> 90', '[80,90]'))

PimaDiabetes$grp.age <- ifelse(PimaDiabetes$age <= 44, '[21, 44]',
               ifelse(PimaDiabetes$age >= 65, '65+', '[45, 64]'))
```

5.2.2.2 Grouping Sparse Categories

The number of times pregnant `pregnant` is a discrete numerical variable. We could also consider it as an ordinal categorical variable.

```
pregnancy = table(PimaDiabetes$pregnant)
barplot(pregnancy, main = "Distribution of pregnacies", xlab = "Pregnant Times")
```



There are a few sparse categories in the variable, we decide to group this variable in the following: 0, 1, 2, 3-4, 5-7, 8+.

```
PimaDiabetes$grp.pregnant <- ifelse(PimaDiabetes$pregnant == 0, '0',
                                    ifelse(PimaDiabetes$pregnant == 1, '1',
                                       ifelse(PimaDiabetes$pregnant == 2, '2',
                                          ifelse((PimaDiabetes$pregnant == 3 | PimaDiabetes
                                             ifelse((PimaDiabetes$pregnant == 5 | PimaDiabe
                                                ifelse(PimaDiabetes$pregnant >= 8, '8+', "I
```

As the last step, we only keep those variables to used in the subsequent modeling.

```
var.names = c("mass", "pedigree", "impute.log.insulin", "impute.triceps", "grp.glucose"
PimaDiabetes = PimaDiabetes[, var.names]
```

5.2.2.3 Save Analytic Dataset

The final analytic data should be saved as a permanent data for the subsequent analysis and modeling and upload the saved data set to HitHub data repository

for an easy access in the future.

```
write.csv(PimaDiabetes, "C:\\Users\\75CPENG\\OneDrive - West Chester University of PA\\Desktop\\c
```

The above csv file is also uploaded to the GitHub data repository at https://pengdsci.github.io/STA551/w03/AnalyticPimaDiabetes.csv.

5.2.3 Pairwise Association

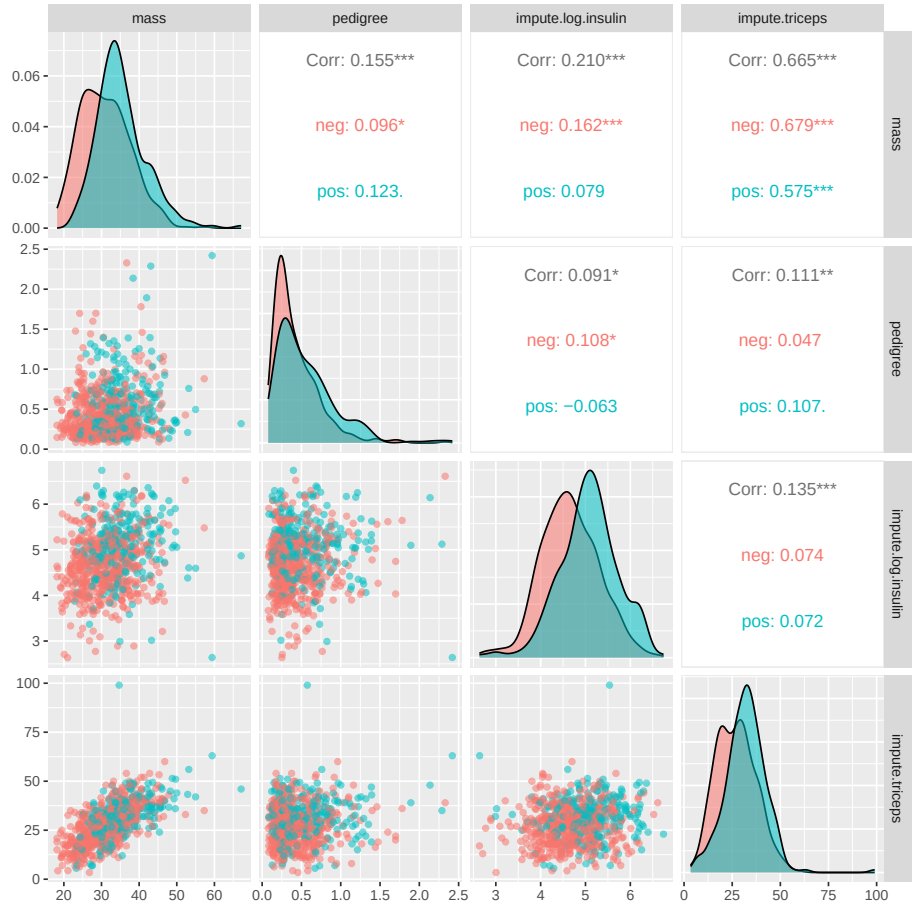
Depending on the types of variables, there are three different combinations of two variables: two numeric variables, two categorical variables, one numeric variable, and one categorical variable. We will assess the association between two variables graphically based on the above three scenarios.

5.2.3.1 Two Numeric Variables

The best visual tool for assessing pairwise linear association between two numeric variables is a pair-wise scatter plot. The pair-wise scatter plot and its variants are available in several different R packages.

```
ggpairs(PimaDiabetes,                    # Data frame
        columns = 1:4,          # Columns
        aes(color = diabetes,  # Color by group (cat. variable)
            alpha = 0.5))      # Transparency
```

```
## plot: [1,1] [====>----------------------------------------------------------------------------
## plot: [1,2] [==========>----------------------------------------------------------------------
## plot: [1,3] [===============>-----------------------------------------------------------------
## plot: [1,4] [====================>----------------------------------------------------------
## plot: [2,1] [=========================>-----------------------------------------------------
## plot: [2,2] [==============================>----------------------------------------------
## plot: [2,3] [====================================>----------------------------------------
## plot: [2,4] [==========================================>------------------------------------
## plot: [3,1] [================================================>------------------------------
## plot: [3,2] [======================================================>----------------------
## plot: [3,3] [============================================================>----------------
## plot: [3,4] [==================================================================>----------
## plot: [4,1] [========================================================================>----
## plot: [4,2] [==============================================================================>
## plot: [4,3] [================================================================================
## plot: [4,4] [================================================================================
```



The off-diagonal plots and numbers indicate the correlation between the pairwise numeric variables. As expected, triceps and mass are significantly correlated. Other paired variables have weak correlations.

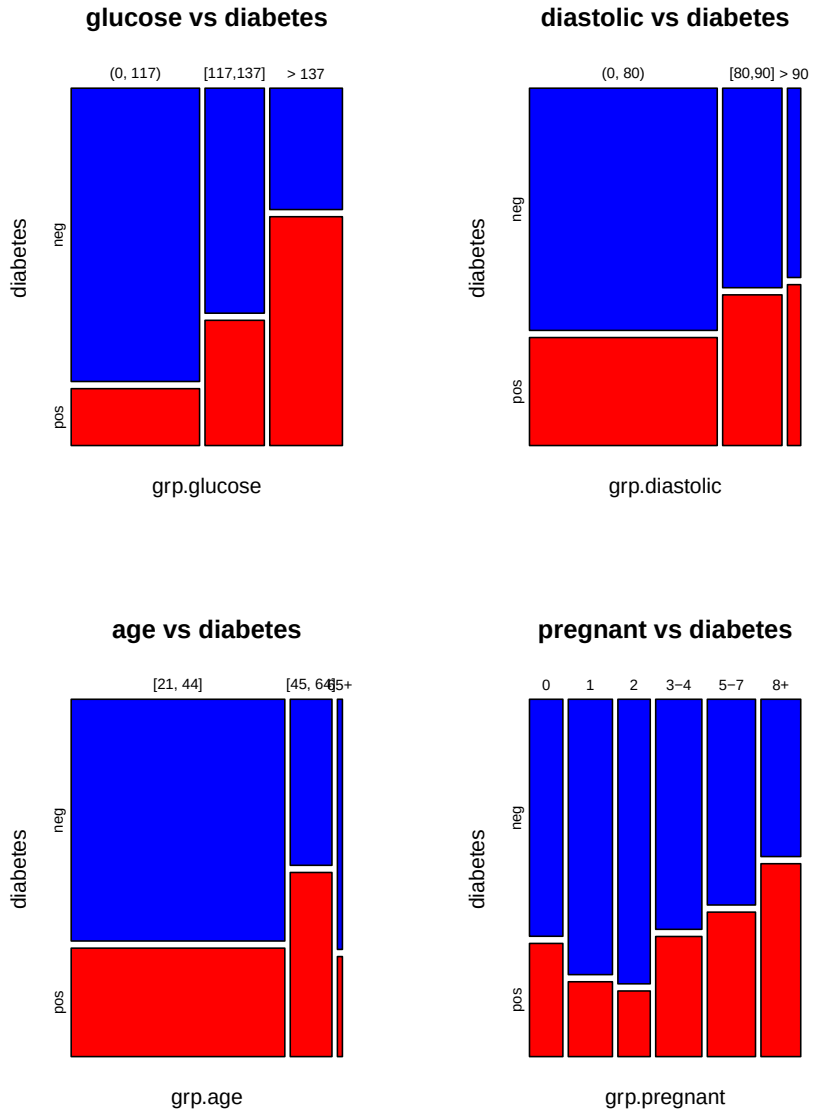
The main diagonal stacked density curves show the potential difference in the distribution of the underlying numeric variable in diabetes and diabetes-free groups. This means that the stacked density curves show the relation between numeric and categorical variables. These stacked density curves are not completely overlapped indicating somewhat correlation between each of these numeric variables and the binary response variable.

Because of the above interpretation between numeric variables and the binary response variable, we will not open a new subsection to illustrate the relationship between a numeric variable and a categorical variable.

5.2.3.2 Two Categorical Variables

Mosaic plots are convenient to show whether two categorical variables are dependent. In EDA, we are primarily interested in whether the response (binary in this case) is independent of categorical variables. Those categorical variables that are independent of the response variable should be excluded in any of the subsequent models and algorithms.

```
par(mfrow = c(2,2))
mosaicplot(grp.glucose ~ diabetes, data=PimaDiabetes,col=c("Blue","Red"), main="glucose vs diabet
mosaicplot(grp.diastolic ~ diabetes, data=PimaDiabetes,col=c("Blue","Red"), main="diastolic vs di
mosaicplot(grp.age ~ diabetes, data=PimaDiabetes,col=c("Blue","Red"), main="age vs diabetes")
mosaicplot(grp.pregnant ~ diabetes, data=PimaDiabetes,col=c("Blue","Red"), main="pregnant vs diab
```

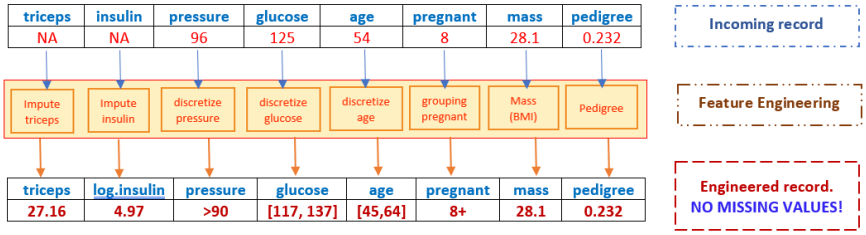


The top two mosaic plots demonstrate the positive association between `glucose` levels and `diastolic` readings. The bottom two mosaic plots also show that diabetes is not independent of `age` and `pregnant` times because the proportion of diabetes cases in individual categories is not identical.

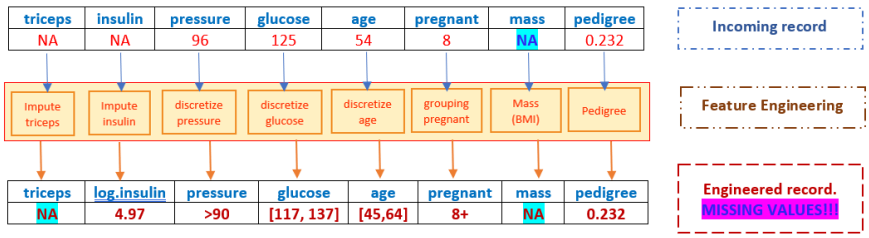
5.3 Concluding Remarks

Begin With the End in Mind

We **begin** with records that may have missing values and **end** up with an engineered record to be fed into models and algorithms!



However, if the incoming record has a missing value in mass, the resulting engineered record cannot be fed to the candidate models and algorithms since it has a missing value (most programs will automatically delete this record), hence, there will be no predicted value for the response!



Therefore, we need a replacement imputation before any model-based imputation. This is practically important since it will prevent the potential failure of the dynamic prediction system!

Chapter 6

Multiple Linear Regression

We discussed the relationship between variables in the previous two modules. The continuous variable with a normal distribution is called the response (dependent) variable and the other variable is called the explanatory (predictor, independent, or risk) variable. If the predictor variable is a factor variable, the model is called the ANOVA model which focuses on comparing the means across all factor levels. If the predictor variable is **continuous**, the model is called simple linear regression (SLR). Note that all predictor variables are assumed to be non-random.

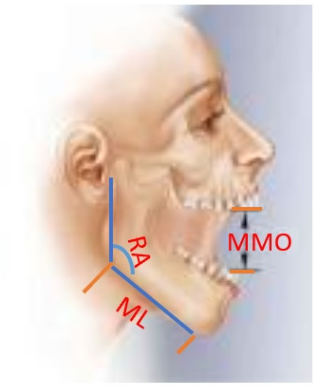
6.1 The Practical Question

Maximum mouth opening (MMO) is also an important diagnostic reference for dental clinicians as a preliminary evaluation. Establishing a normal range for MMO could allow dental clinicians to objectively evaluate the treatment effects and set therapeutic goals for patients performing mandibular functional exercises.

To study the relationship between maximum mouth opening and measurements of the lower jaw (mandible). A researcher randomly selected a sample of 35 subjects and measured the dependent variable, maximum mouth opening (MMO, measured in mm), as well as predictor variables, mandibular length (ML, measured in mm), and angle of rotation of the mandible (RA, measured in degrees) of each of the 35 subjects.

The question is the maximum mouth opening (MMO) is determined by **two variables simultaneously**. We want to assess how these two variables (ML and RA) impact MMO **simultaneously**.

If we pick one predictor variable at a time, ML, to build a simple linear regression model and ignore the other predictor variable (RA), you only get the



MMO = maximum mouth opening

ML = mandible length

RA = angle of rotation of the mandible

Figure 6.1: MMO, ML, and RA

MMO (Y)	ML (X_1)	RA (X_2)	MMO (Y)	ML (X_1)	RA (X_2)
52.34	100.85	32.08	50.82	90.65	38.33
51.90	93.08	39.21	40.48	92.99	25.93
52.80	98.43	33.74	59.68	108.97	36.78
50.29	102.95	34.19	54.35	91.85	42.02
57.79	108.24	35.13	47.00	104.30	27.20
49.41	98.34	30.92	47.23	93.16	31.37
53.28	95.57	37.71	41.19	94.18	27.87
59.71	98.85	44.71	42.76	89.56	28.69
53.32	98.32	33.17	51.88	105.85	31.04
48.53	92.70	31.74	42.77	89.29	32.78
51.59	88.89	37.07	52.34	92.58	37.82
58.52	104.06	38.71	50.45	98.64	33.36
62.93	98.18	43.89	43.18	83.70	31.93
57.62	91.01	41.06	41.99	88.46	28.32
65.64	96.98	41.92	39.45	94.93	24.82
52.85	97.85	35.25	38.91	96.81	23.88
64.43	96.89	45.11	49.10	93.13	36.17
57.25	98.35	39.44			

Figure 6.2: Dental Data for the multiple linear regression model (MLR)

marginal relationship between MMO and ML since you implicitly assume that the relationship between MMO and ML will not be impacted by RA. This implicit assumption is, in general, incorrect. We need to consider all predictor variables at the same time. This is the motivation for studying multiple linear regression (MLR).

6.2 The Process of Building A Multiple Linear Regression Model

The previous motivation example involves two continuous predictor variables. In real-world applications, it is common to have many predictor variables. Predictor variables are also assumed to be non-random. They could be categorical, continuous, or discrete. In a specific application, you may have a set of categorical, continuous, and discrete predictor variables in one data set.

6.2.1 Assumptions of MLR

There are several assumptions of multiple linear regression models.

- The response variable is a normal random variable and its mean is influenced by explanatory variables but not the variance.
- The explanatory variables are assumed to be non-random.
- The explanatory variables are assumed to be uncorrelated to each other.
- The functional form of the explanatory variables in the regression model is correctly specified.
- The data is a random sample taken independently from the study population with a specified distribution.

Some of these assumptions will be used directly to define model diagnostic measures. The idea is to assume all conditions are met (at least temporarily) and then fit the model to the data set.

6.2.2 The Structure of MLR

Assume that there are p predictor variables $\{x_1, x_2, \cdots, x_p\}$, the first-order linear regression is defined in the following form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon$$

β_0 is the intercept, $\beta_1, \beta_2, \cdots, \beta_p$ are called slope parameters. if $\beta_i = 0$, the associated predictor variable x_i is uncorrelated with response vararible y. If $\beta_i > 0$, then y and x_i are positively correlated. In fact, β_1 is the increment of y as x_i increases one unit and other predictors remain unchanged.

The response variable is assumed to be a normal random variable with constant variance. If the first-order linear regression function is correct, then

$$y \to N(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p, \sigma^2).$$

This also implies that $\epsilon \to N(0, 1)$. The residual of each data point can be estimated from the data with an assumed linear regression model.

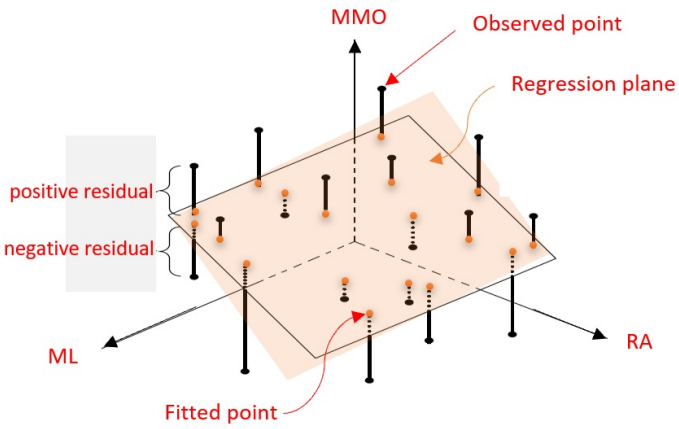


Figure 6.3: Illustrative regression plane: MMO vs ML and RA

For ease of illustration, let's consider the case of the MLR with two predictor variables in the motivation example.

$$MMO = \beta_0 + \beta_1 ML + \beta_2 RA + \epsilon$$

is the first-order linear regression model. The following figure gives the graphical annotations of the fundamental concepts in linear regression models. This is a generalization of the regression line (see the analogous figure in the previous module for the simple linear regression model).

Since MMO is a normal random variable with constant variance, $MMO \to N(\beta_0 + \beta_1 ML + \beta_2 RA, \sigma^2)$, or equivalently, $\epsilon \to N(0, \sigma^2)$. The residuals are defined to be the directional vertical distances between the observed points and the regression plane.

In some practical applications, we may need **the second-order** linear regression model to reflect the actual relationship between predictor variables and the response variable. For example,

$$MMO = \alpha_0 + \alpha_1 ML + \alpha_2 RA + \alpha_3 ML^2 + \alpha_4 RA^2 + \alpha_5 ML \times RA + \epsilon$$

is called (the second-order) linear regression model. With the second-order terms in the regression function, we obtain the regression surface as shown in Figure.

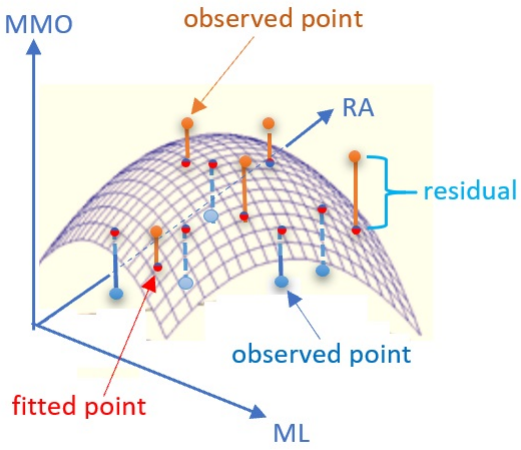


Figure 6.4: Illustrative regression surface: MMO vs ML and RA

If the second-order linear regression is appropriate, then $\epsilon \rightarrow N(0, \sigma^2)$ and $E[MMO] = \alpha_0 + \alpha_1 ML + \alpha_2 RA + \alpha_3 ML^2 + \alpha_4 RA^2 + \alpha_5 ML \times RA$. The residuals of the second-order linear regression model are defined to be the directional distance between the observed points and the regression surface.

6.2.3 More on Model Specifications

In the above section, we introduced both first- and second-order polynomial regression models. In general, it is not common to use high-order polynomial regression models in real-world applications.

- **Interaction effect** - it is common to include interaction terms (i.e., the cross product of two or more predictor variables) in the multiple linear regression models when the effect of one variable on the response variable is dependent on the other predictor variable. In other words, the interaction terms capture the **joint effect** of predictor variables. **It is rare to have third-order or higher-order interaction terms in a regression model**.
- **Dummy variables** - All categorical predictor variables are automatically converted into dummy variables (binary indicator variables). If categorical variables in the data are numerically coded, we have to turn these numerically coded variables into factor variables in the regression model.

- **Discretization and Regrouping** - Discretizing numerical predictor variables and regrouping categorical or discrete predictor variables are two basic pre-process procedures that are actually very common in many practical applications.
  - Sometimes these two procedures are required to satisfy certain model assumptions. For example, if a categorical variable has a few categories that have less than 5 observations, the resulting p-values based on certain hypothesis tests will be invalid. In this case, We have to regroup some of the categories in **meaningful ways** to resolve the **sparsity** issues in order to obtain valid results.
  - In many other applications, we want the model to be easy to interpret. Discretizing numerical variables is common. For example, we can see grouped ages and salary ranges in different applications.

6.2.4 Estimation of Regression Coefficients

A simple and straightforward method for estimating the coefficients of linear regression models is to minimize the sum of the squared residuals - least square estimation (LSE). To find the LSE of the regression coefficients, we need to

- choose the (first-order, second-order, or even high-order) regression function (see 3D hyper-plane or hyper-surface in the above two figures as examples).
- find the distances between the observed points and the hyper-plane (or hyper-surface). These distances are the residuals of the regression - which is dependent on the regression coefficients.
- calculate the sum of squared residuals. This sum of the residuals is still dependent on the regression coefficients.
- find the values for the regression coefficients that minimize the sum of the squared residuals. These values are called the least square estimates (LSEs) of the corresponding regression coefficients.

R function **lm()** implements the above LSE algorithm to find the regression coefficients. We have used this function in ANOVA and simple linear regression models.

6.2.5 Model Diagnostics

Unlike simple linear regression models, the primary assumptions of the regression model focus on the normal distribution of the response variable and the correct regression function. For multiple linear regression models, we need to impose a couple of assumptions in addition to those in the simple linear regression models

- **Residual Diagnostics**

One of the fundamental assumptions of linear regression modeling is that the response variable is normally distributed with a constant variance. This implies $\epsilon \to N(0, \sigma^2)$.

After obtaining the LSE of the regression coefficients, we can estimate the residuals and use these estimated residuals to detect the potential violations of the normality assumption of the response variable. To be more specific, we consider the first-order polynomial regression, the estimated residual of i-th observation is defined to be $e_i = MMO - \hat{\beta}_0 + \hat{\beta}_1 ML + \hat{\beta}_2 RA$

If there is no violation of the normality assumption, we would expect the following residual plot and Q-Q plot.

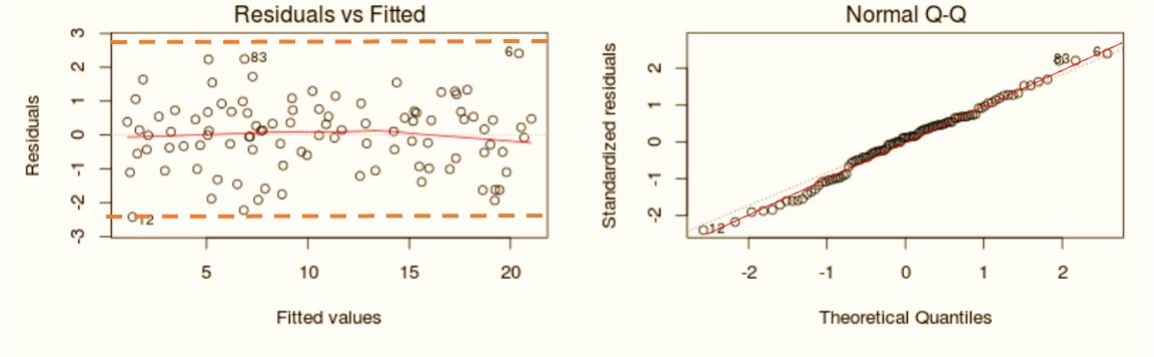


Figure 6.5: Good residual plot and normal Q-Q plot

Some of the commonly seen poor residual plots represent different violations of various assumptions. We can try to use various transformations (such Box-Cox power transformations) of the response variable to correct the issue.

- **Multicollinearity**

Some of the predictor variables are linearly correlated. The consequence of multi-collinearity causes to unstable LSE of the regression coefficients (i.e., the LSEs of the regression coefficients are sensitive to a small change in the model). It also reduces the precision of the estimate coefficients and, hence, the p-values are not reliable.

Multicollinearity affects the coefficients and p-values, but it does not influence the predictions, precision of the predictions, and the goodness-of-fit statistics. If our primary goal is to make predictions, we don't need to understand the role of each independent variable and we don't need to reduce severe multicollinearity.

If the primary goal is to perform association analysis, we need to reduce collinearity since both LSE and p-values are the keys to association analysis.

To detect multicollinearity, we can use the variance inflation factor (VIF) to

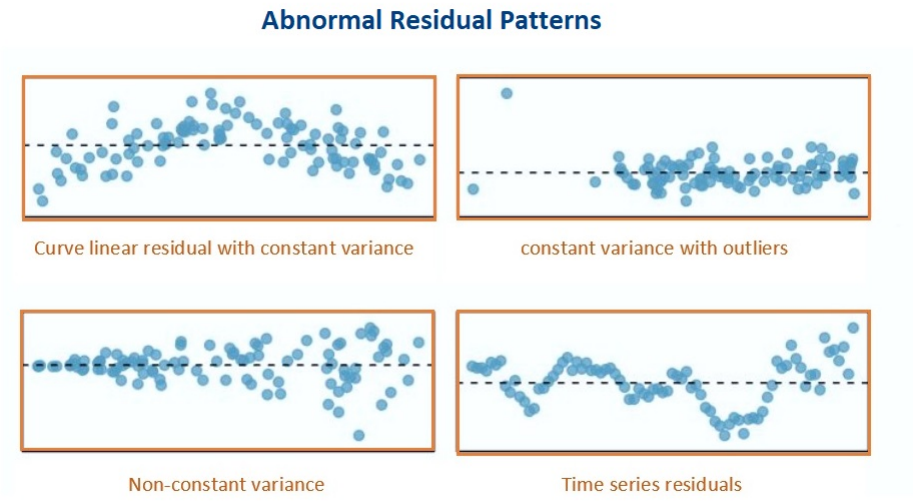


Figure 6.6: Poor residual plots representing various violations of the model assumptions

inspect the multicollinearity of the individual predictor variable. There are some different methods to reduce multicollinearity. Centering predictor variables is one of them and works well sometimes. Some other advanced modeling-based methods are covered in more advanced courses.

6.2.6 Goodness-of-fit and Variable Selection

There several different goodness-of-fit measures are available for the linear regression model due to the assumption of the normality assumption of the response variable.

- **Coefficient of Determination**

We only introduce **the coefficient of determination** R^2 which measures the percentage of variability within the -values that can be explained by the regression model. In simple linear regression models, **the coefficient of determination** R^2 is simply the square of the sample Pearson correlation coefficient.

- **Statistical Significance and Practical Importance**

A small p-value of the significant test for a predictor variable indicates the variable is statistically significant but may not be practically important. On the other hand, some practically important predictor variables may not achieve statistical significance due to the limited sample size. In the practical applications, **we may want to include some of the practically important predictor variables in the final model regardless of their statistical significance**.

- **Model Selection**

One of the criteria for assessing the goodness-of-fit is the parsimony of the model. A parsimonious model is a model that accomplishes the desired level of explanation or prediction with as few predictor variables as possible. There are generally two ways of evaluating a model: Based on predictions and based on goodness of fit on the current data such as R^2 and some likelihood-based measures.

R has an automatic variable selection procedure, **step()**, which uses the goodness-of-fit measure AIC (Akaike Information Criterion) which is not formally introduced in this class due to the level of mathematics needed in the definition, but we can still use it to perform the automatic variable selection. This tutorial gives detailed examples on how to use **step()** (link).

6.3 Case Study 1

We use the dental data in the motivation example for the case study.

```
MMO=c(52.34, 51.90, 52.80, 50.29, 57.79, 49.41, 53.28, 59.71, 53.32, 48.53, 51.59,
      58.52, 62.93, 57.62, 65.64, 52.85, 64.43, 57.25, 50.82, 40.48, 59.68, 54.35,
      47.00, 47.23,  41.19, 42.76, 51.88, 42.77, 52.34, 50.45, 43.18, 41.99, 39.45,
      38.91, 49.10)
##
ML=c(100.85, 93.08, 98.43, 102.95, 108.24, 98.34, 95.57, 98.85,98.32, 92.70, 88.89,
     104.06,  98.18, 91.01, 96.98, 97.85, 96.89, 98.35, 90.65, 92.99, 108.97, 91.85,
     104.30, 93.16, 94.18, 89.56, 105.85, 89.29, 92.58, 98.64, 83.70, 88.46, 94.93,
     96.81, 93.13)
##
RA = c(32.08, 39.21, 33.74, 34.19, 35.13, 30.92, 37.71, 44.71, 33.17, 31.74, 37.07,
       38.71, 43.89, 41.06, 41.92, 35.25, 45.11, 39.44, 38.33, 25.93, 36.78, 42.02,
       27.20, 31.37, 27.87, 28.69, 31.04, 32.78, 37.82, 33.36, 31.93, 28.32, 24.82,
       23.88, 36.17)

DentalData = as.data.frame(cbind(MMO = MMO, ML = ML, RA = RA))
```

- **Pair-wise Scatter Plot**

This pairwise scatter plot tells whether there are significant correlations between **numerical predictor variables**.

We can see the following patterns from the above pair-wise scatter plot.

(1). Both ML and RA are linearly correlated with the response variable MMO. This is what we expected.

(2). ML and RA are not linearly correlated. This indicates that there is no collinearity issue.

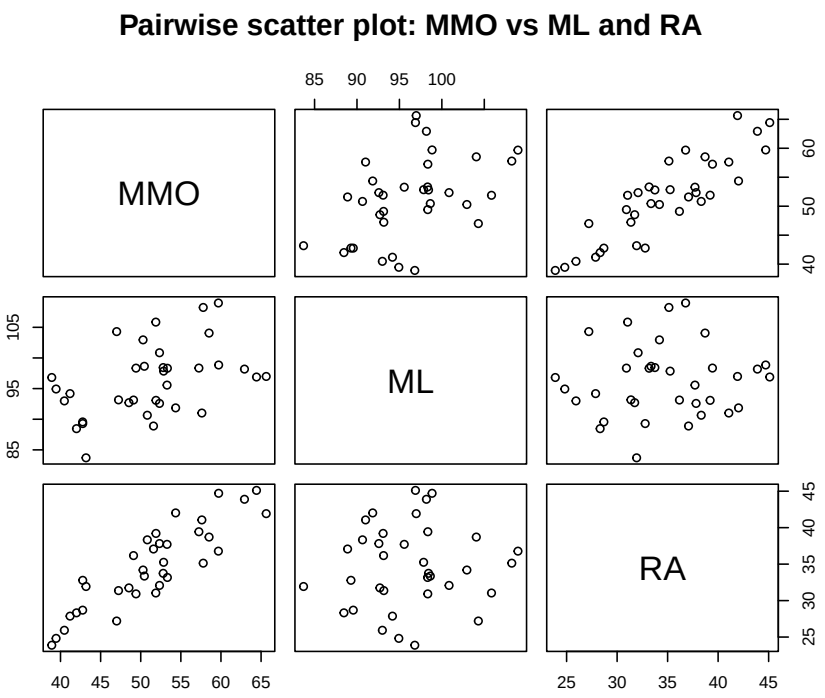


Figure 6.7: Pair-wise scatter plot

(3). We also don't see any special patterns such as outliers and extremely skewed distribution. There is no need to perform discretization and regrouping procedures on the predictor variables.

(4). In this data set, there is no categorical variables or categorical variable with a numerical coding system in this data set. There is no need to create dummy variables.

- **Initial model**

The following initial model includes all predictor variables. The residual plots indicate that

(1). One of the observations seems to be an outlier (observation 15);

(2). There is a minor violation of the assumption of constant variance.

(3). There is also a minor violation of the assumption of normality of the distribution of the residuals.

```
ini.model = lm(MMO ~ ML + RA, data = DentalData)    # fit a linear model with interaction effect
par(mfrow=c(2,2), mar=c(2,3,2,2))
plot(ini.model)
```

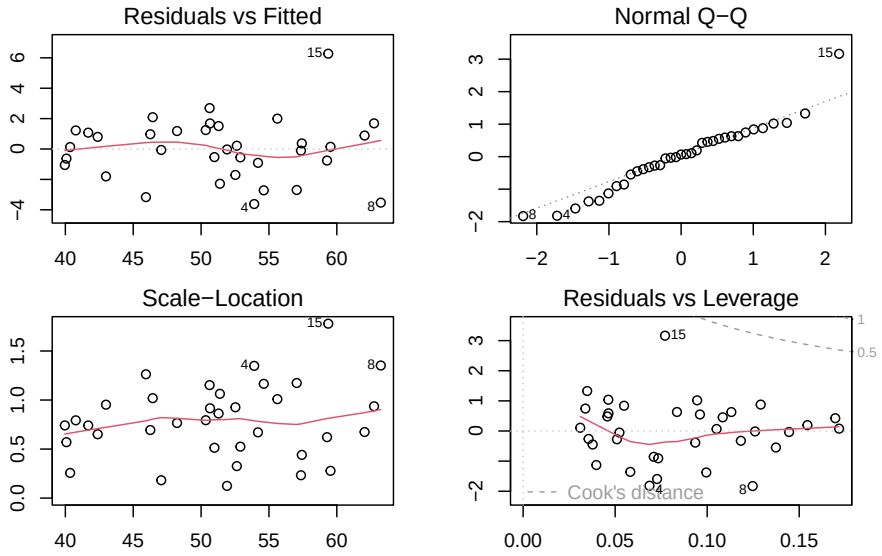


Figure 6.8: Residual plots of the the linear regression model.

Next, we will carry the Box-Cox transformation to identify a potential power transformation of the response variable MMO.

```
library(MASS)
boxcox(MMO ~ ML + RA,
       data = DentalData,
       lambda = seq(-1, 1.5, length = 10),
       xlab=expression(paste(lambda)))
title(main = "Box-Cox Transformation: 95% CI of lambda",
      col.main = "navy", cex.main = 0.9)
```

Since both 0 and 1 are in the 95% confidence interval of λ, technically speaking, there is no need to perform the power transformation. By the optimal λ is closer to 0, we try to perform the log transformation (corresponding to $\lambda = 0$) to see whether there will some improvement of the initial model

```
transform.model = lm(log(MMO) ~ ML * RA, data  = DentalData)
par(mfrow=c(2,2), mar = c(2,2,2,2))
plot(transform.model)
```

The above residual plots indicate an improvement in model fit. We will use the transformed response to build the final model.

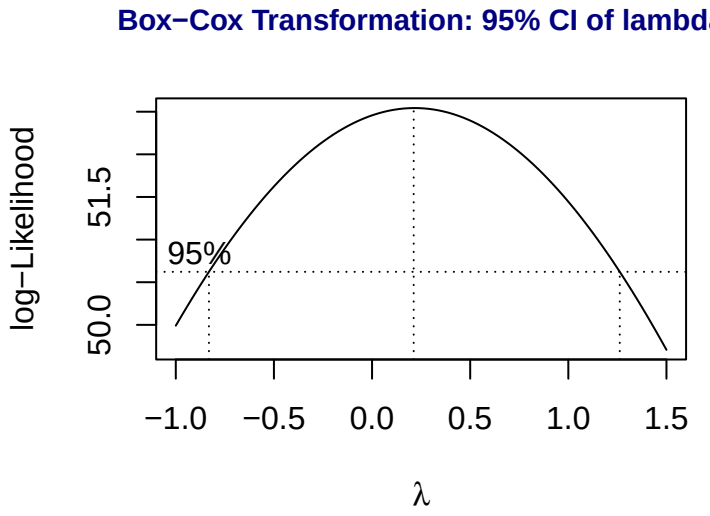


Figure 6.9: Box-cox transformation for power transformation.

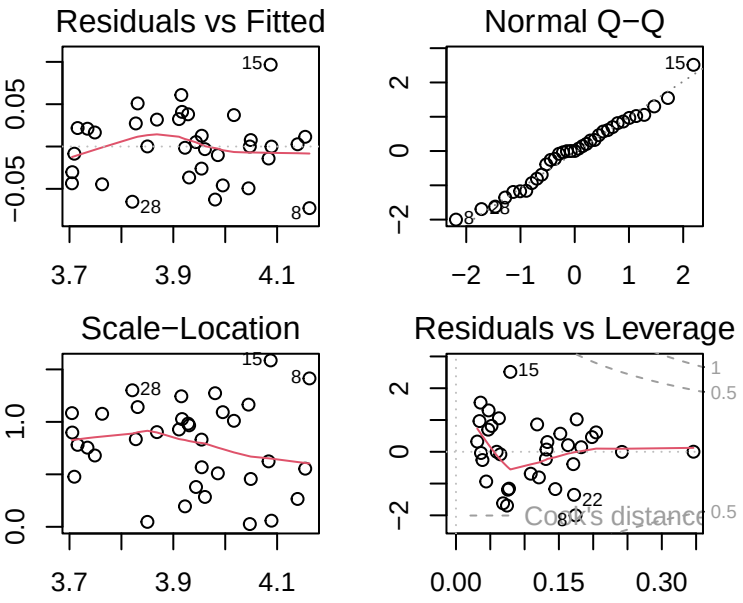


Figure 6.10: Residual plots for the regression with transformed response variable.

Table 6.1: Summarized statistics of the regression coefficients of the model with a log-transformed response

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.9957108	1.0023242	1.9910831	0.0553479
ML	0.0124400	0.0104503	1.1903999	0.2429256
RA	0.0296960	0.0293716	1.0110477	0.3198204
ML:RA	-0.0000884	0.0003059	-0.2890324	0.7744807

Table 6.2: Summary statistics of the regression coefficients of the final model

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	2.2833535	0.1176375	19.410076	0
ML	0.0094391	0.0011705	8.064482	0
RA	0.0212140	0.0012017	17.653748	0

- **Final Model**

The model based on the log-transformed response is summarized in the following.

```
kable(summary(transform.model)$coef, caption = "Summarized statistics of the regression
      coefficients of the model with a log-transformed response")
```

we can see that the interaction effect is insignificant in the model. We drop the highest term in the regression model either manually or automatically. In the next code chunk, we use the automatic variable selection method to find the final model.

```
transform.model = lm(log(MMO)~ML*RA, data = DentalData)
## I will use the automatic variable selection function to search the final model
final.model =  step(transform.model, direction = "backward", trace = 0)
kable(summary(final.model)$coef, caption = "Summary statistics of the regression
      coefficients of the final model")
```

Now we have three candidate models to select from. We extract the coefficient of determination (R^2) of each of the three candidate models.

```
r.ini.model = summary(ini.model)$r.squared
r.transfd.model = summary(transform.model)$r.squared
r.final.model = summary(final.model)$r.squared
##
Rsquare = cbind(ini.model = r.ini.model, transfd.model = r.transfd.model,
                final.model = r.final.model)
kable(Rsquare, caption="Coefficients of correlation of the three candidate models")
```

The second and the third model have almost the same R^2, 92.56% and 92.57%.

Table 6.3: Coefficients of correlation of the three candidate models

ini.model	transfd.model	final.model
0.9204481	0.9257218	0.9255216

Table 6.4: Summary of the final working model

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-31.4247984	6.1474668	-5.111829	1.44e-05
ML	0.4731743	0.0611653	7.735992	0.00e+00
RA	1.0711725	0.0627967	17.057792	0.00e+00

Both models are based on the log-transformed MMO. The interpretations of these two models are not straightforward. The initial model has a slightly lower 92.0%. Since the initial model has a simple structure and is easy to interpret, we choose the initial model as the final model to report. The summarized statistic is given in the following table.

```
summary.ini.model = summary(ini.model)$coef
kable(summary.ini.model, caption = "Summary of the final working model")
```

In summary, both ML and RA are statistically significant (p-value $\approx$ 0) and both are positively correlated to MMO. Further, for a given angle of rotation of the mandible (RA), when mandibular length (ML) increases by 1mm, the maximum mouth opening (MMO) increases by 0.473 mm. However, for holding ML, a 1-degree increase in RA will result in a 1.071 mm increase in MMO.

6.4 Case Study 2

We discussed the ANOVA model in module 8. In fact, the ANOVA model is a special linear regression model. The location is a factor variable. We now build a linear regression using mussel shell length as the response and the location as the predictor variable in the following (code is copied from module 8).

Since predictor variable location is a categorical factor variable, R function **lm()** will automatically define four dummy variables for each category except for the baseline category is, by default, the smallest character values (alphabetical order). In our example, the value **Magadan** is the smallest. Other categories will be compared with the baseline category through the corresponding dummy variable.

To be more specific, the four dummy variables associated with the four categories will be defined by

1. locationNewport = 1 if the location is Newport, 0 otherwise;

Table 6.5: Summary of the ANOVA model

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.0780125	0.0044536	17.5168782	0.0000000
locationNewport	-0.0032125	0.0062983	-0.5100593	0.6133053
locationPetersburg	0.0254304	0.0065193	3.9007522	0.0004300
locationTillamook	0.0021875	0.0059751	0.3661039	0.7165558
locationTvarminne	0.0176875	0.0068029	2.5999834	0.0136962

2. locationPetersburg = 1 if the location is Petersburg, 0 otherwise;
3. locationTillamook = 1 if the location is Tillamook, 0 otherwise;
4. locationTvarminne = 1 if the location is Tvarminne, 0 otherwise.

```
x1 = c(0.0571,0.0813, 0.0831, 0.0976, 0.0817, 0.0859, 0.0735, 0.0659, 0.0923, 0.0836)
x2 = c(0.0873,0.0662, 0.0672, 0.0819, 0.0749, 0.0649, 0.0835, 0.0725)
x3 = c(0.0974,0.1352, 0.0817, 0.1016, 0.0968, 0.1064, 0.1050)
x4 = c(0.1033,0.0915, 0.0781, 0.0685, 0.0677, 0.0697, 0.0764, 0.0689)
x5 = c(0.0703,0.1026, 0.0956, 0.0973, 0.1039, 0.1045)
len  = c(x1, x2, x3, x4, x5)      # pool all sub-samples of lengths
location = c(rep("Tillamook", length(x1)),
             rep("Newport", length(x2)),
             rep("Petersburg", length(x3)),
             rep("Magadan", length(x4)),
             rep("Tvarminne", length(x5)))     # location vector matches the lengths
data.matrix = cbind(len =  len, location = location)   # data a data table
musseldata = as.data.frame(data.matrix)        # data frame
## End of data set creation
##
## Starting building the ANOVA model
anova.model.01 = lm(len ~ location, data = musseldata)  # define a model for generating the ANOVA
##
par(mfrow=c(2,2), mar = c(2,2,2,2))
plot(anova.model.01)
```

The above residual plots indicate no serious violation of the model assumption. The model that generates the above residual plot will be used as the final working model. The inference of the regression coefficients is summarized in the following table.

```
sum.stats = summary(anova.model.01)$coef
kable(sum.stats, caption = "Summary of the ANOVA model")
```

From the above summary tale, we can see that P-values associated with location dummy variables locationNewport, locationTillamook are bigger than 0.05 meaning the means associated with **Newport**, **Tillamook**, and the baseline

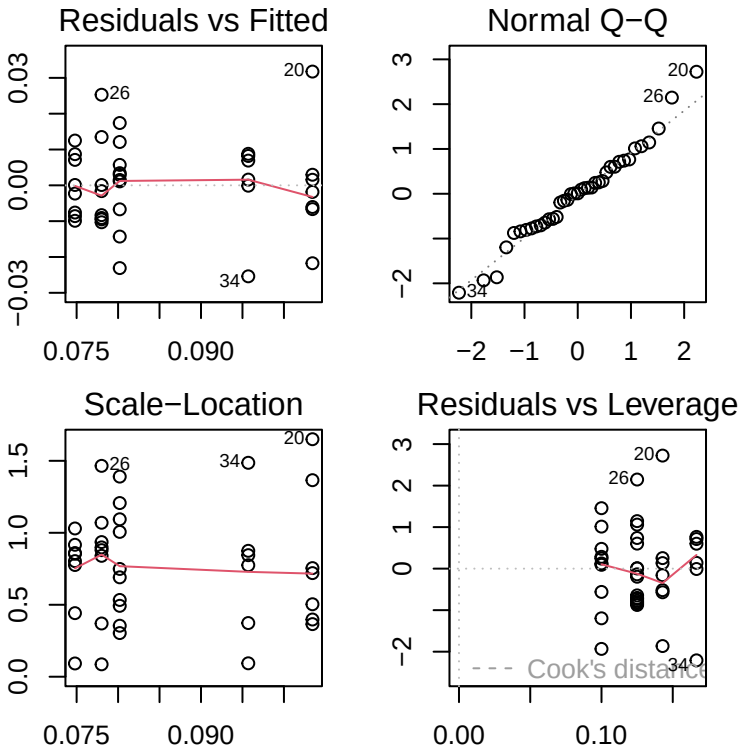


Figure 6.11: Residual plots of the anova model.

Magadan (not appearing in the summary table). The p-values associated with **Petersburg** and **Tvarminn** are less the 0.05 which implies that the mean length of these two locations is significantly different from that of the baseline location **Magadan**. Further, the coefficient associated with dummy variable **locationPetersburg** indicates that the mean length of the mussel shell in **Petersburg** is 0.0543 units longer than that in the baseline location **Magadan**. We can also interpret the coefficients associated with **locationTvarminne**.

Chapter 7

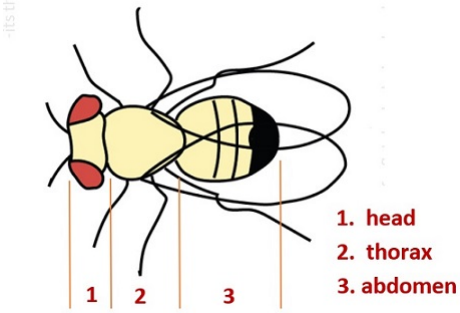
Logistic Regression Models

Linear regression models are used to assess the association between the continuous response variable and other predictor variables. If the response variable is a binary categorical variable, the linear regression model is not appropriate. We need a new model, the logistic regression model, to assess the association between the binary response variable and other predictor variables.

This module focuses on the regression model with a binary response.

7.1 Motivational Example and Practical Question

Example : Longevity in male fruit flies is positively associated with adult size. But reproduction has a high physiological cost that could impact longevity.



The original study looks at the association between longevity and adult size in male fruit flies kept under one of two conditions. One group is kept with sexually active females over the male's life span. The other group is cared for in the same way but kept with females that are not sexually active.

Longevity	ThxLength	IndReprod	Longevity	ThxLength	IndReprod
34	0.78	0	46	0.84	0
42	0.76	0	56	0.76	1
30	0.8	0	76	0.92	1
46	0.88	0	65	0.8	1
40	0.82	0	42	0.76	0
49	0.68	1	19	0.64	0
56	0.8	1	19	0.68	0
70	0.88	1	70	0.88	1
64	0.76	1	26	0.8	0
54	0.88	0	64	0.72	1
85	0.84	1	76	0.92	1
76	0.84	1	33	0.72	0
54	0.88	0	81	0.84	1
61	0.88	0	34	0.72	0
56	0.88	0	54	0.82	0
46	0.76	1	65	0.84	1
44	0.92	0	37	0.68	1
76	0.94	1	39	0.76	1
70	0.84	1	35	0.84	0
64	0.84	1	35	0.64	1
70	0.8	1	30	0.76	0
35	0.84	0	46	0.84	0
65	0.76	1	16	0.64	0
34	0.74	0	46	0.72	1

Figure 7.1: Fruit Flies Data Table

The above table gives the longevity in days for the male fruit flies given an opportunity to reproduce (IndReprod = 0) and for those deprived of the opportunity (IndReprod = 1).

The data was collected from a case-control study design. The original study association analysis used the multiple linear regression model in which Longevity was a response variable and Thorax and IndReprod were used as predictor variables. In this example, we build a logistic regression to assess the association between longevity and reduction. Due to the case-control study design, the resulting logistic regression cannot be used as a predictive model.

Since the response variable is binary (i.e., it can only take on two distinct values, 0 and 1 in this example), the linear regression line is a bad choice since (1) the response variable is not continuous, (2) the fitted regression line can take on any values between negative infinity and positive infinity. The response variable takes on only either 0 or 1 (see the dark red straight line).

A meaningful approach to assess the association between the binary response variable and the predictor variable by looking at how the predictor variables impact the probability of observing the **bigger** value (i.e., the one that has a higher alphabetical order) of the response variable. The above **S** curve describes the relationship between the values of the predictor variable(s) and the probability of observing the **bigger** value of the response variable.

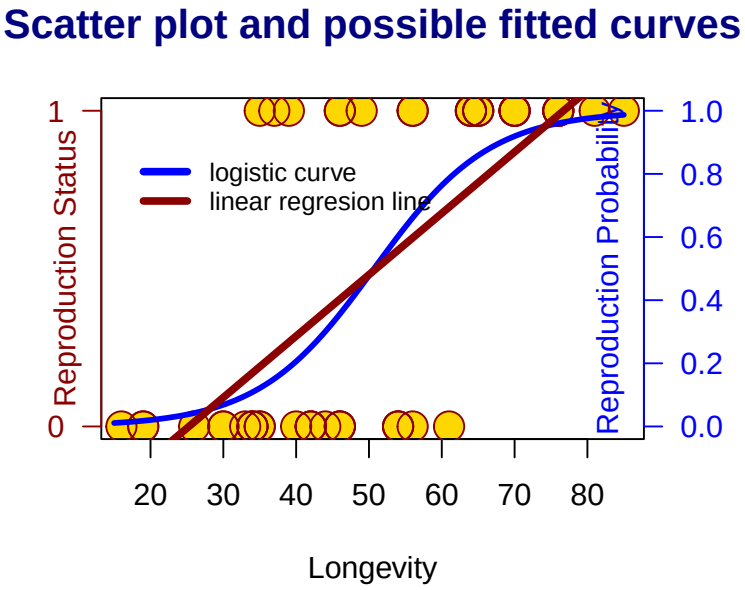


Figure 7.2: The scatter plots of a binary response v.s. a continuous predictor variable

7.2 Logistic Regression Models and Applications

Logistic regression is a member of the generalized linear regression (GLM) family which includes the linear regression models. The model-building process is the same as that in linear regression modeling.

7.2.1 The Structure of the Logistic Regression Model

The actual probability function of observing the **bigger** value of the response variable for giving the predictor variable is given by

$$P(Y = 1) = \frac{\exp(\beta_0 + \beta_1 Longevity)}{1 + \exp(\beta_0 + \beta_1 Longevity)}$$

If β_1 (also called slope parameter) is equal to zero, the longevity and the reproduction status are NOT associated. Otherwise, there is an association between the response and the predictor variables. The sign of the slope parameter reflects the positive or negative association.

7.2.2 Assumptions and Diagnostics

There are assumptions for the binary logistic regression model.

- The response variable must be binary (taking on only two distinct values).
- Predictor variables are assumed to be uncorrelated.
- The functional form of the predictor variables is correctly specified.

The model diagnostics for logistic regression are much more complicated than the linear regression models. We will not discuss this topic in this course.

7.2.3 Coefficient Estimation and Interpretation

The estimation of the logistic regression coefficients is not as intuitive as we saw in the linear regression model (regression lines and regression plane or surface). An advanced mathematical method needs to be used to estimate the regression coefficient. The R function **glm()** can be used to find the estimate of regression coefficients.

The interpretation of the coefficients of the logistic regression model is also not straightforward. In the motivational example, the value of β_1 is the change of log odds of observing reproduction status to be 1. As usual, we will not make an inference from the intercept. In case-control data, the intercept is inestimable.

The output of **glm()** contains information similar to what has been seen in the output of **lm()** in the linear regression model.

7.2.4 Use of glm() and Annotations

We use the motivational example to illustrate the setup of **glm()** and the interpretation of the output.

```
## Fit the logistic regression
mymodel =glm(IndReprod ~ Longevity,    # model formula, the response variable is on the left side.
          family=binomial,             # this must be specified since the response is binary!
          data=fruitflies)             # data frame - the variables in the model MUST identical
                                       # to the ones in the data frame!!!
```

```
glm(formula = IndReprod ~ Longevity, family = binomial, data = fruitflies)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7548  -0.6794  -0.1583   0.5525   2.0535

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.39071    1.72165  -3.712 0.000206 ***
Longevity    0.12605    0.03358   3.753 0.000175 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 67.908  on 48  degrees of freedom
Residual deviance: 39.975  on 47  degrees of freedom
AIC: 43.975

Number of Fisher Scoring iterations: 5
```

The only inforamtion we need for this class.

Figure 7.3: The complete output of glm()

The output has four major pieces of information: model formula, five-number-summary of the deviance residuals, significant test results of the predictor variables, and goodness-of-fit measures. In this course, we only focus on the significance tests.

7.2.5 Applications of Logistic Regression Models

Like other regression models, logistic regression models have two basic applications: association analysis and prediction analysis.

The association analysis focuses on the interpretation of the regression coefficients that have information about whether predictor variables are associated with the response variable through the probability of the **bigger** value of the response variable.

Since the logistic regression builds the relationship between the probability of observing the **bigger** value of the response and the predictor variable, predicting the **value of the response variable** requires a cut-off probability in order to assign a value of 1 or 0 to the response variable. The prediction process of a logistic regression model is depicted in the following figure.

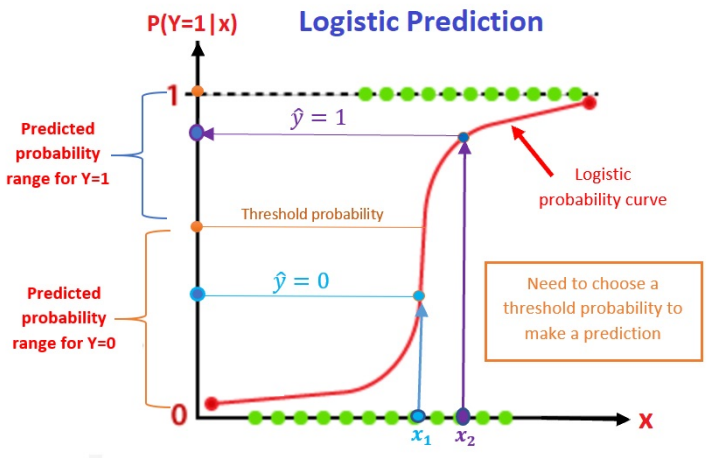


Figure 7.4: Prediction process of the logistic regression models

7.3 Case Studies

We present two examples in this section.

7.3.1 The simple logistic regression model

Suzuki et al. (2006) measured sand grain size on 28 beaches in Japan and observed the presence or absence of the burrowing wolf spider Lycosa ishikariana on each beach. Sand grain size is a measurement variable, and spider presence or absence is a nominal variable. Spider presence or absence is the dependent variable; if there is a relationship between the two variables, it would be sand grain size affecting spiders, not the presence of spiders affecting the sand.

```
grainsize=c(0.245, 0.247, 0.285, 0.299, 0.327, 0.347, 0.356, 0.360, 0.363, 0.364,
            0.398, 0.400, 0.409, 0.421, 0.432, 0.473, 0.509, 0.529, 0.561, 0.569,
            0.594, 0.638, 0.656, 0.816, 0.853, 0.938, 1.036, 1.045)
status=c(0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1,
         1, 1, 1, 1)
spider = as.data.frame(cbind(grainsize = grainsize, status = status))
```

Fitting a Simple Logistic Regression Model

Since there is only one predictor variable in this study, simply choose the simple linear regression model for this data set.

```
spider.model = glm(status ~ grainsize,
                   family = binomial,
                   data = spider)
significant.tests = summary(spider.model)$coef
```

Grain Size (mm)	Status	Numerical Status	Grain Size (mm)	status	Numerical Status
0.245	absent	0	0.432	absent	0
0.247	absent	0	0.473	present	1
0.285	present	1	0.509	present	1
0.299	present	1	0.529	present	1
0.327	present	1	0.561	absent	0
0.347	present	1	0.569	absent	0
0.356	absent	0	0.594	present	1
0.36	present	1	0.638	present	1
0.363	absent	0	0.656	present	1
0.364	present	1	0.816	present	1
0.398	absent	0	0.853	present	1
0.4	present	1	0.938	present	1
0.409	absent	0	1.036	present	1
0.421	present	1	1.045	present	1

Figure 7.5: Spider Data Table

Table 7.1: Summary of the significant tests of the logistic regression model

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-1.647625	1.354191	-1.216686	0.2237237
grainsize	5.121553	3.006174	1.703678	0.0884413

```
kable(significant.tests, caption = "Summary of the significant tests of
      the logistic regression model")
```

Association Analysis

The above significant tests indicate that the grain size does not achieve significance (p-value = 0.08844) at level 0.05. Note that the p-value is calculated based on the sample, it is also a random variable. Moreover, the sample size in this study is relatively small. We will claim the association between the two variables. As the grain size increases by one unit, the log odds of observing the wolf spider burrowing increase by 5.121553. In other words, the grain size and the presence of spiders are positively associated.

Prediction Analysis

As an example, we choose two new grain sizes 0.33 and 0.57, and want to predict the presence of the wide spiders on the beaches with the given grain sizes. We used the R function **predict()** in linear regression, we use the same function to make predictions in the logistic regression model.

```
spider.model = glm(status ~ grainsize,
                   family = binomial,
                   data = spider)
##
```

Table 7.2: Predicted Value of response variable with the given cut-off probability

new.grain.size	pred.response
0.275	0
0.570	1

```
mynewdata = data.frame(grainsize=c(0.275, 0.57))
pred.prob = predict(spider.model, newdata = mynewdata,
        type = "response")
## threshold probability
cut.off.prob = 0.5
pred.response = ifelse(pred.prob > cut.off.prob, 1, 0)  # This predicts the response
pred.table = cbind(new.grain.size = c(0.275, 0.57), pred.response = pred.response)
kable(pred.table, caption = "Predicted Value of response variable
      with the given cut-off probability")
```

7.3.2 Multiple Logistic Regression Model

In this case study, we used a published study on bird introduction in New Zealand. The objective is to predict the success of avian introduction to New Zealand. Detailed information about the study can be found in the following article. https://pengdsci.github.io/STA551/w04/Correlates-of-introduction-success-in-exotic.pdf. The data is included in the article. A text format data file was created and can be downloaded or read directly from the following URL: https://pengdsci.github.io/STA551/w04//img/birds-data.txt.

The response variable: Status - status of success Predictor variables:

- length - female body length (mm)
- mass = female body mass (g)
- range = geographic range (% area of Australia)
- migr = migration score: 1= sedentary, 2 = sedentary and migratory, 3 = migratory
- insect = the number of months in a year with insects in the diet

- diet = diet score: 1 = herbivorous; 2 = omnivorous, 3 = carnivorous.
- clutch = clutch size
- broods = number of broods per season
- wood = use as woodland scored as frequent(1) or infrequent(2)
- upland = use of the upland as frequent(1) or infrequent(2)
- water = use of water scored as frequent(1) or infrequent(2)
- release = minimum number of release events
- indiv = minimum of the number of individuals introduced.

We next read the data from the given URL directly to R. Since there are some records with missing values. We drop those records with at least one missing value.

There are several categorical variables coded in numerical form. Among them, **migr** and **diet** have three categories and the rest of the categorical variables have two categories. In practice, a **categorical variable with more than two categories must be specified as factor variables** so R can define dummy variables to capture the difference across the difference.

We conducted an exploratory analysis on **nigr** and **diet** and found a flat discrepancy across the effect. We simply treat them as discrete numerical variables using the numerical coding as the values of the variables.

```
NZbirds=read.table("https://pengdsci.github.io/STA551/w04/img/birds-data.txt", header=TRUE)
birds = na.omit(NZbirds)
```

- **Build an Initial Model**

We first build a logistic regression model that contains all predictor variables in the data set. This model is usually called the full model. Note that the response variable is the success status (1 = success, 0 = failure). Species is a kind of ID, it should not be included in the model.

```
initial.model = glm(status ~ length + mass + range + migr + insect + diet + clutch + broods +
                      wood + upland + water + release + indiv, family = binomial, data = birds)
coefficient.table = summary(initial.model)$coef
kable(coefficient.table, caption = "Significance tests of logistic regression model")
```

The p-values in the above significance test table are all bigger than 0.5. We next search for the best model by dropping some of the insignificant predictor variables. Since there are so many different ways to drop variables, next we use an automatic variable procedure to search the final model.

- **Automatic Variable Selection**

R has an automatic variable selection function **step()** for searching the final model. We will start from the initial model and drop insignificant variables using AIC as an inclusion/exclusion criterion.

In practice, sometimes, there may be some practically important predictor variables. Practitioners want to include these practically important variables in the

Table 7.3: Significance tests of logistic regression model

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-6.3380099	5.7167615	-1.1086714	0.2675720
length	-0.0028150	0.0053169	-0.5294328	0.5965052
mass	0.0026679	0.0016740	1.5937709	0.1109874
range	-0.1316071	0.3502344	-0.3757688	0.7070888
migr	-2.0435447	1.1158239	-1.8314222	0.0670376
insect	0.1479915	0.2123645	0.6968752	0.4858809
diet	2.0285053	1.8832013	1.0771580	0.2814097
clutch	0.0793804	0.2683046	0.2958594	0.7673375
broods	0.0217705	0.9283268	0.0234513	0.9812903
wood	2.4902098	1.6416010	1.5169397	0.1292819
upland	-4.7134743	2.8648273	-1.6452909	0.0999098
water	0.2349442	2.6701934	0.0879877	0.9298864
release	-0.0129156	0.1932112	-0.0668472	0.9467033
indiv	0.0159265	0.0083240	1.9133152	0.0557077

model regardless of their statistical significance. Therefore we can fit the smallest model that includes only those practically important variables. The final model should be **between** the smallest model, which we will call a **reduced model**, and the initial model, which we will call a **full model**. For illustration, we assume **insect** and **range** are practically important, we want to include these two variables in the final model regardless of their statistical significance.

In summary, we define two models: the full model and the reduced model. The final best model will be the model between the full and reduced models. The summary table of significant tests is given below.

```
full.model = initial.model  # the *biggest model* that includes all predictor variable
reduced.model = glm(status ~ range + insect , family = binomial, data = birds)
final.model =  step(full.model,
                    scope=list(lower=formula(reduced.model),upper=formula(full.model))
                    data = birds,
                    direction = "backward",
                    trace = 0)   # trace = 0: suppress the detailed selection process
final.model.coef = summary(final.model)$coef
kable(final.model.coef , caption = "Summary table of significant tests")
```

- **Interpretation - Association Analysis**

The summary table contains the two practically important variables **range** and **insect**. **range** does not achieve statistical significance (p-value ≈ 1) and **insect** is slightly higher than the significance level of 0.005. Both variables are seemingly positively associated with the response variable.

Table 7.4: Summary table of significant tests

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-3.4376339	2.1379440	-1.6079158	0.1078536
mass	0.0019393	0.0007326	2.6470676	0.0081193
range	0.0000141	0.3098265	0.0000456	0.9999636
migr	-2.0239952	0.9603017	-2.1076659	0.0350599
insect	0.2704685	0.1425911	1.8968119	0.0578528
wood	1.9489529	1.3174130	1.4793789	0.1390391
upland	-4.7306337	2.0981447	-2.2546746	0.0241538
indiv	0.0138120	0.0040579	3.4037151	0.0006648

The following interpretation of the individual predictor variable assumes other life-history variables and the introduction effort variable.

migr and **upland** are negatively associated with the response variable. The odds of success in introducing migratory birds are lower than the sedentary birds. Similarly, birds using upland infrequently have lower odds to be successfully introduced than those using upland frequently.

insect is significant (p-value =0.058). The **odds of success** increase as the number of months of having insects in the diet increases.

mass and **indiv** are positively associated with the response variable. The odds of success increase and the body mass increases Similarly, the odds of success increase as the number of minimum birds of the species increases.

wood does not achieve statistical significance but seemed to be positively associated with the response variable.

- **Predictive Analysis**

As an illustration, we use the final model to predict the status of successful introduction based on the new values of the predictor variables associated with two species. See the numerical feature given in the code chunk.

```
mynewdata = data.frame(mass=c(560, 921),
                       range = c(0.75, 1.2),
                       migr = c(2,1),
                       insect = c(6, 12),
                       wood = c(1,1),
                       upland = c(0,1),
                       indiv = c(123, 571))
pred.success.prob = predict(final.model, newdata = mynewdata, type="response")
##
## threshold probability
cut.off.prob = 0.5
pred.response = ifelse(pred.success.prob > cut.off.prob, 1, 0)  # This predicts the response
```

Table 7.5: Predicted Value of response variable with the given cut-off probability

mass	range	migr	insect	wood	upland	indiv	Pred.Response
560	0.75	2	6	1	0	123	0
921	1.20	1	12	1	1	571	1

```
## Add the new predicted response to Mynewdata
mynewdata$Pred.Response = pred.response
##
kable(mynewdata, caption = "Predicted Value of response variable
      with the given cut-off probability")
```

The predicted status of the successful introduction of the two species is attached to the two new records.

Chapter 8

Basics of Bootstrap Method

The bootstrap method is a data-based simulation method for statistical inference. The method assumes that

- The sample is a random sample representing the population;
- The sample size is large enough such that the empirical distribution can be close to the true distribution.

8.1 Basic Idea of Bootstrap Method.

The objective is to estimate a population parameter such as mean, variance, correlation coefficient, regression coefficients, etc. from a random sample without assuming any probability distribution of the underlying distribution of the population.

For convenience, we assume that the population of interest has a cumulative distribution function $F(x:\theta)$, where θ is a vector of the population. For example, You can think about the following distributions

- **Normal distribution**: $N(\mu, \sigma^2)$, the distribution function is given by

$$f(x:\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

  where $\theta = (\mu, \sigma)$. Since the normal distribution is so fundamental in statistics, we use the special notation for the cumulative distribution $\phi_{\mu,\sigma^2}(x)$ or simply $\phi(x)$. The corresponding probability function

- **Binomial distribution**: $Binom(n, p)$, the probability distribution is given by

$$P(x) = \frac{n!}{x!(n-x)!}p^x(1-p)^{n-x}, x = 0, 1, 2, \cdots, n-1, n.$$

where $\theta = p$. *Caution*: n is NOT a parameter!

We have already learned how to make inferences about population means and variances under various assumptions in elementary statistics. In this note, we introduce a **new approach** to making inferences only based on a given random sample taken from the underlying population.

As an example, we focus on the population mean. For other parameters, we can follow the same idea to make bootstrap inferences.

8.1.1 Random Sample from Population

We have introduced various study designs and sampling plans to obtain random samples from a given population with the distribution function $F(x : \theta)$. Let μ be the population mean.

- **Random Sample**. Let

$$\{x_1, x_2, \cdots, x_n\} \to F(x : \theta)$$

be a random sample from population $F(x : \theta)$.

- **Sample Mean**. The point estimate is given by

$$\widehat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n}$$

- **Sampling Distribution of $\widehat{\mu}$**. In order to construct the confidence interval of μ or make hypothesis testing about μ, we need to know the sampling distribution of $\widehat{\mu}$. From elementary statistics, we have the following results.
  - $\widehat{\mu}$ is normally distributed if (1). n is large; or (2). the population is normal and population variance is known.
  - the standardized $\widehat{\mu}$ follows a t-distribution if the population is normal and population variance is unknown.
  - $\widehat{\mu}$ is **unknown** of the population is not normal and the sample size is not large enough.
- In the last case of the previous bullet point, we don't have the theory to derive the sampling distribution based on a **single** sample. However, if the sampling is not too expensive and time-consuming, we take following the sample study design and sampling plan to repeatedly take a large number, 1000, samples of the same size from the population. We calculate the mean of each of the 1000 samples and obtain 1000 sample means $\{\widehat{\mu}_1, \widehat{\mu}_2, \cdots, \widehat{\mu}_{1000}\}$. Then the empirical distribution of $\widehat{\mu}$.

The following figure depicts how to approximate the sampling distribution of the point estimator of the population parameter.

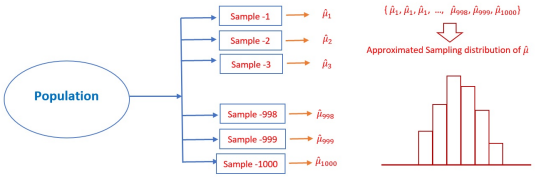


Figure 8.1: Figure 1. Steps for estimating the sampling distribution of a point estimator of the population parameter

Example 1: [**Simulated data**] Assume that the particular numeric characteristics of the WCU student population are the heights of all students.

- We don't know the distribution of the heights.
- We also don't know *whether a specific sample size is large enough to use the central limit theorem*. This means we don't know whether it is appropriate to use the central limit theorem to characterize the sampling distribution of the mean height.

Due to the above constraints, we cannot find the sampling distribution of the sample mean using only the knowledge of elementary statistics. However, if sampling is not expensive, we take repeated samples with the same sample size. The resulting sample means can be used to approximate the sampling distribution of the sample mean.

Next, we use R and the simulated data set https://pengdsci.github.io/STA 551/w05/w05-wcuheights.txt to implement the above idea. I will use simple code with comments to explain the task of each line of code so you can easily understand the coding logic.

```
# read the delimited data from URL
wcu.height = read.table("https://pengdsci.github.io/STA551/w05/w05-wcuheights.txt", header = TRUE
sample.mean.vec = NULL      # define an empty vector to hold sample means of repeated samples.
for(i in 1:1000){           # starting for-loop to take repeated random samples with n = 81
  ith.sample = sample( wcu.height$Height,       # population of all WCU students heights
                       81,                      # sample size = 81 values in the sample
                       replace = FALSE          # sample without replacement
                     )                          # this is the i-th random sample
   sample.mean.vec[i] = mean(ith.sample)        # calculate the mean of i-th sample and save it t
                                                # the empty vector: sample.mean.vec
  }
```

Next, we make a histogram of the sample means saved in **sample.mean.vec**.

```
hist(sample.mean.vec,                                                    # data used for hi
     breaks = 14,                                                        # specify the numb
     xlab = "sample means of repeated samples",                          # change the label
     main="Approximated Sampling Distribution of the Sample Mean")       # add a title to
```

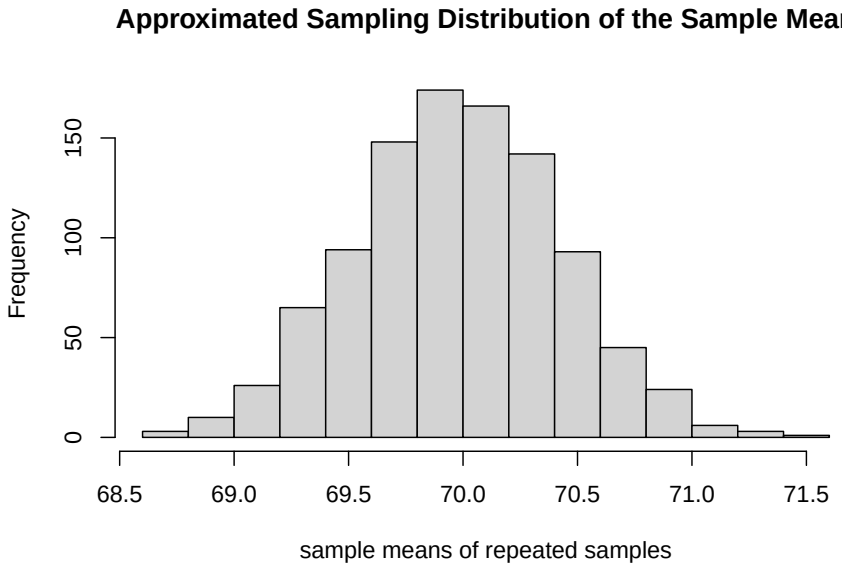


Figure 8.2: Figure 2. Approximated sampling distribution of sample mean used the repeated samples.

8.1.2 Bootstrap Sampling and Bootstrap Sampling Distribution

Recall the situation in **Example 1** in which we were not able to use the normality assumption of the population and the central limit theorem (CLT) but were allowed to take repeated samples from the population. In practice, taking samples from the population can be very expensive. Is there any way to estimate the sampling distribution of the sample mean? The answer is YES under the assumption the sample yields a valid estimation of the original population distribution.

- **Bootstrap Sampling** With the assumption that the sample yields a good approximation of the population distribution, we can take bootstrap samples from the **actual** sample. Let

$$\{x_1, x_2, \cdots, x_n\} \to F(x : \theta)$$

be the actual random sample taken from the population. A **bootstrap sample** is obtained by taking a sample **with replacement** from the original data set (not the population!) with the same size as the original sample. Because **with replacement** was used, some values in the bootstrap sample appear once, some twice, and so on, and some do not appear at all.

- **Notation of Bootstrap Sample**. We use $\{x_1^{(i*)}, x_2^{(i*)}, \cdots, x_n^{(i*)}\}$ to denote the i^{th} bootstrap sample. Then the corresponding mean is called bootstrap sample mean and denoted by $\hat{\mu}_i^*$, for $i = 1, 2, ..., n$.
- **Bootstrap sampling distribution** of the sample mean can be estimated by taking a large number, say B, of bootstrap samples. The resulting B bootstrap sample means are used to estimate the sampling distribution. Note that, in practice, B is bigger than 1000.

The above Bootstrap sampling process is illustrated in the following figure.

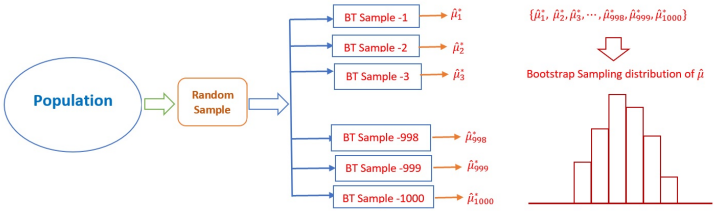


Figure 8.3: Steps for the Bootstrap sampling distribution of a point estimator of the population parameter

- **Example 2:** [**continue to use WCU Heights**]. We use the Bootstrap method to estimate the sampling distribution of the sample mean.

```
### Read the delimited data from the URL
wcu.height = read.table("https://pengdsci.github.io/STA551/w05/w05-wcuheights.txt", header = TRUE
# taking the original random sample from the population
original.sample = sample( wcu.height$Height,     # population of all WCU students heights
                          81,                    # sample size = 81 values in the sample
                          replace = FALSE        # sample without replacement
                  )
### Bootstrap sampling begins
bt.sample.mean.vec = NULL       # define an empty vector to hold sample means of repeated samples.
for(i in 1:1000){               # starting for-loop to take bootstrap samples with n = 81
  ith.bt.sample = sample( original.sample,     # Original sample with 81 WCU students' heights
                          81,                  # sample size = 81 MUST be equal to the sample size
                          replace = TRUE       # MUST use WITH REPLACEMENT!!
                  )                            # This is the i-th Bootstrap sample
  bt.sample.mean.vec[i] = mean(ith.bt.sample) # calculate the mean of i-th bootstrap sample and
```

```
                                                    # save it in the empty vector: sample.bt
  }
```

The following histogram shows the bootstrap sampling distribution of the sample with size $n = 81$.

```
hist(bt.sample.mean.vec,                                    # data used for histo
     breaks = 14,                                           # specify the number
     xlab = "Bootstrap sample means",                       # change the label of
     main="Bootstrap Sampling Distribution of the Sample Mean")   # add a title to the
```

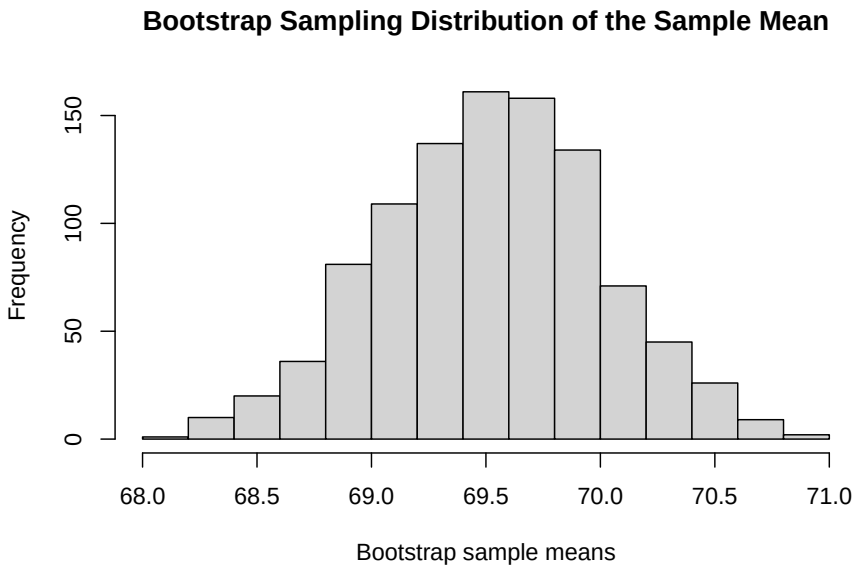


Figure 8.4: Figure 4. Bootstrap sampling distribution of sample means

8.1.3 Relationship between Two Estimated Sampling Distributions

We can see that the two sampling distributions are slightly different. If we are allowed to take repeated samples from the population, we should always use the repeated sample approach since it yields a better estimate of the true sampling distribution.

The bootstrap estimate of the sampling distribution is used when no theoretical confidence intervals are unavailable and the repeated sample is not possible. This does not mean that the bootstrap methods do not have limitations. In fact, the implicit assumption of the bootstrap method is that **the original sample**

has enough information to estimate the true population distribution.

8.2 Bootstrap Confidence Intervals

First of all, all bootstrap confidence intervals are constructed based on the bootstrap sampling distribution of the underlying point estimator of the parameter of interest.

There are at least five different bootstrap confidence intervals. You can find these definitions from Chapter 4 of Roff's eBook https://ebookcentral.proquest.com/lib/wcupa/reader.action?docID=261114&ppg=7 (need WCU login credential to access the book). We only focus on the percentile method in which we simply define the confidence limit(s) by using the corresponding percentile(s) of the bootstrap estimates of the parameter of interest. R has a built-in function, **quantile()**, to find percentiles.

- **Example 3**: We construct a 95% two-sided bootstrap percentile confidence interval of the mean height of WCU students. This is equivalent to finding the 2.5% and 97.5% percentiles. We use the following one-line code.

```
CI = quantile(bt.sample.mean.vec, c(0.025, 0.975))
CI
```

```
##     2.5%    97.5%
## 68.55556 70.45710
```

```
#kable(CI, caption = "95% bootstrap percentile confidence interval of the mean height")
```

Various bootstrap methods were implemented in the R library **{boot}**. UCLA Statistical Consulting https://stats.idre.ucla.edu/r/faq/how-can-i-generate-bootstrap-statistics-in-r/ has a nice tutorial on bootstrap confidence intervals. You can use the built-in function **boot.ci()** to find all 5 bootstrap confidence intervals after you create the boot object. I will leave it to you if you want to explore more about the library.

8.3 Bootstrap Confidence Interval of Correlation Coefficient

As a case study, we will illustrate one bootstrap method to sample a random sample with multiple variables and use the bootstrap samples to calculate the corresponding bootstrap correlation coefficient. The bootstrap percentile confidence interval of the correlation coefficient.

8.3.1 Bootstrapping Data Set

There are different ways to take bootstrap samples. **The key point is that we cannot sample individual variables in the data frame separately to avoid mismatching!** The method we introduce here is also called bootstrap sampling cases. Here are the basic steps:

- We define the vector of row ID. That is, $ID = \{1, 2, 3, ..., n\}$.
- Take a bootstrap sample from ID (i.e., sampling with replacement) with same size = n, denoted by ID^*. As commented earlier, there will be replicates in ID^* and some values in ID are not in ID^*.
- Use ID^* to select the corresponding rows to form a bootstrap sample and then perform bootstrap analysis.

Here is an example of taking the bootstrap sample from the original sample with multiple variables. The data set we use here is well-known and is available at https://pengdsci.github.io/STA551/w05/w05-iris.txt

```
# read data into R from URL:
iris = read.table("https://pengdsci.github.io/STA551/w05/w05-iris.txt", header = TRUE)
n = dim(iris)[1]        # returns the dimension of the data frame, 1st component is th
bt.ID = sample(1:n, replace = TRUE)   # bootstrap IDs, MUST use replacement method!
sort(bt.ID)                  # check the content of bt.ID. I sort the bt.ID to see the
```

```
##   [1]   1   2   2   2   5   6   6   7  10  11  13  13  15  16  19  19  20  21  22  2
##  [29]  31  31  34  36  37  38  38  39  40  40  41  43  44  44  45  45  46  47  47  4
##  [57]  57  59  59  60  61  62  64  65  65  67  67  68  71  71  72  75  77  79  80  8
##  [85]  89  90  91  91  92  92  92  92  93  95  96  96  96  97  97  98  98 100 101 10
## [113] 113 118 118 118 119 120 121 121 124 125 126 128 129 129 130 130 135 135 135 13
## [141] 141 141 142 143 144 145 145 146 148 149
```

Next, we use the above bt.ID to take the bootstrap sample from the original data set **iris**.

```
bt.iris = iris[bt.ID,]   # taking bootstrap cases (or rows, records) using the bt.ID
bt.iris                  # display the bootstrap sample
```

8.3.2 Confidence Interval of Coefficient Correlation

In this section, we construct a 95% bootstrap percentile confidence interval for the coefficient correlation between the SepalLength and SepalWidth given in **iris**. Note that R built-in function **cor(x,y)** can be used to calculate the bootstrap correlation coefficient directly. The R code for constructing the bootstrap confidence interval for the coefficient correlation is given below.

```
iris = read.table("https://pengdsci.github.io/STA551/w05/w05-iris.txt", header = TRUE)
n = dim(iris)[1]        # returns the dimension of the data frame, 1st component is th
##
```

```
bt.cor.vec = NULL    # empty vector bootstrap correlation coefficients
for (i in 1:5000){   # this time I take 5000 bootstrap samples for this example.
  bt.ID.i = sample(1:n, replace = TRUE)  # bootstrap IDs, MUST use replacement method!
  bt.iris.i = iris[bt.ID.i, ]            # i-th bootstrap ID
  bt.cor.vec[i] = cor(bt.iris.i$SepalLength, bt.iris.i$SepalWidth)   # i-th bootstrap correlation
}
bt.CI = quantile(bt.cor.vec, c(0.025, 0.975) )
bt.CI
```

```
##       2.5%      97.5%
## -0.2558453  0.0376732
```

Interpretation: We are 95% confident that there is no statistically significant correlation between sepal length and sepal width based on the given sample. This may be because the data set contains three different types of iris.

Next, we make two plots to visualize the relationship between the two variables.

```
par(mfrow=c(1,2))   # Layout a plot sheet: 1 row and 2 columns
## histogram
hist(bt.cor.vec, breaks = 14,
       main="Bootstrap Sampling \n  Distribution of Correlation",
      xlab = "Bootstrap Correlation Coefficient")
## scatter plot
plot(iris$SepalLength, iris$SepalWidth,
     main = "Sepal Length vs Width",
     xlab = "Sepal Length",
     ylab = "Sepal Width")
```

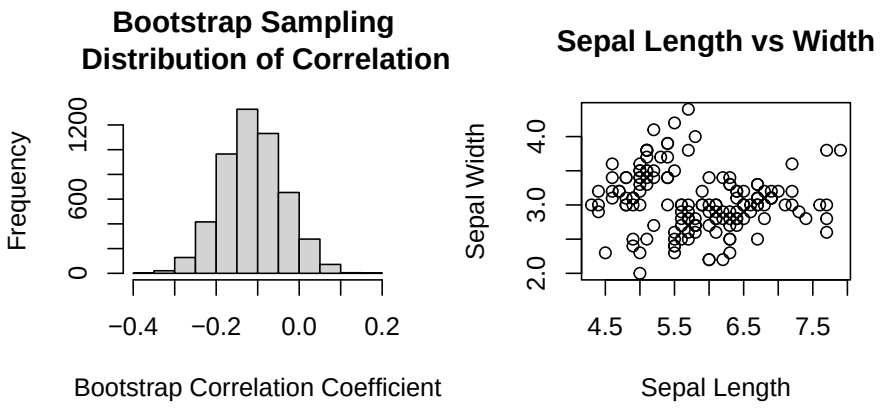


Figure 8.5: Left panel: histogram of the bootstrap coefficient of correlation. Right panel: the scatter plot of the sepal length and width.

Chapter 9

Method of Cross-Validation

In classical statistical modeling, we select candidate models based on exploratory data analysis and visual analysis. The candidate models are then fit to the analytic data set. Since all models have some sort of assumptions (think about linear regression and logistic and Poisson regression models), and then carry out model diagnostic analyses to identify potential violations of the model assumption. Of cause, we assume that the data represent the population. This modeling process is depicted in the following diagram.

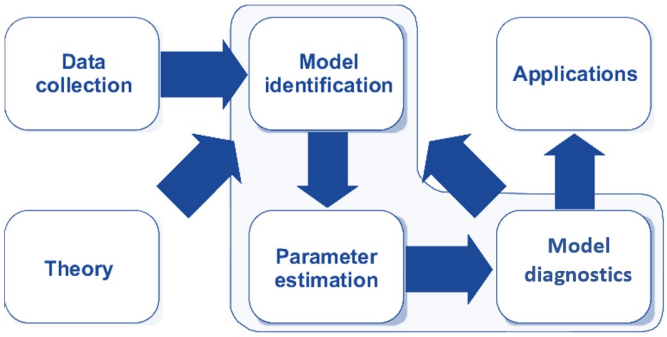


Figure 9.1: Statistical modeling process.

9.1 Regression Models

Recall the process of the following two representative statistical models

9.1.1 Linear Regression Model

First, we look at the process of how to build a linear regression model for a data set.

- **Explicit and Implicit Assumptions**. The following are some of the assumptions about a normal-based linear regression model:

  - Predictor (feature) variables are (linearly or curve-linearly) correlated to the response variable (also called label in machine learning terms);
  - Predictor variables themselves are NOT highly linearly correlated (i.e., no serious collinearity);
  - The response variable is normally distributed;
  - The variance of the response variable is constant;
  - There are no outlier (influential) observations;
  - information on all relevant predictor variables is available in the data (i.e., no important predictor is missing);

- **Model Fitting (Parameter Estimating)**. The least-square estimator (LSE), which is equivalent to the maximum likelihood estimator (MLE) when the response variable is assumed to be a normal distribution, can be used to estimate the regression coefficients.

- **Model Selection and Diagnostics**. Since several implicit and explicit assumptions have been assumed underlying the linear regression, different sets of diagnostic measures were developed to detect different potential violations of the model.

  - *global goodness-of-fit*: R2, AIC and SBC (requires normality assumption of the response variable), MSE, etc.
  - *local goodness-of-fit/model selection*: R, F-test (need normality and equal variance of the response variable), t-test, likelihood ratio test, Mallow's Cp, etc.
  - *normality*: QQ-plot and probability plot for a visual check, goodness-of-fit tests such as Anderson-Darling, Cramer-von Miss, Kolmogorov-Smirnov, Shapiro-Wilks, and Pearson's chi-square tests, etc.
  - *detecting outliers/influential observations*: leverage point-hat matrix, DFITT - defined based on leave-one-out resampling method, cook's distance, (scaled) residual plots, etc.
  - *verifying constant variance*: F-test (requires normality), Brown-Forsythe test (nonparametric), Breusch-Pagan Test (also nonparametric), Bartlett's Test (requires normality), etc.
  - *detecting collinearity*: Variance inflation factor (VIF) for data-based and structural collinearity.
  - *detecting mission of determinant variable*:

9.1.2 Logistic Regression Model

For the binary logistic model, we also follow the same steps to identify the final model. For example, the well-known binary logistic regression modeling follows similar steps:

- **Assumptions**: Unlike the linear regression model which has a strong assumption of normality, the logistic regression model assumes the following
  - *binomial distribution*: The dependent variable is binary.
  - *independence*: The logistic regression requires the observations to be independent of each other.
  - *collinearity*: The logistic regression requires there to be little or no multicollinearity among the independent variables.
  - *linearity*: The logistic regression assumes linearity of independent variables and the log odds of the event of interest.
  - *large sample size*: The logistic regression typically requires a large sample size. A general guideline is that you need a minimum of 10 observations with the least frequent outcome for each independent variable in your model.
  - *mis-specification*: No important variables are omitted. No extraneous variables are included.
  - *measurement error*: The independent variables are measured without error.
- **Model Fitting**: The coefficients of the logistic regression model are estimated using a maximum likelihood estimator. Note LSE cannot be estimated for the logistic regression model.
- **Model Selection and Diagnostics**: Unlike the normal linear regression model, there are a few diagnostic methods one can use in logistic regression models.
  - *misspecification*: link test (large sample test);
  - *goodness-of-fit*: log-likelihood chi-square and pseudo-R-square; Hosmer-Lemeshow's lack of fit test AIC and SBC.
  - *multi-collinearity*: VIF
  - *influential points*: Cook's distance, DBETA, deviance residuals

9.1.3 Inference about Association and Prediction

In classical statistics, most problems are related to the significance of the regression coefficients. The p-values associated with corresponding regression coefficients are calculated based on certain assumptions. Prediction intervals are also constructed based on certain assumptions.

We can also use resampling methods to relax the assumptions about the distribution of the response variable to make inferences about the regression coefficients and construct prediction intervals.

In practice, the prediction problems usually involve both classical statistical models and machine learning algorithms. The model selection process and performance evaluation in classical statistical modeling is dependent on model assumptions, while in machine learning algorithms, model selection and evaluation use data-driven methods that are also valid for modern statistical modeling.

In the next few sections, we will introduce the data-driven methods for machine learning algorithms and statistical models.

9.2 Data Splitting Methods

When training and testing machine learning and statistical models without assuming strong assumptions (particularly the distributional information), we break down the data into three separate and distinct data sets: training data, validation data, and testing data. The basic idea is depicted in the following chart.

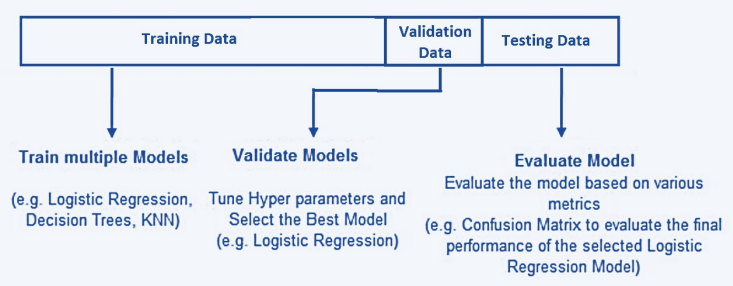


Figure 9.2: The-way data splitting method.

9.2.1 Training Data

The training data set is the sample of data used to fit the model. In other words, the training data teaches the model how it's supposed to learn and think. To do this, training data is often presented in pairs: predictor variables (feature variables) and a response variable (also called a label).

Training data is the first set of data the model/algorithm is exposed to. During each stage of training, the model will be fit to the training data and estimate the parameters (also called weight in some machine learning algorithms such as neural networks).

Because the training data set is used to estimate the parameters (i.e., teaching the algorithm), it requires a certain amount of information to make the algorithms and models reliable. It makes up between 60% and 80% of the total data.

9.2.2 Validation Data

The validation data set is a sample of data held back from training the model. This data set provides an unbiased evaluation of a model fit on the training data set while tuning model hyperparameters. In more basic terms, validation data is an unused portion of your training data and helps determine if the initial model is accurate.

A model **hyperparameter** is a configuration that is external to the model and whose value cannot be estimated from data.

- They are often used in processes to help estimate model parameters.
- They are often specified by the practitioner.
- They can often be set using heuristics.
- They are often tuned for a given predictive modeling problem.

We cannot know the best value for a model hyperparameter on a given problem. We may use rules of thumb, copy values used on other problems, or search for the best value by trial and error.

When a machine learning algorithm is tuned for a specific problem, such as the cut-off probability determination in the logistic prediction, then we are tuning the hyperparameters of the model or order to discover the parameters of the model that result in the most skillful predictions. The validation data helps tune a machine-learning model while checking for overfitting.

Overfitting happens when the model/algorithm is too closely fitted to the training data — producing results tied to the specifics of that first data set. After validation, the team will often return to the training data and run it again, making adjustments to values and parameters to improve the model.

9.2.3 Test data

The test data set is a sample of data used to provide an unbiased evaluation of a final model fit on the training data set or to test the model. Put more simply, test data is a set of **unlabeled inputs** (i.e., the response value is removed from the data) that test whether the model is producing the correct outputs in the real world.

The key difference between a validation data set and a test data set is that the validation data set is used during model configuration, while the test data set is reserved to evaluate the final model.

Test data is about 20% of the total data and should be completely separate from the training data — which our model should know very well by this point.

In summary, the following chart gives an example of three-way splitting data with a sample configuration of the sub-set sizes.

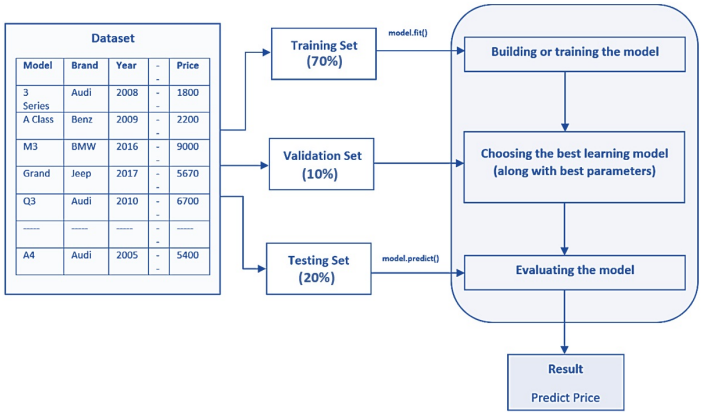


Figure 9.3: The-way data splitting method with an example of sub-sample sizes.

9.3 Cross-validation

Partitioning the available data into three sets drastically reduces the number of samples that can be used for training the model, and the results can depend on a particular random choice for the pair of (train, validation) sets. To address this reliability and stability, a solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing a CV.

9.3.1 K-fold Cross-validation

Many different cross-validation methods are defined based on the following basic k-fold CV, the **training set** is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following is the procedure that uses 5 “folds”:

As an example, we explain the five-fold cross-validation for determining the optional cut-off probability in logistic prediction problems. Without loss of generality, we consider possible cut-off probabilities. For each cut-off probability, we calculate the prediction accuracy of the model based on the confusion matrix using the one-fold of validation data (see the following figure).

With the above process of calculation of the confusion matrix and the accuracy metrics based on the first iteration of the 5-fold CV with a given cut-off probability, we next will explain the logic of finding the optimal cut-off probability (also called hyperparameter) based on a set of given cut-off probabilities.

From the above flow chart, we see that the performance measure reported by 5-fold cross-validation is the average of the values of accuracy computed in the loop. This approach can be computationally expensive but does not waste too

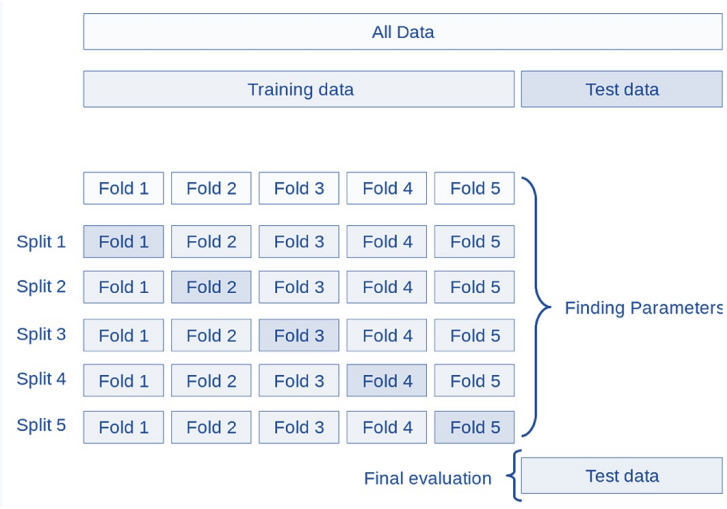


Figure 9.4: Figure 4. Demonstration of a five-fold cross-validation.

much data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems such as inverse inference where the number of samples is very small.

9.3.2 Other Cross-Validation Methods

There are many other cross-validation methods defined using the same logic in k-fold cross-validation. We will not detail these CV methods. Instead, we describe some of them that are occasionally used in practice.

- **Leave-one-out (LOO)** cross-validation is a simple cross-validation. Each training set is created by taking all the records (also called **samples** in machine learning) except one, the validating set being the record (also called **sample** in machine learning) left out. Therefore, this cross-validation procedure does not waste much data as only one sample is removed from the training set.
- **Leave-P-Out** is very similar to Leave-One-Out as it creates all the possible training/test sets by removing samples (i.e., records) from the complete set. Unlike Leave-One-Out and K-Fold, the test sets will overlap for $p > 1$.
- **Stratified K-Fold** is a variation of k-fold that returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set.
- **Leave-One-Group-Out** is a cross-validation scheme that holds out the samples according to a third-party-provided array of integer groups. This group information can be used to encode arbitrary domain-specific pre-

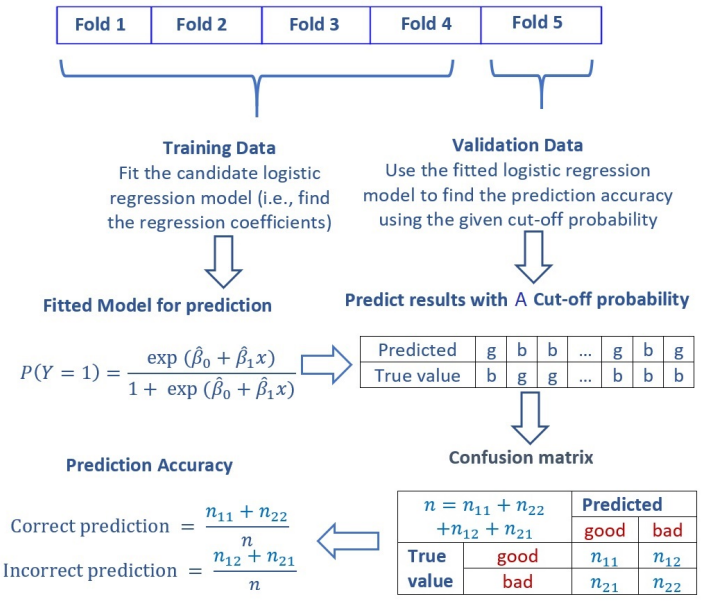


Figure 9.5: Confusion matrix and accuracy metrics.

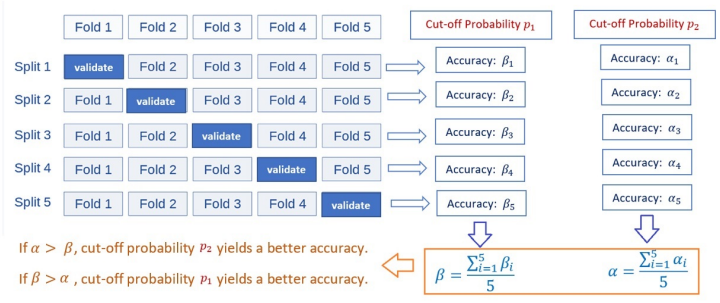


Figure 9.6: The pseudo-program of 5-fold CV for tuning hyperparameter - cut-off probability.

defined cross-validation folds. Each training set is thus constituted by all the samples except the ones related to a specific group.

9.4 Case Study Using Fraud Data

In this section, we use the fraud data at https://pengdsci.github.io/datasets/FraudIndex/fraudidx.csv. The data set has only two variables. The response variable is a binary fraud status variable with character values "good" and "bad". The predictor variable is an index variable. The sample size is 33236. The fraud proportion is about 23.34%. The objective is to build a logistic regression model to predict fraud for future transactions.

9.4.1 Data Partition

Since the same size is large, we split the sample by 70%:30% with 70% data for training and validating models and 30% for testing purposes. The labels (value of the fraud status) of testing and validation data will be removed when calculating the accuracy measures.

```
fraud.data = read.csv("https://pengdsci.github.io/datasets/FraudIndex/fraudidx.csv")[,-1]
## recode status variable: bad = 1 and good = 0
good.id = which(fraud.data$status == " good")
bad.id = which(fraud.data$status == "fraud")
##
fraud.data$fraud.status = 0
fraud.data$fraud.status[bad.id] = 1
nn = dim(fraud.data)[1]
train.id = sample(1:nn, round(nn*0.7), replace = FALSE)
training = fraud.data[train.id,]
testing = fraud.data[-train.id,]
```

9.4.2 Finding Optimal Cut-off Probability

We define a sequence of 20 candidate cut-off probabilities and then use 5-fold cross-validation to identify the optimal cut-off probability for the final detection model.

```
n0 = dim(training)[1]/5
cut.0ff.prob = seq(0,1, length = 22)[-c(1,22)]        # candidate cut off prob
pred.accuracy = matrix(0,ncol=20, nrow=5, byrow = T)  # null vector for storing prediction accura
## 5-fold CV
for (i in 1:5){
  valid.id = ((i-1)*n0 + 1):(i*n0)
  valid.data = training[valid.id,]
  train.data = training[-valid.id,]
  train.model = glm(fraud.status ~ index, family = binomial(link = logit), data = train.data)
```

```
  newdata = data.frame(index= valid.data$index)
  pred.prob = predict.glm(train.model, newdata, type = "response")
  # define confusion matrix and accuracy
  for(j in 1:20){
    pred.status = rep(0,length(pred.prob))
    valid.data$pred.status = as.numeric(pred.prob >cut.0ff.prob[j])
    a11 = sum(valid.data$pred.status == valid.data$fraud.status)
    pred.accuracy[i,j] = a11/length(pred.prob)
  }
}
```

```
## Warning: glm.fit:

## Warning: glm.fit:

## Warning: glm.fit:

## Warning: glm.fit:

## Warning: glm.fit:
```

```
###
avg.accuracy = apply(pred.accuracy, 2, mean)
max.id = which(avg.accuracy ==max(avg.accuracy ))
### visual representation
tick.label = as.character(round(cut.0ff.prob,2))
plot(1:20, avg.accuracy, type = "b",
     xlim=c(1,20),
     ylim=c(0.5,1),
     axes = FALSE,
     xlab = "Cut-off Probability",
     ylab = "Accuracy",
     main = "5-fold CV performance"
     )
axis(1, at=1:20, label = tick.label, las = 2)
axis(2)
segments(max.id, 0.5, max.id, avg.accuracy[max.id], col = "red")
text(max.id, avg.accuracy[max.id]+0.03, as.character(round(avg.accuracy[max.id],4)), c
```

The above figure indicates that the optimal cut-off probability that yields the best accuracy is 0.57.

9.4.3 Reporting Test

This subsection reports the performance of the model using the test data set. Note that the model needs to be fit to the original training data to find the

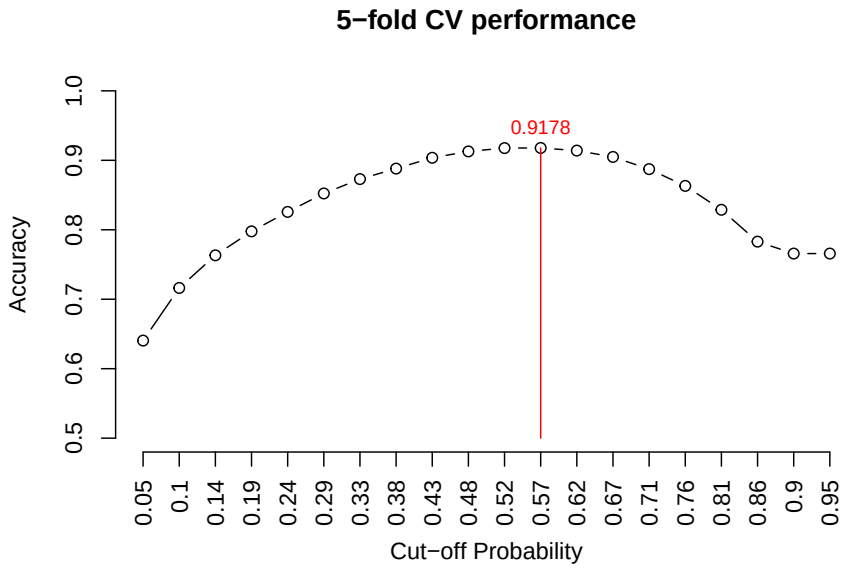


Figure 9.7: Figure 7. 5-fold CV performance plot

regression coefficients and then use the holdout testing sample to find the accuracy.

```
test.model = glm(fraud.status ~ index, family = binomial(link = logit), data = training)
```

```
## Warning: glm.fit:
```

```
newdata = data.frame(index= testing$index)
pred.prob.test = predict.glm(test.model, newdata, type = "response")
testing$test.status = as.numeric(pred.prob.test > 0.57)
a11 = sum(testing$test.status == testing$fraud.status)
test.accuracy = a11/length(pred.prob.test)
kable(as.data.frame(test.accuracy), align='c')
```

test.accuracy
0.9203691

The accuracy is 92.18%. This accuracy indicates that there is no under-fitting. In models and algorithms with multiple feature variables, CV can help detect (and, hence, avoid)over and underfitting.

9.5 Concluding Remarks

We have used 5-fold cross-validation to find the hyperparameter (optimal cut-off probability). This is only one application for estimating a hyperparameter. In practice, we can use cross-validation to select the best models in different model families. For example, we have candidate models such as decision tree-based models, support vector machines, neural networks, and logistic regression models. Cross-validation is a powerful tool for selecting the best model.

Chapter 10

Perfomance Measures of Algorithms and Models

In this note, we summarize the performance measures that are used in data science projects (i.e. in both machine learning and classical statistics). The focus is on the measures related to classification and regression models and algorithms.

Note that, in practice, looking at a single metric may not give us the whole picture of the problem we are trying to solve. In other words, we may want to use a set of metrics to have a concrete evaluation of the candidate models and algorithms.

10.1 Classification Performance Metrics

Since the performance measures for regression models are relatively simple, we focus on the commonly used performance measures of binary classification models and algorithms.

10.1.1 Confusion Matrix for Binary Decision

We have used the logistic regression model as an example to illustrate the cross-validation method. Most of the performance measures are defined based on the confusion matrix.

Consider a binary classification (prediction) model that passed all diagnostics and model selection processes. Any binary decision based on the model will inevitably result in two possible errors that are summarized in the following confusion matrix (as mentioned and used in the previous case study).

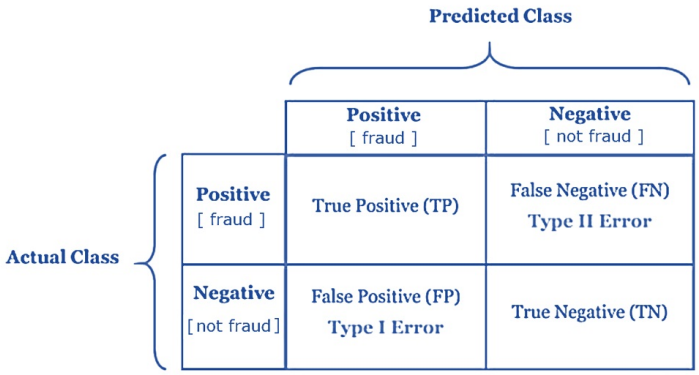


Figure 10.1: The layout of the binary decision confusion matrix.

The following are a few probabilities that will be used as the element in the definition of performance measures.

- **True Positive (TP)** is the number of correct predictions that an example is positive which means positive class correctly identified as positive - P[Predicted Positive | Actual Positive].
- **False Negative (FN)** is the number of incorrect predictions that an example is negative which means positive class incorrectly identified as negative - P[Predicted Negative | Actual Positive].
- **False positive (FP)** is the number of incorrect predictions that an example is positive which means negative class incorrectly identified as positive - P[Predicted Positive | Actual Negative].
- **True Negative (TN)** is the number of correct predictions that an example is negative which means negative class correctly identified as negative - P[Predicted Negative | Actual Negative].

The above conditional probabilities are defined by conditioning on the **actual status**. These probabilities are used to assess the model performance in the stage of model development.

One of the important steps in the data science process is to monitor the performance of the deployed models in the production environment. We can use a different set of conditional probabilities for this purpose.

- **Positive Predictive Value (PPV)** is the percentage of predictive positives that are confirmed to be positive - P[Confirmed Positive | Predicted Positive]. In the clinical term, PPV is the percentage of patients with a positive test who actually have the disease.
- **Negative Predictive Value (NPV)** is the percentage of predictive posi-

tives that are confirmed to be positive - P[Confirmed Negative | Predicted Negative]. In the clinical term, NPV is the percentage of patients with a negative test who do not have the disease.

10.1.2 Local Performance Measures (for Model Development)

For ease of understanding, we use the following hypothetical confusion matrix based on the clinical binary decision.

	Disease present	Disease absent
Test positive	a (TP)	b (FP)
Test negative	c (FN)	d (TN)

TP: True positive, FP: False positive, FN: False negative, TN: True negative

Figure 10.2: The layout of the clinical binary decision confusion matrix.

The following performance measures are defined based on the above confusion matrix.

- **Classification Accuracy** measures the percentage of labels that are correctly predicted.

$$\text{accuracy} = \frac{a+d}{a+b+c+d}$$

- **Precision** is a valid performance measure for a class whose distribution is imbalanced (one class is more frequent than others). In this case, even if we predict all samples as the most frequent class, we would get a high accuracy rate. This does not make sense at all because your model is not learning anything, and is just predicting everything as the top class. **Precision** measures the percentage of true positives among all predictive positives.

$$\text{precision} = \frac{a}{a+b}$$

- **Recall** is another important metric, which is defined as the fraction of samples from a class that is correctly predicted by the model. More formally,

$$\text{Recall} = P[\text{predict disaese}|\text{Actual disease}] = \frac{a}{a+c}$$

- **F1 Score**: Depending on the application, we may want to give higher priority to recall or precision. But there are many applications in which

both recall and precision are important. Therefore, it is natural to think of a way to combine these two into a single metric. One popular metric which combines precision and recall is called F1-score, which is the harmonic mean of precision and recall defined as

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The generalized version of the F-score is defined below. As we can see F1-score is a special case of F_β when $\beta = 1$.

$$F_1 = \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2(\text{precision} + \text{recall})}$$

It is good to mention that there is always a trade-off between the precision and recall of a model. If we want to make the precision too high, we would end up seeing a drop in the recall rate, and vice versa.

10.1.3 Global Performance Measures

Sensitivity and specificity are two other popular metrics mostly used in medical and biology-related fields. They are used as building blocks for well-known global measures such as ROC and the area under the curve (AUC). They are defined in the forms of conditional probability in the following based on the above clinical confusion matrix.

- **Sensitivity (True Positive Rate, Recall)** - The probability of those who received a positive result on this test out of those who actually have a disease (when judged by the 'Gold Standard'). It is the same as **recall**.

$$\text{sensitivity} = \frac{a}{a + c}$$

- **Specificity (True Negative Rate)** - The probability of those who received a negative result on this test out of those who do not actually have the disease (when judged by the 'Gold Standard').

$$\text{specificity} = \frac{d}{b + d}$$

Next, we define a metric to assess the global performance measure for the binary decision models and algorithms. From the previous case study of cross-validation. Each candidate cut-off probability defines a confusion matrix and, consequently, sensitivity and specificity associated with the confusion matrix.

- **An ROC curve (receiver operating characteristic curve)** is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: sensitivity and (1 - specificity). Note that the (1 - specificity = false positive rate).

In other words, the ROC curve is the plot of the False Positive Rate (FPR) against the True Positive Rate (TPR) calculated from each decision boundary (such as the cut-off probability in logistic models).

https://github.com/pengdsci/STA551/blob/main/w06/img/w06-Animated-ROC.gif

The primary use of the ROC is to compare the global performance between candidate models (that are not necessarily to be within the same family). As an illustrative example, the following ROC curves are calculated based on a logistic regression model and a support vector machine (SVM). Both are binary classifiers.

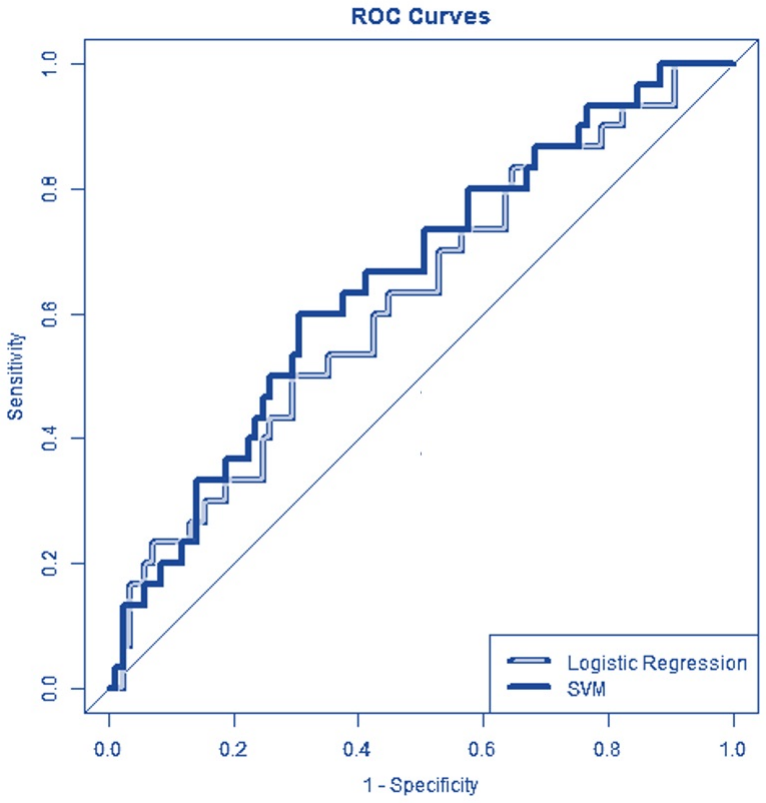


Figure 10.3: Figure 4. Using ROC for model selection.

We can see that the SVM is globally better than the logistic regression. However, at some special decision boundaries, the logistic regression model is locally better than SVM.

- **Area Under The Curve (AUC)**

If two ROC curves intersect at least one point, we may want to report the area under the curves (AUC) to compare the global performance between the two corresponding models. See the illustrative example below.

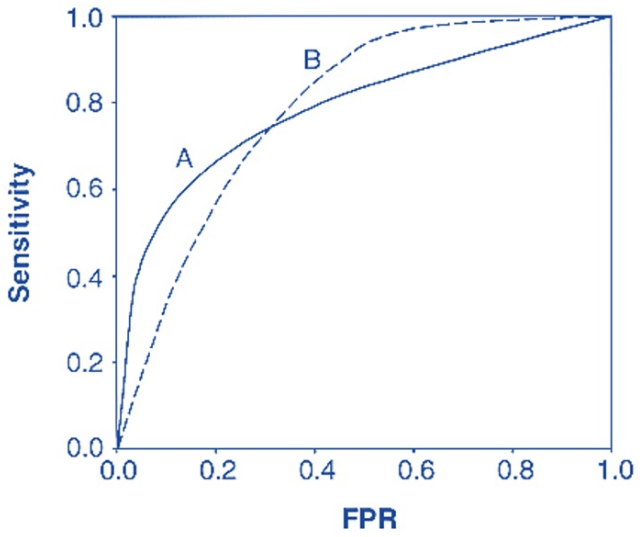


Figure 10.4: Using ROC for model selection.

10.2 Case Study - Logistic regression model with the fraud data

This case study shows how to calculate the local and global performance metrics for logistic predictive models. We have used the confusion matrix in the case study in the previous note. Here we will use the optimal cut-off probability as the decision threshold to define a confusion matrix and then define the performance measure based on this matrix.

We reload the data and create the training and testing data sets. We pretend the optimal cut-off probability is based on what is obtained through the CV. The testing data set will be used to report the local and global performance measures.

```
fraud.data = read.csv("https://pengdsci.github.io/datasets/FraudIndex/fraudidx.csv")[,-1]
## recode status variable: bad = 1 and good = 0
good.id = which(fraud.data$status == " good")
bad.id = which(fraud.data$status == "fraud")
##
fraud.data$fraud.status = 0
fraud.data$fraud.status[bad.id] = 1
nn = dim(fraud.data)[1]
train.id = sample(1:nn, round(nn*0.7), replace = FALSE)
training = fraud.data[train.id,]
testing = fraud.data[-train.id,]
```

10.2.1 Local Performance Meaures

Since we have identified the optimal cut-off probability to be 0.57. Next, we will use the **testing data** set to report the local measures.

```
test.model = glm(fraud.status ~ index, family = binomial(link = logit), data = training)
```

```
## Warning: glm.fit:
```

```
newdata = data.frame(index= testing$index)
pred.prob.test = predict.glm(test.model, newdata, type = "response")
testing$test.status = as.numeric(pred.prob.test > 0.57)
### components for defining various measures
TN = sum(testing$test.status ==0 & testing$fraud.status==0)
FN = sum(testing$test.status ==0 & testing$fraud.status ==1)
FP = sum(testing$test.status ==1 & testing$fraud.status==0)
TP = sum(testing$test.status ==1 & testing$fraud.status ==1)
###
sensitivity = TP / (TP + FN)
specificity = TN / (TN + FP)
###
precision = TP / (TP + FP)
recall = sensitivity
F1 = 2*precision*recall/(precision + recall)
metric.list = cbind(sensitivity = sensitivity,
                    specificity = specificity,
                    precision = precision,
                    recall = recall,
                    F1 = F1)
kable(as.data.frame(metric.list), align='c', caption = "Local performance metrics")
```

Table 10.1: Local performance metrics

sensitivity	specificity	precision	recall	F1
0.7171502	0.9761374	0.9023081	0.7171502	0.7991443

10.2.2 Global Measure: ROC and AUC

In order to create an ROC curve, we need to select a sequence of decision thresholds and calculate the corresponding sensitivity and specificity.

CAUTION: ROC and AUC are used for model selection, we still use the **training data** to construct the ROC and calculate the AUC.

```
cut.off.seq = seq(0, 1, length = 100)
sensitivity.vec = NULL
specificity.vec = NULL
###
training.model = glm(fraud.status ~ index, family = binomial(link = logit), data = tra
```

```
## Warning: glm.fit:
```

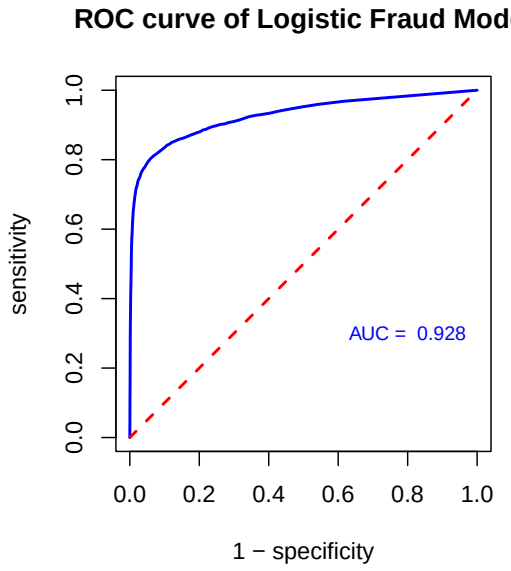
```
newdata = data.frame(index= training$index)
pred.prob.train = predict.glm(training.model, newdata, type = "response")
for (i in 1:100){
  training$train.status = as.numeric(pred.prob.train > cut.off.seq[i])
### components for defining various measures
TN = sum(training$train.status == 0 & training$fraud.status == 0)
FN = sum(training$train.status == 0 & training$fraud.status == 1)
FP = sum(training$train.status == 1 & training$fraud.status == 0)
TP = sum(training$train.status == 1 & training$fraud.status == 1)
###
sensitivity.vec[i] = TP / (TP + FN)
specificity.vec[i] = TN / (TN + FP)
}
one.minus.spec = 1 - specificity.vec
sens.vec = sensitivity.vec
## A better approx of ROC, need library {pROC}
  prediction = pred.prob.train
  category = training$fraud.status == 1
  ROCobj <- roc(category, prediction)
```

```
## Setting levels: control = FALSE, case = TRUE
```

```
## Setting direction: controls < cases
```

```
  AUC = round(auc(ROCobj),4)
##
par(pty = "s")    # make a square figure
```

```
plot(one.minus.spec, sens.vec, type = "l", xlim = c(0,1), ylim = c(0,1),
     xlab ="1 - specificity",
     ylab = "sensitivity",
     main = "ROC curve of Logistic Fraud Model",
     lwd = 2,
     col = "blue", )
segments(0,0,1,1, col = "red", lty = 2, lwd = 2)
#AUC = round(sum(sens.vec*(one.minus.spec[-101]-one.minus.spec[-1])),4)
text(0.8, 0.3, paste("AUC = ", AUC), col = "blue", cex = 0.8)
```



Chapter 11

From Statistics Models to ML Algorithms

There are a lot of debates on the difference between statistics and machine learning in statistics and machine learning communities. It is sure that statistics and machine learning are not the same although there is an overlap. A major difference between machine learning and statistics is indeed their purpose.

- Statistics focuses on the inference and interpretability of the relationships between variables.
- Machine learning focuses on the accuracy of the prediction of future values of (response) variables and detecting hidden patterns. Machine learning is traditionally considered to be a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior.

A lot of statistical models can make predictions, but predictive accuracy is not their strength while machine learning models provide various degrees of interpretability sacrifice interpretability for predictive power. For example, regularized regressions as machine learning algorithms are interpretable but neural networks (particularly multi-layer networks) are almost uninterpretable.

Statistics and machine learning are two of the key players in data science. As data science practitioners, our primary interest is to develop and select the right tools to build data solutions for real-world applications.

11.1 Some Technical Terms and ML Types

Before discussing machine learning algorithms, we first introduce some technical terms in ML and types of machine learning algorithms.

11.1.1 Machine Learning Problems and Jargon

Statistics	Machine Learning	Comments
data point, record, row of data	example, instance	Both domains also use "observation," which can refer to a single measurement or an entire vector of attributes depending on context.
response variable, dependent variable	label, output	Both domains also use "target." Since practically all variables depend on other variables, the term "dependent variable" is potentially misleading.
regressions	supervised learners, machines	Both estimate output(s) in terms of input(s).
regression intercept	bias	the default prediction of a linear model in the special case where all inputs are 0.
Maximize the likelihood to estimate model parameters	Minimize the entropy to derive the best parameters in categorical regression or maximize the likelihood for continuous regression.	For discrete distributions, maximizing the likelihood is equivalent to minimizing the entropy.
logistic/multinomial regression	maximum entropy, MaxEnt	They are equivalent except in special multinomial settings like ordinal logistic regression.

11.1.2 Types of Machine Learning Problems

Based on the type of problems that we are trying to solve, we can classify the Machine learning problem into three different categories.

Classification Problem: Classification is a problem that requires machine learning algorithms to assign a class label (response value) to examples (vector of predictor values) from the problem domain. A very intuitive example is classifying credit card transactions into two labels "fraud" or "not a fraud."

Regression Problem: Regression is a problem that requires machine learning algorithms to predict continuous (response) variables. An elementary example will be to predict the temperature of the city. (Temperature can take any numeric value between -50 to +150 degrees.)

Clustering Problem: Clustering is a type of problem that requires the use of Machine Learning algorithms to group the given data records into a specified number of cohesive units. A simple example will be to group the credit card holders according to their monthly spending.

There are many machine learning algorithms and statistical models running in the practice. This note primarily overviews the commonly used methods for classification and prediction.

11.2 Categories of Machine Learning

There are different ways to categorize machine learning algorithms using different criteria. But none of them are perfect although each has its own merits.

11.2.1 Supervised Learning

Input data is called training data and has a known label (i.e., the response in statistics) or result such as spam/not-spam (classification) or a stock price at a time (forecasting/prediction).

Supervised learning is a subcategory of machine learning. It is defined by its use of labeled data (i.e., the response is available in the data) sets to train algorithms that classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the **cross-validation** process.

Supervised learning helps organizations solve a variety of real-world problems at scale, such as classifying spam in a separate folder from an email inbox, classifying credit transactions into fraud or non-fraud categories, etc.

There are a lot of supervised machine learning algorithms. Example algorithms include **Logistic Regression**, Decision Trees, Support Vector Machines, Neural Networks, etc.

11.2.2 Unsupervised Learning

Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled (i.e., no response variable is used or required in) data sets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information makes it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition. We have briefly mentioned this clustering algorithm when discussing feature engineering.

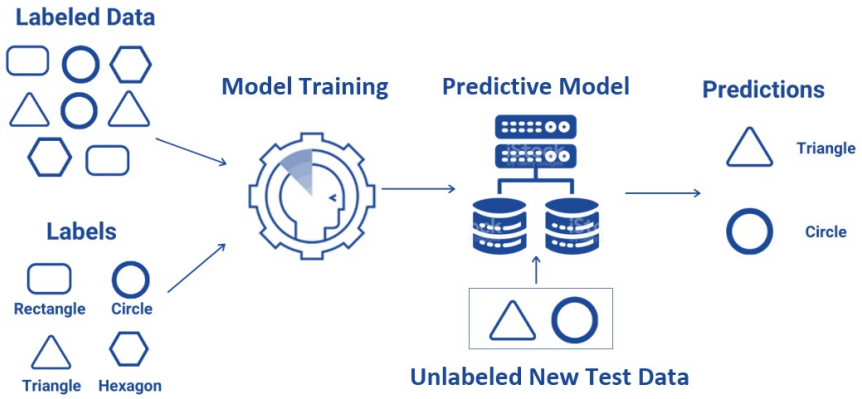


Figure 11.1: Illustration of supervised machine learning.

Figure 11.2: Illustration of unsupervised machine learning.

Example problems are clustering, dimensionality reduction, and association rule learning.

Example algorithms include clustering analysis and K-Means.

11.2.3 Semi-Supervised Learning

Input data is a mixture of labeled and unlabeled examples, that is, some records have response values and some don't.

We can use either a predictive ML model trained based on the labeled data to predict the labels of the unlabeled data or a clustering algorithm such as k-means to assign labels to the unlabeled data to make a larger pseudo-labeled data for ML model with a better performance.

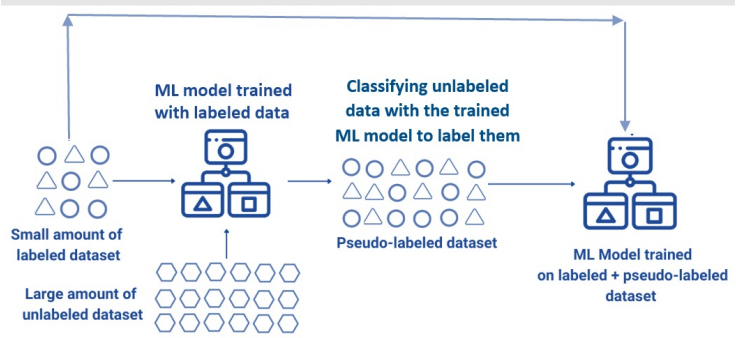


Figure 11.3: Illustration of semi-supervised machine learning.

The following chart gives a non-numerical example of how semi-supervised learning works.

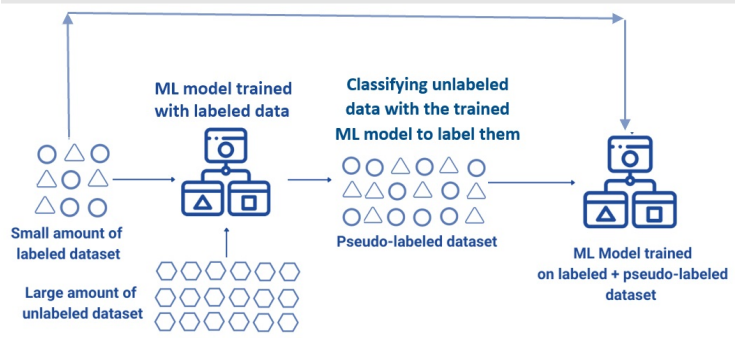


Figure 11.4: Illustration of how semi-supervised machine learning works.

Semi-supervised learning and reinforcement learning are hot topics in the machine learning community. Example algorithms are extensions to other flexible methods that make assumptions about how to model the unlabeled data.

11.3 From Statistics to Machine Learning

As mentioned earlier, most statistical models can be used as a learning algorithm. In this section, we use two examples to demonstrate the equivalent between some of the basic statistical models and their corresponding machine learning models.

11.3.1 Logistic Regression Model Revisited

Recall that the binary logistic regression model with n feature variables $x_1, x_2, \cdots, x_n$ is given by

$$P[Y=1] = \frac{\exp(w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n)}{1 + \exp(w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n)} = \frac{\exp(w_0 + \sum_{i=1}^n w_ix_i)}{1 + \exp(w_0 + \sum_{i=1}^n w_ix_i)}$$

where $w_0, w_1, \cdots, w_n$ are regression coefficients. The model can be rewritten as

$$P[Y=1] = \frac{1}{1 + \exp[-(w_0 + \sum_{i=1}^n w_ix_i)]}$$

Let

$$f(x) = \frac{1}{1+\exp(-x)}.$$

This function is the well-known **logistic function**. The curve of the logistic function is given by

```
x = seq(-5, 5, length = 100)
y = 1/(1+exp(-x))
plot(x,y, type = "l", lwd = 2, xlab =" ", ylab = " ",
     main = "Logistic Curve", col = "blue")
```

Using this logistic function, we can re-express the logistic model as

$$P[Y=1] = f\left(w_0 + \sum_{i=1}^n w_ix_i\right)$$

Next, we represent the above logistic regression model in the following diagram.

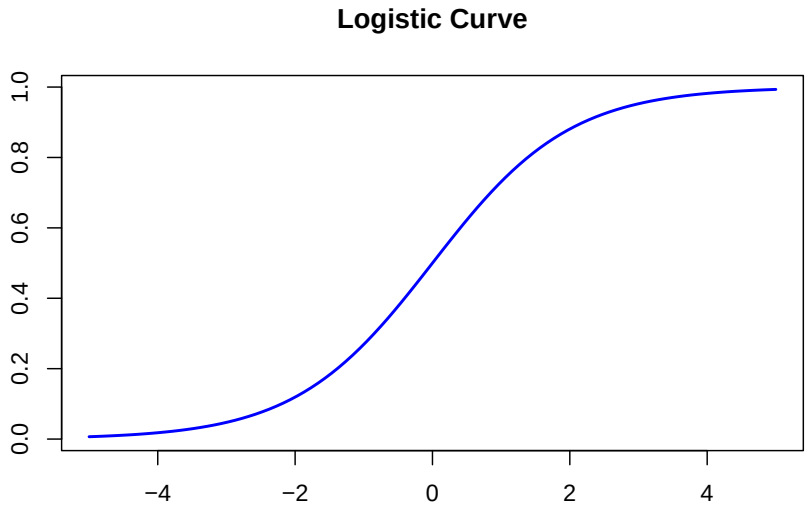


Figure 11.5: The curve of the logistic function.

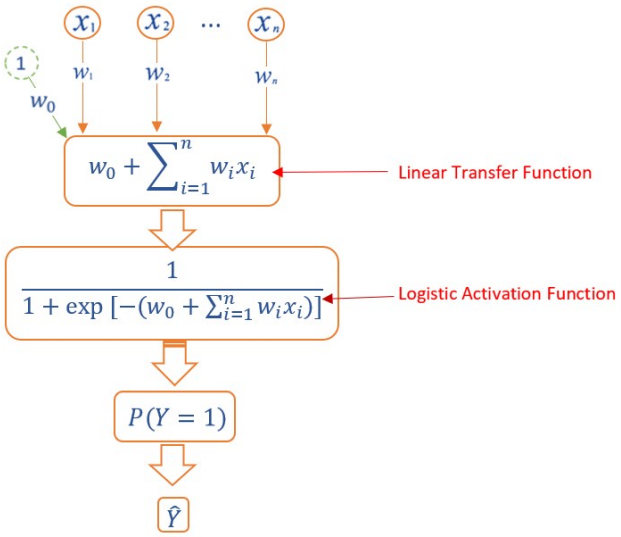


Figure 11.6: Diagram Representation of logistic regression models.

We will see that the above diagram of the logistic regression model is the architecture of the basic single layer **sigmoid** neural network model - perceptron.

11.3.2 Single Layer Neural Network - Perceptron

Perceptron is a type of artificial neural network, which is a fundamental concept in machine learning. Its architecture is the same as the diagram of the logistic regression model.

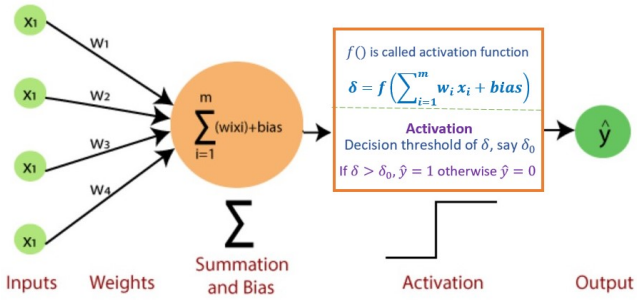


Figure 11.7: Architecture of Single layer neural network models (perceptron).

Each input x_i has an associated weight w_i (like regression coefficient). The sum of all weighted inputs, $\sum_{i=1}^{n} w_i x_i$, is then passed through a nonlinear activation function $f()$, to transform the pre-activation level of the neuron to an output y_j. For simplicity, the bias term is set to **b** which is equivalent to the intercept of a regression model.

To summarize, we explicitly list the major components of perceptron in the following.

- **Input Layer**: The input layer consists of one or more input neurons, which receive input signals from the external world or from other layers of the neural network.
- **Weights**: Each input neuron is associated with a weight, which represents the strength of the connection between the input neuron and the output neuron.
- **Bias**: A bias term is added to the input layer to provide the perceptron with additional flexibility in modeling complex patterns in the input data.
- **Activation Function**: The activation function determines the output of the perceptron based on the weighted sum of the inputs and the bias term. Common activation functions used in perceptrons include the `step function`, `sigmoid function`, and `ReLU function`, etc.
- **Output**: The output of the perceptron is a single binary value, either 0 or 1, which indicates the class or category to which the input data belongs.

Note that when the sigmoid (i.e., logistic) function

$$f(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}.$$

is used in the perceptron. This means that the single-layer perception with logistic activation is equivalent to the binary logistic regression.

Remarks:

1. The output of the above perceptron network is binary, i.e., $\hat{Y} = 0$ or 1 since an implicit decision boundary based on the sign of the value of the transfer function $\sum_{i=1}^{m} w_i x_i + b$. In the sigmoid perceptron network, this is equivalent to setting the threshold probability to 0.5. To see this, not that, if $\sum_{i=1}^{m} w_i x_i + b = 0$, then

   $$P\left[Y = 1 \middle| \sum_{i=1}^{m} w_i x_i + b\right] = \frac{1}{1+\exp\left[-(\sum_{i=1}^{m} w_i x_i + b)\right]} = \frac{1}{1+\exp(0)} = \frac{1}{2}$$

   This means, if the cut-off probability 0.5 is used in the logistic predictive model, this logistic predictive model is equivalent to the perceptron with sigmoid being the activation function.

2. There are several other commonly used activation functions in perceptron. The sigmoid activation function is only one of them. This implies that the binary logistic regression model is a special perceptron network model.

11.3.3 Multi-layer Perceptron

A Multi-Layer Perceptron (MLP) contains one or more hidden layers (apart from one input and one output layer). While a single-layer perceptron can only learn linear functions, a multi-layer perceptron can also learn non-linear functions. The following is an illustrative MLP.

The `major components` in the above MLP are described in the following.

Input Layer: The Input layer has three nodes. The Bias node has a value of 1. The other two nodes take X_1 and X_2 as external inputs (which are numerical values depending upon the input data set). No computation is performed in the Input layer, so the outputs from nodes in the Input layer are 1, X_1, and X_2 respectively, which are fed into the Hidden Layer.

Hidden Layer: The Hidden layer also has three nodes with the Bias node having an output of 1. The output of the other two nodes in the Hidden layer depends on the outputs from the Input layer (1, X_1, X_2) as well as the weights associated with the connections (edges). Figure 16 shows the output calculations

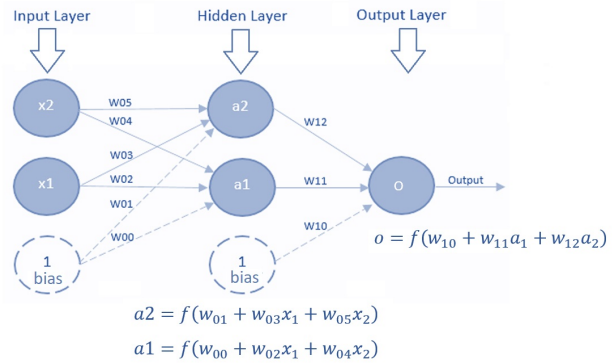


Figure 11.8: Multi-layer perceptron.

for the hidden nodes. Remember that $f()$ refers to the activation function. These outputs are then fed to the nodes in the Output layer.

Output Layer: The Output layer has two nodes that take inputs from the Hidden layer and perform similar computations as shown in the above figure. The values calculated (Y_1 and Y_2) as a result of these computations act as outputs of the Multi-Layer Perceptron.

11.3.4 Commonly Used Activation Functions

The sigmoid function is only one of the activation functions used in neural networks. The table below lists several other commonly used activation functions in neural network modeling.

11.3.5 Algorithms for Estimating Weights

We know that the estimation of the regression coefficient in logistic regression is to maximize the likelihood function defined based on the binomial distribution. Algorithms such as Newton and its variants, scoring methods, etc. are used to obtain the estimated regression coefficients.

In neural network models, the weights are estimated by minimizing the loss function (also called cost function) when training neural networks. The loss function could be defined as **mean square error (MSE)** for regression tasks and **cross-entropy (cs)** for classification tasks.

Learning algorithms **forward and backward propagation** that depend on each other are used in minimizing the underlying **loss function**.

- **Forward propagation** is where input data is fed through a network, in a forward direction, to generate an output. The data is accepted by hidden layers and processed, as per the activation function, and moves to

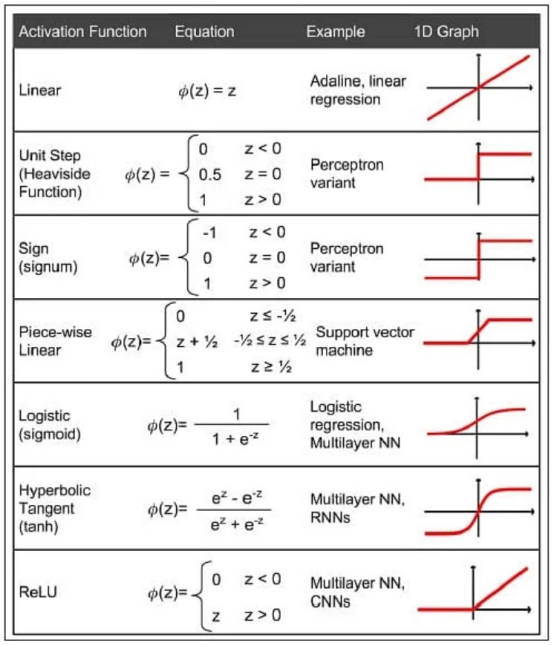


Figure 11.9: Popular activation functions in neural networks.

the successive layer. During forward propagation, the activation function is applied, based on the weighted sum, to make the neural network flow non-linearly using bias. Forward propagation is the way data moves from left (input layer) to right (output layer) in the neural network.

- **Backpropagation** is used to improve the prediction accuracy of a node is expressed as a loss function or error rate. Backpropagation calculates the slope of (gradient) a **loss function** of other weights in the neural network and updates the weights using gradient descent through the learning rate.

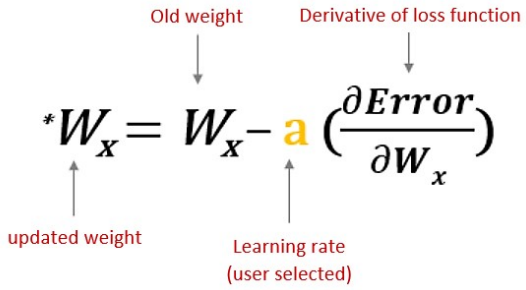


Figure 11.10: Updating weights with backpropagation algorithm.

The general architecture of the backpropagation network model is depicted in the following diagram.

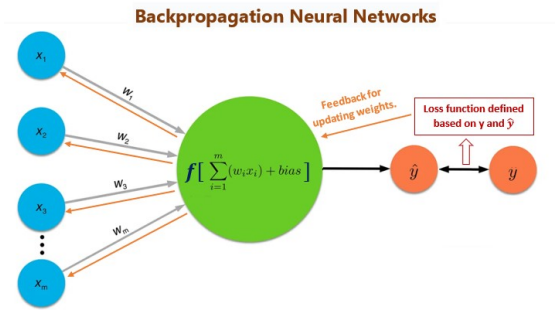


Figure 11.11: The idea of backpropagation neural networks.

The algorithm of backpropagation is not used in classical statistics. This is why the neural network model outperformed the classical logistic model in terms of predictive power.

The R library `neuralnet` has the following five algorithms:

backprop - traditional `backpropagation`.

rprop+ - resilient backpropagation with weight backtracking.

rprop- - resilient backpropagation without weight backtracking.

sag - modified globally convergent algorithm (gr-prop) with the smallest absolute gradient.

slr - modified globally convergent algorithm (gr-prop) with the smallest learning rate.

11.4 Implementing NN with R

Several R libraries can run neural network models. `nnet` is the simplest one that only implements single-layer networks. `neuralnet` can run both single-layer and multiple-layer neural networks. `RSNNS` (R Stuttgart Neural Network Simulator) is a wrapper of multiple R libraries that implements different network models.

11.4.1 Syntax of `neuralnet`

We use `neuralnet` library to run the neural network model in the example (code for installing and loading this library is placed in the setup code chunk).

The syntax of `neuralnet()` is given below

```
neuralnet(formula,
          data,
          hidden = 1,
          threshold = 0.01,
          stepmax = 1e+05,
          rep = 1,
          startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor =list(minus = 0.5, plus = 1.2),
          learningrate=NULL,
          lifesign = "none",
          lifesign.step = 1000,
          algorithm = "rprop+",
          err.fct = "sse",
          act.fct = "logistic",
          linear.output = TRUE,
          exclude = NULL,
          constant.weights = NULL,
          likelihood = FALSE)
```

The detailed `help document` can be found at https://www.rdocumentation.org/packages/neuralnet/versions/1.44.2/topics/neuralnet.

11.4.2 Feature Conversion for `neuralnet`

`neuralnet()` requires all features to be in the numeric form (dummy variable for categorical features, normalization of numerical features). The model formula in `neuralnet()` requires dummy variables to be explicitly defined. It is also highly recommended to scale all numerical features before being included in the network model. The objective is to find all feature names (numeric and all dummy variables) and write them in the model formula like the one in `glm`: `response ~ var_1 + var_2 + ... +var_k`

To explain the modeling process in detail, we will outline major steps in the following subsections.

11.4.3 Numeric Feature Scaling

There are different types of scaling and standardization. The one we use in the following has

$$scaled.var = \frac{orig.var - \min(orig.var)}{\max(orig.var) - \min(orig.var)}$$

The scaled numeric feature is unitless (similar to the well-known z-score transformation).

```
Pima = read.csv("https://pengdsci.github.io/STA551/w03/AnalyticPimaDiabetes.csv")[,-1]
Pima$pedigree = (Pima$pedigree-min(Pima$pedigree))/(max(Pima$pedigree)-min(Pima$pedigr
Pima$impute.log.insulin = (Pima$impute.log.insulin-min(Pima$impute.log.insulin))/(max(
Pima$impute.triceps = (Pima$impute.triceps-min(Pima$impute.triceps))/(max(Pima$impute.
```

11.4.4 Extract All Feature Names

In practical applications, there may be many categorical features in the model and each category could have many categories. It is practically infeasible to write all resulting dummy features explicitly. We can use the R function to extract variables from a model formula that will be used in a model. Make sure, all categorical feature variables must be defined in a non-numerical form (i.e., should not be numerically encoded). We can also use the R function `relevel()` to change the baseline of an unordered categorical feature variable.

Next, we use the R function `model.matrix()` to extract the names of all feature variables (including implicitly defined dummy feature variables from `model.matrix()`).

```
PimaMtx0 = model.matrix(~ mass + pedigree + impute.log.insulin + impute.triceps + grp.g
colnames(PimaMtx0)
```

```
PimaMtx = model.matrix(~ ., data = Pima)
colnames(PimaMtx)
```

```
##  [1] "(Intercept)"          "mass"                 "pedigree"             "impute.lo
##  [5] "impute.triceps"       "grp.glucose[117,137]" "grp.glucose> 137"     "grp.diast
##  [9] "grp.diastolic> 90"    "grp.age[45, 64]"      "grp.age65+"           "grp.pregn
## [13] "grp.pregnant2"        "grp.pregnant3-4"      "grp.pregnant5-7"      "grp.pregn
## [17] "diabetespos"
```

There are some naming issues in the above dummy feature variables for network modeling (although they are good for regular linear and generalized linear regression models). We need to rename them by excluding special characters in order to build neural network models. These issues can be avoided at the stage of feature engineering (if we initially planned to build neural network models). Next, we clean up the variables before defining the network model formula.

```
colnames(PimaMtx)[4] <- "logInsulin"
colnames(PimaMtx)[5] <- "triceps"
colnames(PimaMtx)[6] <- "glucose117To137"
colnames(PimaMtx)[7] <- "glucoseGt137"
colnames(PimaMtx)[8] <- "diastolic80To90"
colnames(PimaMtx)[9] <- "diastolicGt90"
colnames(PimaMtx)[10] <- "age45To64"
colnames(PimaMtx)[11] <- "age65older"
colnames(PimaMtx)[12] <- "pregnant1"
colnames(PimaMtx)[13] <- "pregnant2"
```

```
colnames(PimaMtx)[14] <- "pregnant3To4"
colnames(PimaMtx)[15] <- "pregnant5To7"
colnames(PimaMtx)[16] <- "pregnant8Plus"
```

11.4.5 Define Model Formula

For convenience, we encourage you to use **CamelCase** notation (`CamelCase` is a way to separate the words in a phrase by making the first letter of each word capitalized and not using spaces) in naming feature variables.

```
columnNames = colnames(PimaMtx)
columnList = paste(columnNames[-c(1,length(columnNames))], collapse = "+")
columnList = paste(c(columnNames[length(columnNames)],"~",columnList), collapse="")
modelFormula = formula(columnList)
modelFormula
```

```
## diabetespos ~ mass + pedigree + logInsulin + triceps + glucose117To137 +
##     glucoseGt137 + diastolic80To90 + diastolicGt90 + age45To64 +
##     age65older + pregnant1 + pregnant2 + pregnant3To4 + pregnant5To7 +
##     pregnant8Plus
```

11.4.6 Training and Testing NN Model

We follow the routine steps for building a neural network model to predict diabetes.

11.4.6.1 Data Splitting

We split the data into 70% for training the neural network and 30% for testing.

```
n = dim(PimaMtx)[1]
testID = sample(1:n, round(n*0.7), replace = FALSE)
testDat = PimaMtx[testID,]
trainDat = PimaMtx[-testID,]
```

11.4.6.2 Build NN Model

```
NetworkModel = neuralnet(modelFormula,
                         data = trainDat,
                         hidden = 1,               # single layer NN
                         rep = 1,                  # number of replicates in training NN
                         threshold = 0.01,         # threshold for the partial derivatives as sto
                         learningrate = 0.1,       # user selected rate
                         algorithm = "rprop+"
```

```
                                )
kable(NetworkModel$result.matrix)
```

error	14.0370974
reached.threshold	0.0097069
steps	21874.0000000
Intercept.to.1layhid1	90.1823029
mass.to.1layhid1	-2.2395240
pedigree.to.1layhid1	-42.0089980
logInsulin.to.1layhid1	23.6797205
triceps.to.1layhid1	38.6772974
glucose117To137.to.1layhid1	-12.6254994
glucoseGt137.to.1layhid1	-34.6507358
diastolic80To90.to.1layhid1	2.1930003
diastolicGt90.to.1layhid1	-5.3138288
age45To64.to.1layhid1	-10.9854888
age65older.to.1layhid1	36.4614114
pregnant1.to.1layhid1	9.7337152
pregnant2.to.1layhid1	-16.5880762
pregnant3To4.to.1layhid1	-7.5045887
pregnant5To7.to.1layhid1	-10.2882513
pregnant8Plus.to.1layhid1	-0.7988814
Intercept.to.diabetespos	0.8903213
1layhid1.to.diabetespos	-0.7406113

```
plot(NetworkModel, rep="best")
```

```
logiModel = glm(factor(diabetes) ~., family = binomial, data = Pima)
pander(summary(logiModel)$coefficients)
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-5.303	0.7241	-7.324	2.411e-13
mass	0.08627	0.01911	4.514	6.357e-06
pedigree	2.366	0.6969	3.395	0.000685
impute.log.insulin	0.3452	0.7145	0.4831	0.629
impute.triceps	-0.06055	1.082	-0.05595	0.9554
grp.glucose[117,137]	0.8903	0.2431	3.663	0.0002494
grp.glucose> 137	1.92	0.2639	7.277	3.414e-13
grp.diastolic[80,90]	-0.1163	0.2245	-0.5179	0.6045
grp.diastolic> 90	-0.242	0.4281	-0.5654	0.5718
grp.age[45, 64]	0.3513	0.2616	1.343	0.1793
grp.age65+	-0.6649	0.6581	-1.01	0.3124
grp.pregnant1	-0.2712	0.3609	-0.7517	0.4523
grp.pregnant2	-0.2223	0.3913	-0.568	0.5701
grp.pregnant3-4	0.435	0.3381	1.287	0.1982

	Estimate	Std. Error	z value	Pr(>|z|)
grp.pregnant5-7	0.6543	0.3326	1.967	0.04919
grp.pregnant8+	1.107	0.3553	3.114	0.001844

11.4.6.3 About Cross-validation in Neural Network

The algorithm of Cross-validation is primarily used for tuning hyper-parameters. For example, in the sigmoid perceptron, the optimal cut-off scores for the binary decision can be obtained through cross-validation. One of the important hyperparameters in the neural network model is the learning rate α (in the backpropagation algorithm) that impacts the learning speed in training neural network models.

```
n0 = dim(trainDat)[1]/5
cut.off.score = seq(0,1, length = 22)[-c(1,22)]        # candidate cut off prob
pred.accuracy = matrix(0,ncol=20, nrow=5, byrow = T)  # null vector for storing prediction accura
###
for (i in 1:5){
  valid.id = ((i-1)*n0 + 1):(i*n0)
  valid.data = trainDat[valid.id,]
  train.data = trainDat[-valid.id,]
  ####
  train.model = neuralnet(modelFormula,
                          data = train.data,
                          hidden = 1,                # single layer NN
                          rep = 1,                   # number of replicates in training NN
                          threshold = 0.01,          # threshold for the partial derivatives as sto
                          learningrate = 0.1,        # user selected rate
                          algorithm = "rprop+"
                          )
    pred.nn.score = predict(train.model, valid.data)
    for(j in 1:20){
    #pred.status = rep(0,length(pred.nn.score))
    pred.status = as.numeric(pred.nn.score > cut.off.score[j])
    a11 = sum(pred.status == valid.data[,17])
    pred.accuracy[i,j] = a11/length(pred.nn.score)
  }
}
###
avg.accuracy = apply(pred.accuracy, 2, mean)
max.id = which(avg.accuracy ==max(avg.accuracy ))
### visual representation
tick.label = as.character(round(cut.off.score,2))
plot(1:20, avg.accuracy, type = "b",
     xlim=c(1,20),
```

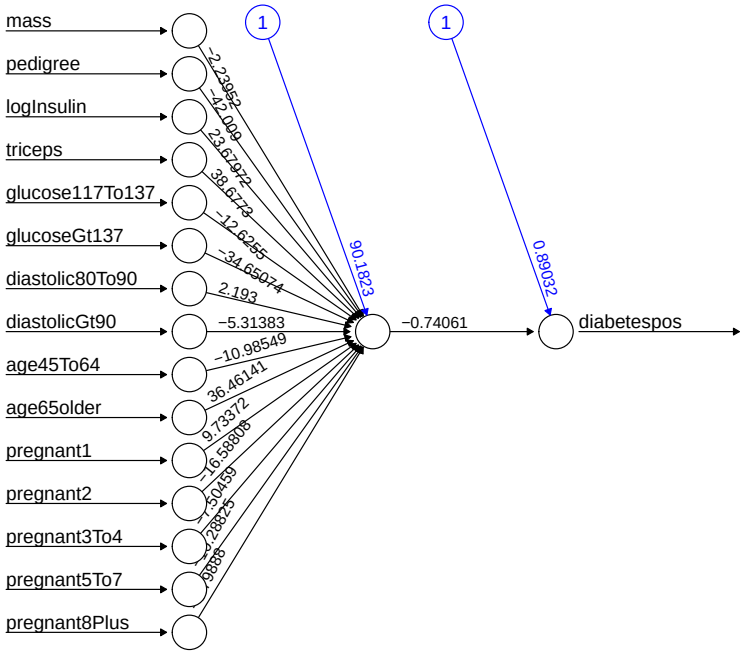
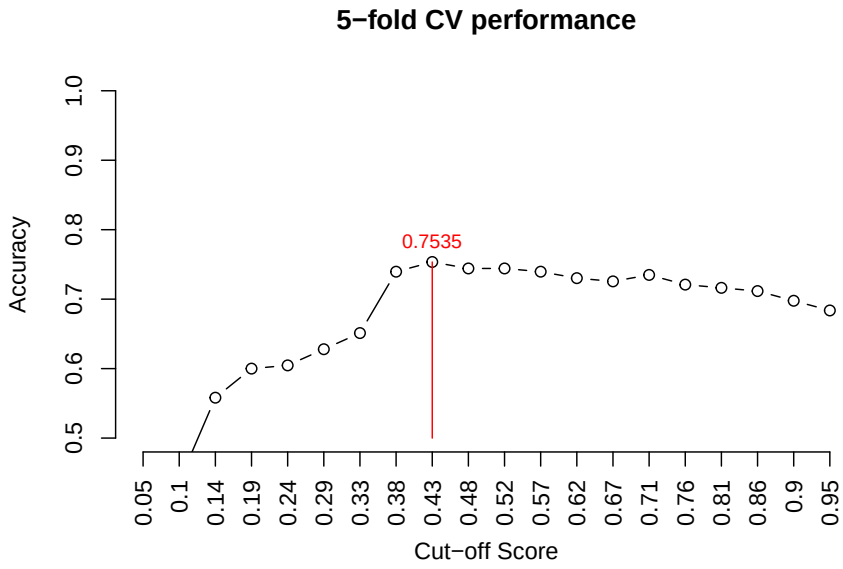


Figure 11.12: Single-layer backpropagation Neural network model for Pima Indian diabetes

```
     ylim=c(0.5,1),
     axes = FALSE,
     xlab = "Cut-off Score",
     ylab = "Accuracy",
     main = "5-fold CV performance"
     )
axis(1, at=1:20, label = tick.label, las = 2)
axis(2)
segments(max.id, 0.5, max.id, avg.accuracy[max.id], col = "red")
text(max.id, avg.accuracy[max.id]+0.03, as.character(round(avg.accuracy[max.id],4)), col = "red",
```



11.4.6.4 Testing Model Performance

```
#Test the resulting output
nn.results <- predict(NetworkModel, testDat)
results <- data.frame(actual = testDat[,17], prediction = nn.results > .57)
confMatrix = table(results$prediction, results$actual)                # confusion matrix
accuracy=sum(results$actua == results$prediction)/length(results$prediction)
list(confusion.matrix = confMatrix, accuracy = accuracy)
```

```
## $confusion.matrix
##
```

```
##           0   1
##   FALSE 276  94
##   TRUE   53  84
##
## $accuracy
## [1] 0.7100592
```

11.4.6.5 ROC Analysis

Recall that the ROC curve is the plot of sensitivity against (1 - specificity) calculated from the confusion matrix based on a sequence of selected cut-off scores. Definitions of sensitivity and specificity are given in the following confusion matrix

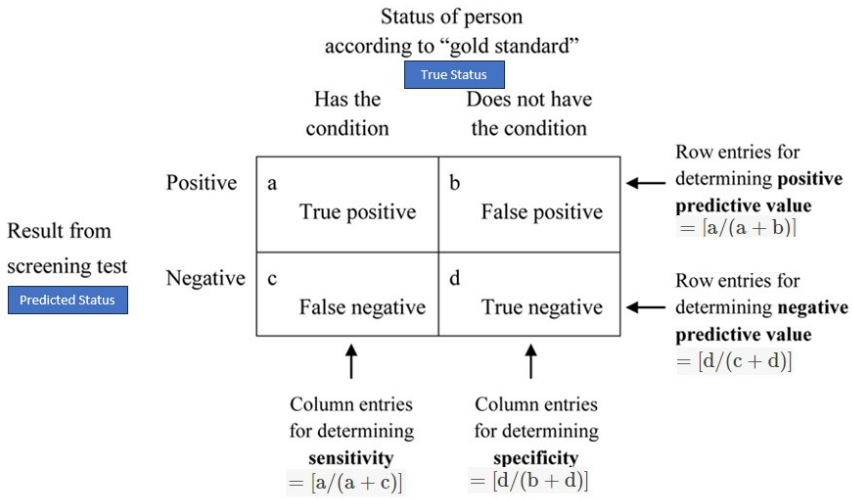


Figure 11.13: Confusion matrix and sensitivity and specificity.

Next, we construct a ROC for the above NN model based on the training data set.

```
nn.results = predict(NetworkModel, trainDat)  # Keep in mind that trainDat is a matrix
cut0 = seq(0,1, length = 20)
SenSpe = matrix(0, ncol = length(cut0), nrow = 2, byrow = FALSE)
for (i in 1:length(cut0)){
    a = sum(trainDat[,"diabetespos"] == 1 & (nn.results > cut0[i]))
    d = sum(trainDat[,"diabetespos"] == 0 & (nn.results < cut0[i]))
    b = sum(trainDat[,"diabetespos"] == 0 & (nn.results > cut0[i]))
    c = sum(trainDat[,"diabetespos"] == 1 & (nn.results < cut0[i]))
    sen = a/(a + c)
    spe = d/(b + d)
```

```
    SenSpe[,i] = c(sen, spe)
}
# plotting ROC
plot(1-SenSpe[2,], SenSpe[1,], type ="l", xlim=c(0,1), ylim=c(0,1),
     xlab = "1 - specificity", ylab = "Sensitivity", lty = 1,
     main = "ROC Curve", col = "blue")
abline(0,1, lty = 2, col = "red")

## A better approx of ROC, need library {pROC}
  prediction = as.vector(nn.results)
  category = trainDat[,"diabetespos"] == 1
  ROCobj <- roc(category, prediction)
```

```
## Setting levels: control = FALSE, case = TRUE
```

```
## Setting direction: controls < cases
```

```
  AUC = auc(ROCobj)[1]
  ##
###
text(0.8, 0.3, paste("AUC = ", round(AUC,4)), col = "purple", cex = 0.9)
legend("bottomright", c("ROC of the model", "Random guessing"), lty=c(1,2),
       col = c("blue", "red"), bty = "n", cex = 0.8)
```

The above ROC curve indicates that the underlying neural network is better than the random guess since the area under the curve is significantly greater than 0.5. In general, if the area under the ROC curve is greater than 0.65, we say the predictive power of the underlying model is acceptable.

11.4.7 About Deep Learning

From Wikipedia, the free encyclopedia

Deep learning is part of a broader family of machine learning methods, which is based on artificial neural networks with representation learning. The adjective "deep" in deep learning refers to the use of multiple layers in the network. Methods used can be either supervised, semi-supervised, or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks, convolutional neural networks, and transformers have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, climate science, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

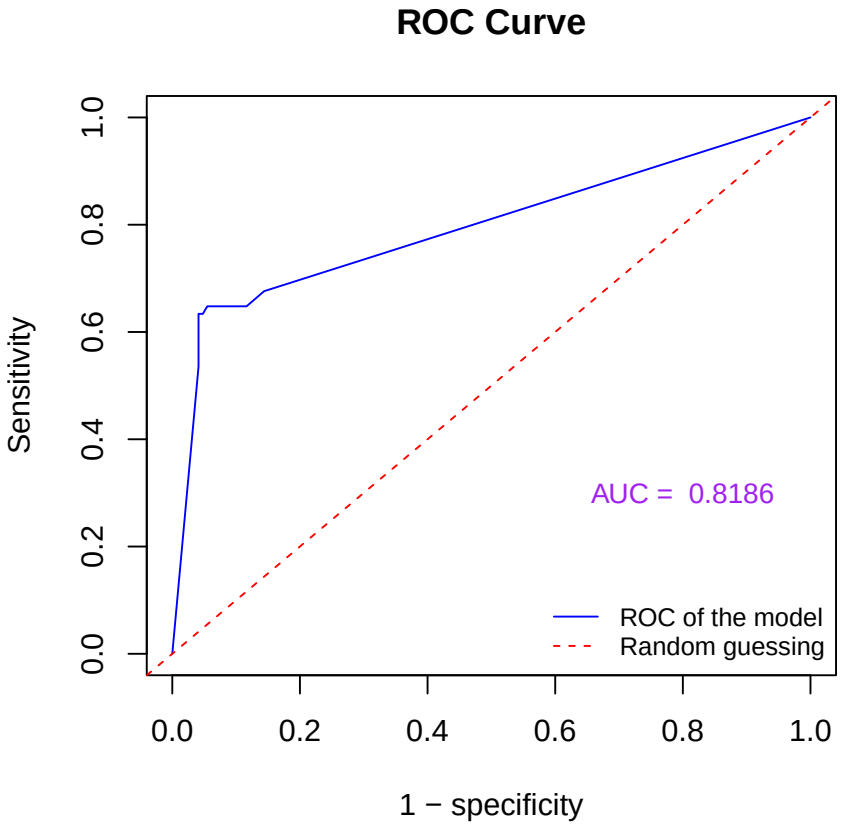


Figure 11.14: Figure 14: ROC Curve of the neural network model.

11.5 Clustering Algorithms

Clustering and principal component analysis are two classical multivariate statistical methods. These two methods and their extensions are now the core methods in machine learning. We will briefly introduce both of them in this class.

Chapter 12

An Overview of Supervised ML Algorithms

When crunching data to model business decisions, you are most typically using supervised and unsupervised learning methods.

Algorithms are often grouped by similarity in terms of their function (how they work). Again, there will be no perfect classification of machine learning algorithms and there is also no way to exhaust all machine learning algorithms. We only list those most commonly used algorithms here based on similarity to keep things simple.

For illustrative purposes, I will use some examples from the well-known **Introduction to statistical learning: with Applications in R** https://www.statlearning.com/.

This note will outline the supervised learning algorithms including statistical models and those developed by the machine learning community. We will build a decision tree model for the diabetes data set in the case study.

12.1 Statistical Algorithms

Regression is concerned with modeling the relationship between variables that is iteratively refined using a performance measure defined based on errors in the predictions made by the model. Regression methods are a workhorse of statistics and have been the backbone of statistical machine learning.

The most popular regression algorithms are:

- Ordinary Least Squares Regression (OLSR)
- Linear Regression
- Logistic Regression

- Step-wise Regression
- LOcally Estimated Scatter-plot Smoothing (LOESS) - a nonparametric regression.

12.1.1 LOESS Regression

The following visualization represents the LOESS regression based on two variables **Sales** and **Price** in the data set *Carseats* in the book **ISLR**.

```
data("Carseats")
kable(head(Carseats))
```

Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	U
9.50	138	73	11	276	120	Bad	42	17	Y
11.22	111	48	16	260	83	Good	65	10	Y
10.06	113	35	10	269	80	Medium	59	12	Y
7.40	117	100	4	466	97	Medium	55	14	Y
4.15	141	64	3	340	128	Bad	38	13	Y
10.81	124	113	13	501	72	Bad	78	16	N

```
lw1 = loess(Sales ~ Price, data = Carseats)
plot(Sales ~ Price, data = Carseats, pch=19, cex=0.8)
j = order(Carseats$Price)  # sort the data vector and returns the index
                           # of the values of the original data vector
lines(Carseats$Price[j],lw1$fitted[j],col="red",lwd=3)
```

The `loess()` is a data-driven nonparametric regression (local polynomial regression), in other words, the explicit model parameters in an explicitly expressed model to estimate. `loess` regression is analogous to single variable regression such as simple linear and nonlinear regression models.

To plot the `smooth` fitted regression curve, we need to use function `order()` the indices of the original data vector after it was ordered. For example,

```
x = c(3, 1, 0, 4, -5)
order(x)
```

```
## [1] 5 3 2 1 4
```

```
## The above index can sort the data below
x[order(x)]
```

```
## [1] -5  0  1  3  4
```

We can use the loess nonparametric regression model to predict as usual using the generic function `predict()` with an input data frame.

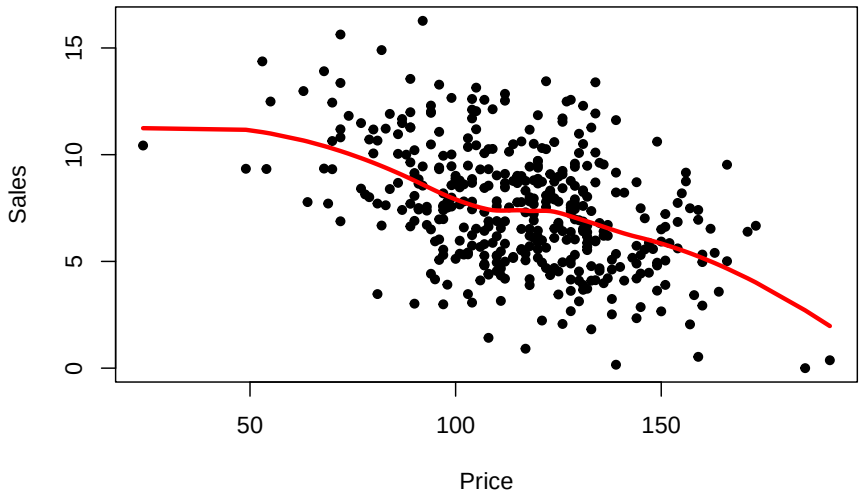


Figure 12.1: Figure 5. LOESS regression: Sales Prices

```
lw1 = loess(Sales ~ Price, data = Carseats)
predict(lw1, data.frame(Price =c(134,121)),se = TRUE)  # new data must be within
```

```
## $fit
## [1] 6.751464 7.371896
##
## $se.fit
## [1] 0.2261343 0.2216638
##
## $residual.scale
## [1] 2.518948
##
## $df
## [1] 393.7244
```

```
                                                    # Existing price range
```

12.1.2 Regularized Regression

We have discussed some degree of detail in linear and logistic regression models from both classical statistics perspective and machine learning perspective in terms of model training and performance evaluation.

The following **regularized regression** algorithms are recently developed learning algorithms modified from classical statistics.

- **Ridge Regression**

Ridge regression does not reduce the number of correlated feature variables. It brings bias to the estimated regression coefficient to reduce the impact of multi-correlated feature variables. In other words, it sacrifices the unbiasedness of the estimated regression to gain the stability of the estimation.

- **Least Absolute Shrinkage and Selection Operator (LASSO)**

LASSO filters the feature variables with a small magnitude of the absolute regression coefficients. All numerical feature variables must be standardized when using LASSO regression for prediction. Since some of the feature variables will be filtered out from the model. It is considered a dimension reduction method that becomes a popular tool in the machine learning community.

The following figure explains the relationships between regular least square regression, ridge regression, and LASSO.

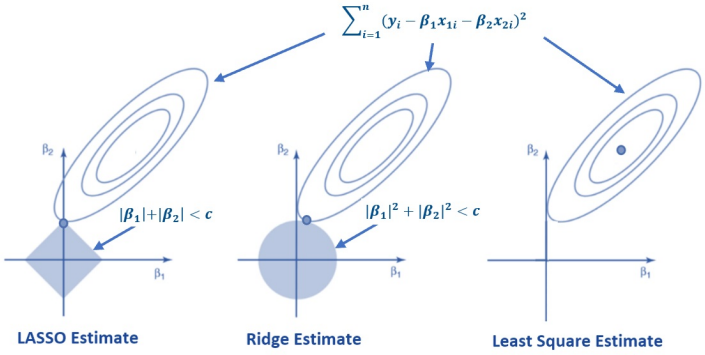


Figure 12.2: Relationship between least square, ridge, and LASSO regressions.

Because both **ridge** and **LASSO** fall into the same theoretical framework (although functioning in very different ways), Stanford statisticians developed an R library, `glmnet` to implement various regularized regression including both of these two regularized regression methods.

12.1.3 Instance-based Algorithms

First of all, observations/samples/instances all mean the same thing in machine learning.

The instance-based learning model is a decision problem with instances or examples of training data that are deemed important or required to the model. Such methods typically build up a database of example data and compare new data to the database using a similarity measure in order to find the best match and make a prediction. For this reason, instance-based methods are also called winner-take-all methods and memory-based learning (sometimes also called lazy learning. Focus is put on the representation of the stored instances and similarity measures used between instances.

The most popular instance-based algorithms are:

- k-Nearest Neighbor (kNN)
- Support Vector Machines (SVM)

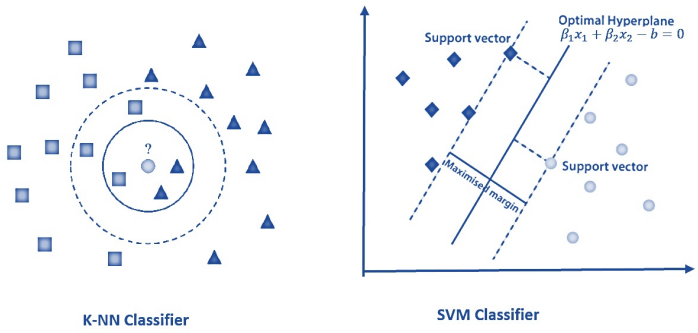


Figure 12.3: Instance-based learning algorithms: KNN and SVM.

Both kNN and SVM are intuitive. The more instances the more the accuracy. ISLR (2nd edition) has case studies using R for kNN (using **knn()** in library **{class}** in section 4.7.6, starting from page 181) and SVM (using **svm()** in library **{e1071}** in sections 9.6.1 and 9.6.2, starting from page 389).

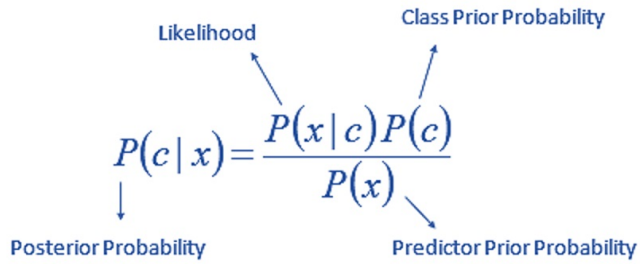
12.1.4 Naïve Bayes - A Bayesian Algorithm

Bayesian methods are those that explicitly apply Bayes' Theorem for problems such as classification and regression. There several Bayesian algorithms have been developed so far. We only introduce the basic but commonly used in practice - Naïve Bayes.

The Naïve Bayes classifier is a simple probabilistic classifier that is based on the Bayes theorem but with strong assumptions regarding independence. Historically, this technique became popular with applications in email filtering, spam detection, and document categorization. Although it is often outperformed by

other techniques, and despite the naïve design and oversimplified assumptions, this classifier can perform well in many complex real-world problems.

The theory behind Naïve Bayes is straightforward as depicted in the following.



$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 12.4: Naive Bayes Classifier.

There are several libraries in R that have the function to implement aïve Bayes. ISLR has a lab on the application of aïve Bayes (section 4.7.5, starting from page 180) using the **naiveBayes()** function in R library **{e1071}**.

12.2 Decision Tree Algorithms

https://github.com/pengdsci/STA551/blob/main/w07/img/w07.1-GIFtree.gif

The Decision Tree (DT) algorithm is based on conditional probabilities. Unlike the other classification algorithms, decision trees generate rules. A rule is a conditional statement that can easily be understood by humans and easily used within a database to identify a set of records. It is easy to interpret and implement in real-world applications. Among several basic tree-based algorithms, Classification and Regression Tree (CART) is most frequently used in practice.

This subsection focuses on the basic decision tree with some technical description of steps in decision tree induction. The general structure of a decision tree algorithm is in the following example of predicting the survival of Titanic passengers.

The above decision tree involves three variables: sex, age, and sibsp (sibling and spouse). We can easily convert the tree to a set of rules (conditional statements) to make a prediction of the survival status for a new incoming data point.

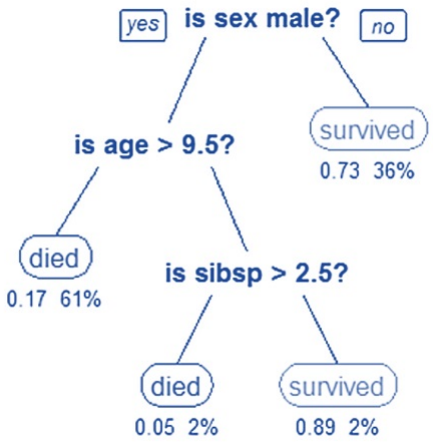


Figure 12.5: Illustration of decision tree algorithm: predicting Titanic survival.

12.2.1 Structure and Technical Terms

The following diagram illustrates the basic structure of a decision tree.

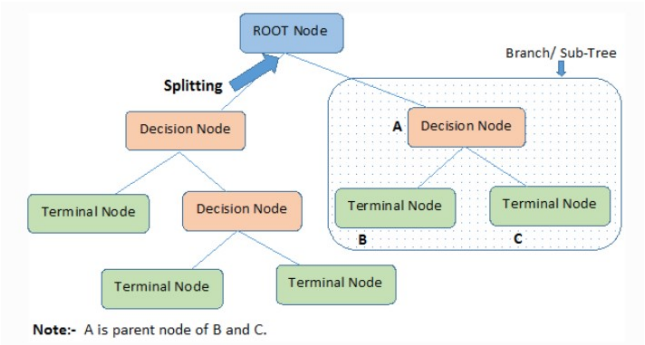


Figure 12.6: Decision tree structure.

Root Node: It represents the entire population or sample and this further gets divided into two or more homogeneous sets.

Splitting: It is a process of dividing a node into two or more sub-nodes.

Decision Node: When a sub-node splits into further sub-nodes, then it is called the decision node.

Leaf / Terminal Node: Nodes that do not split are called Leaf or Terminal Node.

Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.

Branch / Sub-Tree: A subsection of the entire tree is called a branch or sub-tree.

Parent and Child Node: A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

The following example based on toy data illustrates how a decision grows and how to use a decision tree to make predictions.

The toy data set is given below.

RID	Age	Income	Student	Credit rating	Class: buys computer
1	Youth	High	No	Fair	No
2	Youth	High	No	Excellent	No
3	Middle_aged	High	No	Fair	Yes
4	Senior	Medium	No	Fair	Yes
5	Senior	Low	Yes	Fair	Yes
6	Senior	Low	Yes	Excellent	No
7	Middle_aged	Low	Yes	Excellent	yes
8	Youth	Medium	No	Fair	No
9	Youth	Low	Yes	Fair	Yes
10	Senior	Medium	Yes	Fair	Yes
11	Youth	Medium	Yes	Excellent	Yes
12	Middle_aged	Medium	No	Excellent	Yes
13	Middle_aged	High	Yes	Fair	Yes
14	Senior	Mdium	No	Excellent	no

Figure 12.7: Decision tree structure using a toy data.

The fully grown tree is given below (note the variable **class** is the binary response variable).

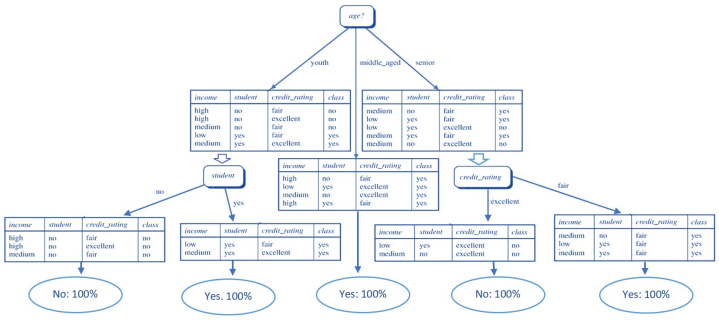


Figure 12.8: Fully grown decision tree using a toy data.

```
DataSet = data.frame(
Age = c("Youth", "Youth", "Middle_aged", "Senior", "Senior","Senior","Middle_aged","Youth","Youth
        "Middle_aged","Senior"),
Income = c("High","High","High","Medium","Low", "Low","Low","Medium","Low","Medium","Medium","Med
Student = c("No", "No","No","No","Yes","Yes","Yes","No","Yes","Yes","Yes","No","Yes","No"),
CreditRating = c("Fair", "Excellent","Fair","Fair","Fair","Excellent","Excellent",
                 "Fair","Fair","Fair","Excellent","Excellent","Fair","Excellent"),
Class = c("No", "No", "Yes", "Yes","Yes","No","Yes", "No", "Yes","Yes","Yes","Yes","Yes","No")
)
#pander(DataSet)
```

12.2.2 Decision Tree Growing - Impurity Measures

Growing a decision tree is an iterative process of splitting the feature space into some sub-spaces according to certain criteria defined based on feature variables. The predictive performance of a decision is dependent on the size of the trained tree. A small size will cause underfitting issues and a large size will result in overfitting issues.

The questions are (1) how to control the size of a decision to obtain the best performance; (2) how to select the feature variables to define the root and subsequent child nodes; (3) how to split a feature variable.

Gini index and **entropy** are the two popular impurity measures commonly used in decision tree induction.

12.2.2.1 Gini Index

- **Gini Index** considers a split for each attribute (for a continuous attribute, usually considers binary split). The Gini Index measures the impurity of subgroups (D) split by a feature variable.

$$\text{Gini}(D) = \sum_{i=1}^{m} p_i(1-p_i) = 1 - \sum_{i=1}^{m} p_i^2$$

Where p_i is the probability of an object that is being classified to a particular class.

For example, we calculate the Gini index using the above decision tree. The root node (age) has three child nodes. We show how to calculate the weighted Gini index of feature variable age in the following steps.

- **Weights**: P(youth) = 5/14, P(middle_aged) = 4/14, P(senior) = 5/14.

- **D = Youth**: $p_1 = P(Yes) = 2/5, p_2 = P(No) = 3/5$, therefore, $Gini_{youth} = 1 - p_1^2 - p_2^2 = 1 - 4/25 - 9/25 = 12/25$
- **D = Middle_aged**: $p_1 = P(Yes) = 4/4, p_2 = P(No) = 0/45$, therefore, $Gini_{middle_aged} = 1 - p_1^2 - p_2^2 = 1 - 16/16 - 0/16 = 0$
- **D = Senior**: $p_1 = P(Yes) = 3/5, p_2 = P(No) = 2/5$, therefore, $Gini_{senior} = 1 - p_1^2 - p_2^2 = 1 - 9/25 - 4/25 = 12/25$

The **Gini index** of age is given by

$$Gini_{age} = \frac{5}{14} \times \frac{12}{25} + \frac{4}{14} \times 0 + \frac{5}{14} \times \frac{12}{25} = \frac{5}{14} \times \frac{24}{25} = \frac{12}{35} \approx 0.343.$$

R Function for GINI Index

```
GINI.calc = function(DatName, VarName, ClsName){
   #
   freqTB0 = table(DatName[,VarName], DatName[,ClsName])
   freqTB = data.frame(NO = freqTB0[, 1], YES = freqTB0[, 2])
   freqTB$Tot = freqTB$NO + freqTB$YES
   freqTB$P1 = freqTB$NO/freqTB$Tot
   freqTB$P2 = freqTB$YES/freqTB$Tot
   freqTB$CateGINI = 1-(freqTB$P1)^2 - (freqTB$P2)^2
   freqTB$ROWPER = (freqTB$NO + freqTB$YES)/sum(freqTB$Tot)
   freqTB$ComponentGini = (freqTB$CateGINI) * (freqTB$ROWPER)
   GINI.idx = sum(freqTB$ComponentGini)
   GINI.idx
}
```

```
giniAge = GINI.calc(DatName=DataSet, VarName="Age", ClsName = "Class")
giniIncome = GINI.calc(DatName=DataSet, VarName="Income", ClsName = "Class")
giniStudent = GINI.calc(DatName=DataSet, VarName="Student", ClsName = "Class")
giniCreditRating = GINI.calc(DatName=DataSet, VarName="CreditRating",
                             ClsName = "Class")
pander(cbind(giniAge = giniAge, giniIncome = giniIncome,
             giniStudent = giniStudent, giniCreditRating = giniCreditRating))
```

giniAge	giniIncome	giniStudent	giniCreditRating
0.3429	0.4405	0.3673	0.4286

We can similarly calculate the Gini index for other feature variables in the data

set. **When we choose a feature variable to define the root node, we choose the feature with smallest Gini index**. The Gini index is used in the classic CART algorithm and is very easy to calculate.

12.2.2.2 Entropy and Information Gain

Entropy is another impurity measure that is defined by

$$E = \sum_{i=1}^{m} (-p_i \log_2 p_i).$$

Where p_i is the same as that defined in Gini index. We can find the entropy at the root node and each child node based on the above tree based on the toy data. A low Entropy indicates that the data labels are quite uniform.

- **root (parent) node entropy (before splitting)**: $E(\text{D}) = -(5/14)\log_2(5/14) - (9/14)\log_2(9/14) = 0.940286$
- **child node: youth**: $E(\text{D}) = -(2/5)\log_2(2/5) - (3/5)\log_2(3/5) = 0.9709506$
- **child node: middle_aged**: perfectly pure node had entropy 0.
- **child node: senior**: $E(\text{D}) = -(3/5)\log_3(2/5) - (2/5)\log_2(3/5) = 0.9709506$
- **Weighted average of entropy at child nodes**: $E(\text{child}) = \frac{5}{14} \times 0.9709506 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.9709506 = \frac{5}{14} \times 0.9709506 \approx 0.3467681$
- **Information Gain**: InfoGain $= E(\text{Parent Node}) - E(\text{Child Nodes}) = 0.940286 - 0.3467681 = 0.5935179$.

Information gain measures whether a further split is worthwhile.

```
infoGain.calc = function(DatName, VarName, ClsName){
   freqTB0 = table(DatName[,VarName], DatName[,ClsName])
   freqTB = data.frame(NO = freqTB0[, 1], YES = freqTB0[, 2])
   freqTB$Tot = freqTB$NO + freqTB$YES
   freqTB$P1 = freqTB$NO/freqTB$Tot
   freqTB$P2 = freqTB$YES/freqTB$Tot
   ###
   freqTB$ROWPER = (freqTB$NO + freqTB$YES)/sum(freqTB$Tot)
   ### Delete zero cell prob to calculate the entropy
   pNO = sum(freqTB$NO)/sum(freqTB$Tot)
   pYES = sum(freqTB$YES)/sum(freqTB$Tot)
```

```
   propYES = freqTB$YES/sum(freqTB$Tot)
   ParentEnt = -pNO*log2(pNO) -pYES *log2(pYES)
   ### entropy of child nodes
   P1 = freqTB$P1
   P2 = freqTB$P2
   logP1 = log2(freqTB$P1)
   logP2 = log2(freqTB$P2)
   logP1[which(!is.finite(logP1))] = 0
   logP2[which(!is.finite(logP2))] = 0
   ChildEnt = (-P1*logP1 - P2*logP2)
   ###
   infoGain =ParentEnt - sum(ChildEnt*propYES)
   #info.Gain = sum(freqTB$infoGain)
   list(ParentEnt = ParentEnt, ChildEnt = ChildEnt, propYES = propYES,
        infoGain = infoGain)
}
```

```
infoGain.calc(DatName=DataSet, VarName="Age", ClsName = "Class")
```

```
## $ParentEnt
## [1] 0.940286
##
## $ChildEnt
## [1] 0.0000000 0.9709506 0.9709506
##
## $propYES
## [1] 0.2857143 0.2142857 0.1428571
##
## $infoGain
## [1] 0.5935179
```

```
entAge = infoGain.calc(DatName=DataSet, VarName="Age",
                       ClsName = "Class")$infoGain
entIncome = infoGain.calc(DatName=DataSet, VarName="Income",
                          ClsName = "Class")$infoGain
entStudent = infoGain.calc(DatName=DataSet, VarName="Student",
                           ClsName = "Class")$infoGain
entCreditRating = infoGain.calc(DatName=DataSet, VarName="CreditRating",
                                ClsName = "Class")$infoGain
kable(cbind(infoGainAge = entAge, infoGainIncome = entIncome,
            infoGainStudent = entStudent,
            infoGainCreditRating = entCreditRating))
```

infoGainAge	infoGainIncome	infoGainStudent	infoGainCreditRating
0.5935179	0.3612133	0.4755916	0.3783096

12.2.3 Binary v.s. Multi-way Splits

In principle, trees are not restricted to binary splits but can also be grown with multi-way splits - based on the Gini index or other selection criteria. However, the (locally optimal) search for multi-way splits in numeric variables would become much more burdensome. Hence, tree algorithms often rely on the greedy forward selection of binary splits where subsequent binary splits in the same variable can also represent multi-way splits.

12.2.4 Boosted Trees - Ensemble Algorithms

Ensemble methods are models composed of multiple weaker models that are independently trained and whose predictions are combined in some way to make the overall prediction.

Much effort is put into what types of weak learners to combine and the ways in which to combine them. This is a very powerful class of techniques and as such is very popular.

12.2.4.1 Bootstrapped Aggregation (Bagging)

With the understanding of regular decision, we can

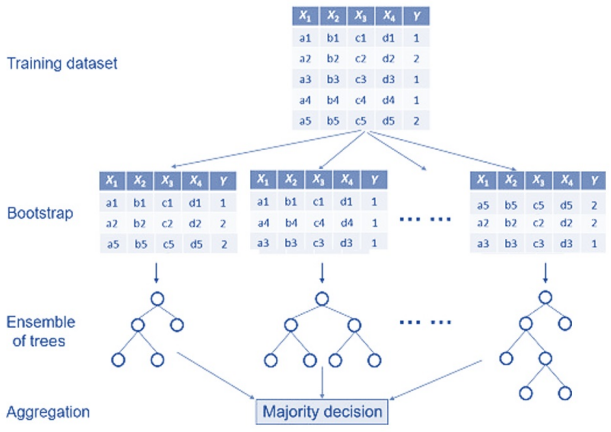


Figure 12.9: Demonstration of bootstrap aggregation algorithm.

12.2.4.2 Random Forest

Random forest (RF) algorithms make output predictions by combining outcomes from a sequence of decision trees. Each tree is constructed independently and depends on a random vector sampled from the input data, with all the trees in the forest having the same distribution. The predictions from the forests are averaged using bootstrap aggregation and random feature selection. RF models have been demonstrated to be robust predictors for both small sample sizes and high dimensional data

The following diagram illustrates how RF was constructed and how the decision is made based on the set of individual trees.

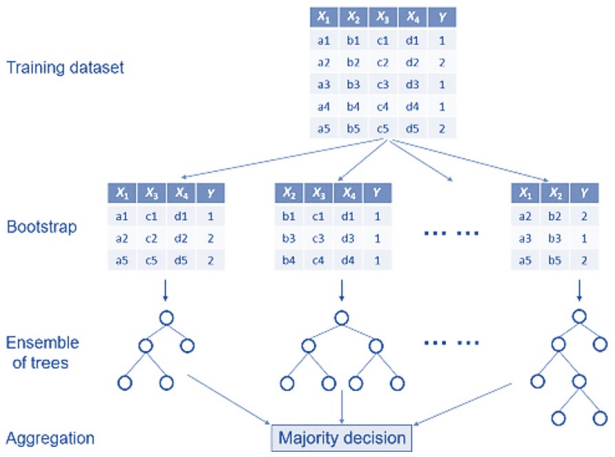


Figure 12.10: Demonstration of random forest algorithm.

12.3 Case Study - Predicting Diabetes

This is a new model that is different from logistic and neural network models. We load the analytic data set.

```
Pima = read.csv("https://pengdsci.github.io/STA551/w03/AnalyticPimaDiabetes.csv")[,-1]
# We use a random split approach
n = dim(Pima)[1]  # sample size
# caution: using without replacement
train.id = sample(1:n, round(0.7*n), replace = FALSE)
train = Pima[train.id, ]    # training data
test = Pima[-train.id, ]    # testing data
```

12.3.1 rpart Library

we will `rpart()` to write a wrapper so we can pass the arguments of purity measures and penalty measures to construct different decision trees. The cross-validation method will be used to select the optimal decision tree as the candidate predictive to compare with the logistic model in the previous section.

> Note that `rpart()` has a control option that allows users to set up various control parameters (so-called hyper-parameters) to allow the function to identify the optimal tree. One of those control parameters is the number of cross-validation when pruning the decision. Once `xval` is specified, `rpart()` prunes the tree automatically based on the given control parameters. This is internal k-fold cross-validation for identifying an optimal tree based on the information provided in the argument `parms` in which the purity measures and penalty matrix. More information can be found in the article https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

rpart() syntax

```
tree = rpart(modelFormula,          # model formula similar to that in the logistic models
             data ,
             na.action  = na.rpart,  # By default, deleted if the outcome is missing,
                                     # kept if features are missing
             method = "class",       # Classification form factor
             model  = FALSE,         # keep a copy of the model frame in the result? I
                  x = FALSE,         # keep a copy of the x matrix in the result.
                  y = TRUE,          # keep a copy of the dependent variable in the result.
                                     # If missing and model is supplied this defaults to FALSE
              parms = list( # loss matrix. Penalize false positive or negative more heavily
                         loss = matrix(c(0,b,c,0), ncol = 2),  # b = FP, c = FN
                         split = purity),   # "gini" or "information"

             ## rpart algorithm options (These are defaults)
             control = rpart.control(
                       minsplit = 20,       # minimum number of observations required before spli
                       minbucket= 10,       # minimum number of observations in any terminal node
                       cp  = 0.01,          # complexity parameter used as the stopping rule,  0.
                       maxcompete  = 4,     # number of competitor splits retained in the output
                       maxsurrogate   = 5,  # number of surrogate splits retained in the output
                       usesurrogate   = 2,  # how to use surrogates in the splitting process
                       xval   = 10,         # number of cross-validations
                       surrogatestyle = 0,  # controls the selection of the best surrogate
                       maxdepth  = 30       # maximum depth of any node of the final tree)
     )
```

`rpart()` has a lot of flexibility to construct decision trees as it has user controls. It is particularly useful in applications where the costs of `false positive` and `false negative` are different.

Next, we write a wrapper so we can build different decision trees conveniently.

```
# arguments to pass into rpart():
# 1. data set (training /testing);
# 2. Penalty coefficients
# 3. Impurity measure
##
tree.builder = function(in.data, fp, fn, purity){
   tree = rpart(diabetes ~ .,                # including all features
                data = in.data,
                na.action  = na.rpart,       # By default, deleted if the outcome is m
                                             # kept if predictors are missing
                method = "class",            # Classification form factor
                model  = FALSE,
                x = FALSE,
                y = TRUE,
              parms = list( # loss matrix. Penalize false positives or negatives more he
                           loss = matrix(c(0, fp, fn, 0), ncol = 2, byrow = TRUE),
                           split = purity),          # "gini" or "information"
               ## rpart algorithm options (These are defaults)
               control = rpart.control(
                          minsplit = 10,  # minimum number of observations required befo
                          minbucket= 10,  # minimum number of observations in any termin
                          cp  = 0.01,  # complexity parameter for stopping rule, 0.02 ->
                         xval = 10     # number of cross-validation )
                          )
               )
  }
```

Using the above function, we define six different decision tree models in the following.

- Model 1: `gini.tree.11` is based on the Gini index without penalizing false positives and false negatives.
- Model 2: `info.tree.11` is based on entropy without penalizing false positives and false negatives.
- Model 3: `gini.tree.110` is based on the Gini index: cost of false negatives is 10 times the positives.
- Model 4: `info.tree.110` is based on entropy: cost of false negatives is 10

times the positives.

- Model 5: `gini.tree.101` is based on the Gini index: cost of false positive is 10 times the negatives.
- Model 6: `info.tree.101` is based on entropy: cost of false positive is 10 times the negatives.

The tree diagram of the above two regular decision models is given below.

```
## Call the tree model wrapper.
gini.tree.1.1 = tree.builder(in.data = train, fp = 1, fn = 1, purity = "gini")
info.tree.1.1 = tree.builder(in.data = train, fp = 1, fn = 1, purity = "information")
gini.tree.1.10 = tree.builder(in.data = train, fp = 1, fn = 10, purity = "gini")
info.tree.1.10 = tree.builder(in.data = train, fp = 1, fn = 10, purity = "information")
## tree plots
par(mfrow=c(1,2))
rpart.plot(gini.tree.1.1, main = "Tree with Gini index: non-penalization")
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```

```
rpart.plot(info.tree.1.1, main = "Tree with entropy: non-penalization")
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```

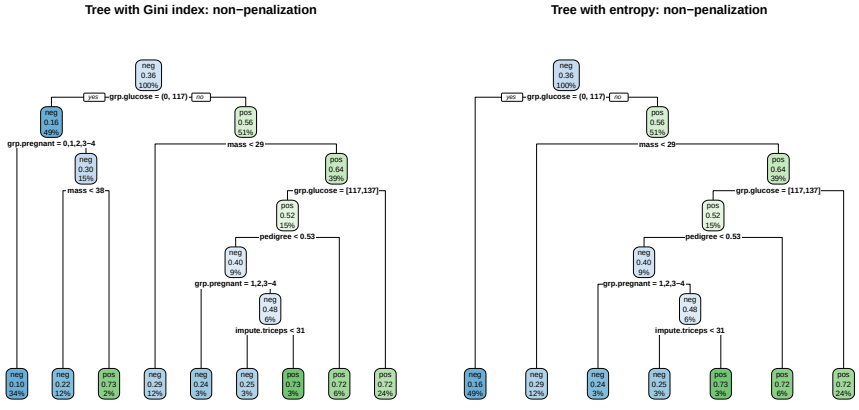


Figure 12.11: Non-penalized decision tree models using Gini index (left) and entropy (right).

```
par(mfrow=c(1,2))
rpart.plot(gini.tree.1.10, main = "Tree with Gini index: penalization")
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine round
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```

```
rpart.plot(info.tree.1.10, main = "Tree with entropy: penalization")
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine round
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```

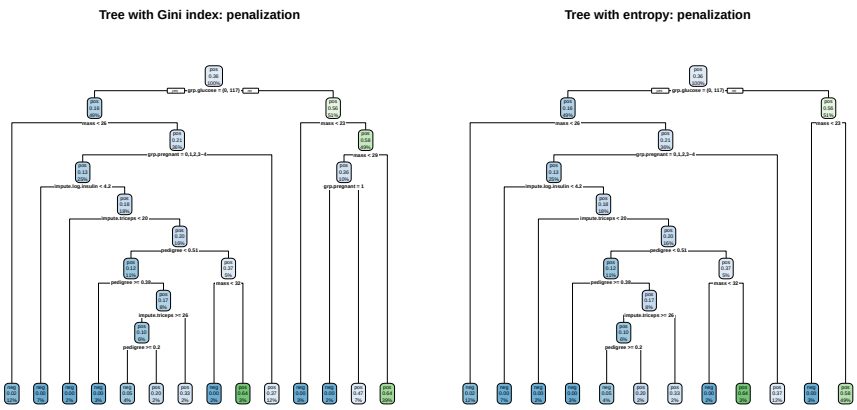


Figure 12.12: penalized decision tree models using Gini index (left) and entropy (right).

12.3.2 ROC for Model Selection

We built 4 different decision tree models previously. Next, we use ROC analysis to select the best among the four candidate models.

```
# function returning a sensitivity and specificity matrix
SenSpe = function(in.data, fp, fn, purity){
  cutoff = seq(0,1, length = 20)   # 20 cut-offs including 0 and 1.
  model = tree.builder(in.data, fp, fn, purity)
  ## Caution: decision tree returns both "success" and "failure" probabilities.
  ## We need only "success" probability to define sensitivity and specificity!!!
  pred = predict(model, newdata = in.data, type = "prob") # two-column matrix.
  senspe.mtx = matrix(0, ncol = length(cutoff), nrow= 2, byrow = FALSE)
```

```
  for (i in 1:length(cutoff)){
  # CAUTION: "pos" and "neg" are values of the label in this data set!
  # The following line uses only "pos" probability: pred[, "pos"] !!!!
  pred.out =  ifelse(pred[,"pos"] >= cutoff[i], "pos", "neg")
  TP = sum(pred.out =="pos" & in.data$diabetes == "pos")
  TN = sum(pred.out =="neg" & in.data$diabetes == "neg")
  FP = sum(pred.out =="pos" & in.data$diabetes == "neg")
  FN = sum(pred.out =="neg" & in.data$diabetes == "pos")
  senspe.mtx[1,i] = TP/(TP + FN)
  senspe.mtx[2,i] = TN/(TN + FP)
  }
  ## A better approx of ROC, need library {pROC}
  prediction = pred[, "pos"]
  category = in.data$diabetes == "pos"
  ROCobj <- roc(category, prediction)
  AUC = auc(ROCobj)
  ##
  list(senspe.mtx= senspe.mtx, AUC = round(AUC,3))
 }
```

The above function has three arguments for users to choose different types of decision trees including the 4 trees discussed in the previous subsection. Next we use this function to build 6 different trees and plot their corresponding ROC curves so we can see the global performance of these tree algorithms.

```
giniROC11 = SenSpe(in.data = train, fp=1, fn=1, purity="gini")
```

```
## Setting levels: control = FALSE, case = TRUE
```

```
## Setting direction: controls < cases
```

```
infoROC11 = SenSpe(in.data = train, fp=1, fn=1, purity="information")
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

```
giniROC110 = SenSpe(in.data = train, fp=1, fn=10, purity="gini")
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

```
infoROC110 = SenSpe(in.data = train, fp=1, fn=10, purity="information")
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

```
giniROC101 = SenSpe(in.data = train, fp=10, fn=1, purity="gini")
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

```
infoROC101 = SenSpe(in.data = train, fp=10, fn=1, purity="information")

## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```

Next, we plot the ROC curves and calculate the areas under the ROC curves for Individual decision tree models.

```
par(pty="s")      # set up square plot through graphic parameter
plot(1-giniROC11$senspe.mtx[2,], giniROC11$senspe.mtx[1,], type = "l", xlim=c(0,1), yli
     xlab="1 - specificity: FPR", ylab="Sensitivity: TPR", col = "blue", lwd = 2,
     main="ROC Curves of Decision Trees", cex.main = 0.9, col.main = "navy")
abline(0,1, lty = 2, col = "orchid4", lwd = 2)
lines(1-infoROC11$senspe.mtx[2,], infoROC11$senspe.mtx[1,], col = "firebrick2", lwd = 1
lines(1-giniROC110$senspe.mtx[2,], giniROC110$senspe.mtx[1,], col = "olivedrab", lwd =
lines(1-infoROC110$senspe.mtx[2,], infoROC110$senspe.mtx[1,], col = "skyblue", lwd = 2)
lines(1-giniROC101$senspe.mtx[2,], giniROC101$senspe.mtx[1,], col = "red", lwd = 2, lty
lines(1-infoROC101$senspe.mtx[2,], infoROC101$senspe.mtx[1,], col = "sienna3", lwd = 2)
legend("bottomright", c(paste("gini.1.1,  AUC =", giniROC11$AUC),
                        paste("info.1.1,   AUC =",infoROC11$AUC),
                        paste("gini.1.10, AUC =",giniROC110$AUC),
                        paste("info.1.10, AUC =",infoROC110$AUC),
                        paste("gini.10.1, AUC =",giniROC101$AUC),
                        paste("info.10.1, AUC =",infoROC101$AUC)),
                        col=c("blue","firebrick2","olivedrab","skyblue","red","sienna3"
                        lty=rep(1,6), lwd=rep(2,6), cex = 0.5, bty = "n")
```

The above ROC curves represent various decision trees and their corresponding AUC. The model with the largest AUC is considered the best decision tree among the existing ones.

12.3.3 Optimal Cut-off Score Determination

As usual, once the final model is determined, we need to find the optimal cut-off score for reporting the predictive performance of the final model with the test data. Please keep in mind the optimal cut-off determination through cross-validation must be based on the training data set.

In practical applications, one may end up with two or more **final models** with similar AUCs. In this case, we need to report the performance of all *final models* based on the test data and let clients choose one to deploy (and possibly leave the rest as challengers). For this reason, we write a function to determine the optimal cut-off for a given decision tree (based on this project) since different decision trees have their own optimal cut-off.

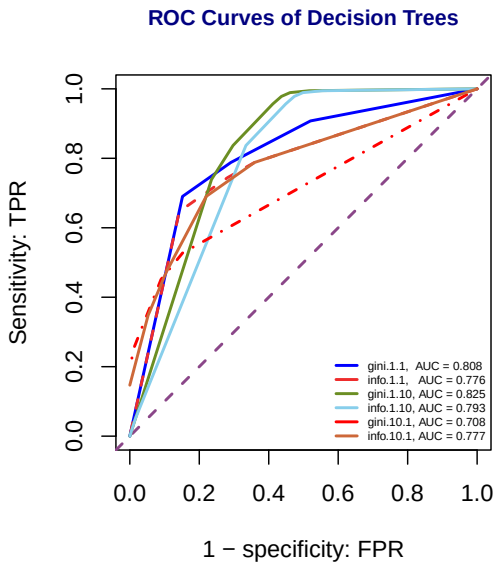


Figure 12.13: Comparison of ROC curves

```
Optm.cutoff = function(in.data, fp, fn, purity){
  n0 = dim(in.data)[1]/5
  cutoff = seq(0,1, length = 20)                # candidate cut off prob
  ## accuracy for each candidate cut-off
  accuracy.mtx = matrix(0, ncol=20, nrow=5)     # 20 candidate cutoffs and gini.11
  ##
  for (k in 1:5){
     valid.id = ((k-1)*n0 + 1):(k*n0)
     valid.dat = in.data[valid.id,]
     train.dat = in.data[-valid.id,]
     ## tree model
     tree.model = tree.builder(in.data, fp, fn, purity)
     ## prediction
     pred = predict(tree.model, newdata = valid.dat, type = "prob")[,2]
     ## for-loop
     for (i in 1:20){
        ## predicted probabilities
        pc.1 = ifelse(pred > cutoff[i], "pos", "neg")
        ## accuracy
        a1 = mean(pc.1 == valid.dat$diabetes)
        accuracy.mtx[k,i] = a1
```

```
      }
    }
  avg.acc = apply(accuracy.mtx, 2, mean)
  ## plots
  n = length(avg.acc)
  idx = which(avg.acc == max(avg.acc))
  tick.label = as.character(round(cutoff,2))
  ##
  plot(1:n, avg.acc, xlab="cut-off score", ylab="average accuracy",
       ylim=c(min(avg.acc), 1),
       axes = FALSE,
       main=paste("5-fold CV optimal cut-off \n ",purity,"(fp, fn) = (", fp, ",", fn,'
       cex.main = 0.9,
       col.main = "navy")
       axis(1, at=1:20, label = tick.label, las = 2)
       axis(2)
       points(idx, avg.acc[idx], pch=19, col = "red")
       segments(idx , min(avg.acc), idx , avg.acc[idx ], col = "red")
      text(idx, avg.acc[idx]+0.03, as.character(round(avg.acc[idx],4)), col = "red", c
  }
```

For demonstration, we use the above function to calculate the optimal cut-off of 6 decision trees constructed earlier in the following.

```
par(mfrow=c(3,2))
Optm.cutoff(in.data = train, fp=1, fn=1, purity="gini")
Optm.cutoff(in.data = train, fp=1, fn=1, purity="information")
Optm.cutoff(in.data = train, fp=1, fn=10, purity="gini")
Optm.cutoff(in.data = train, fp=1, fn=10, purity="information")
Optm.cutoff(in.data = train, fp=10, fn=1, purity="gini")
Optm.cutoff(in.data = train, fp=10, fn=1, purity="information")
```

As anticipated, different trees have their own optimal cut-off. Please keep in mind that the cut-off is random (based on the randomly split training data), there may have different cut-offs in different runs. It is dependent on the tree size, sometimes, we may end up with multiple optimal cut-offs. Technically speaking, we choose any one of them for implementation. A better recommendation is to choose the average of these multiple cut-offs and the final cut-off to be used on the testing data set.

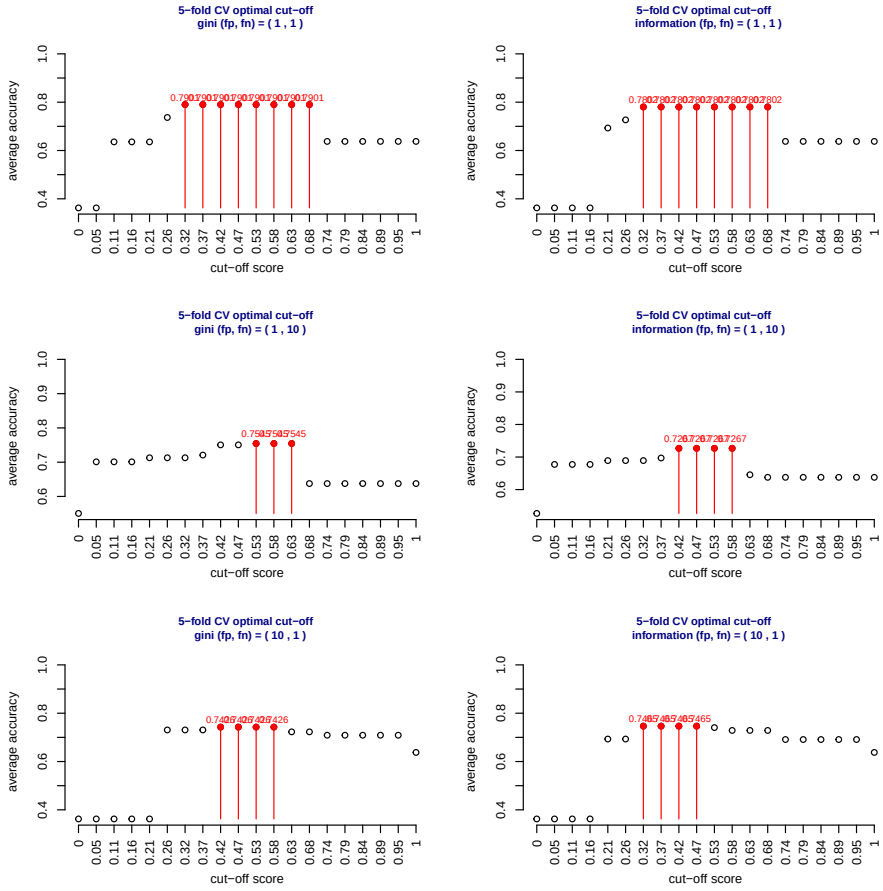


Figure 12.14: Plot of optimal cut-off determination

Chapter 13

An Overview of Unsupervised ML Algorithms

This note overviews the basic unsupervised machine learning algorithms (also known as knowledge discovery) in which models are not supervised using a training data set. Instead, models themselves find the **hidden patterns** and insights from the given data.

The goal of unsupervised learning is to find the underlying structure of the data set and group that data according to similarities. Common algorithms used in unsupervised learning include clustering, anomaly detection, neural networks, and approaches for learning latent variable models.

The following types of unsupervised machine learning algorithms are commonly used in practice.

- K-means Clustering
- Hierarchical Clustering
- Anomaly Detection
- Principal Component Analysis

There are many methods to calculate this distance information; the choice of distance measures is a critical step in clustering. It defines how the similarity of two data points (x, y) is calculated and it will influence the shape of the clusters. The choice of distance measures is a critical step in clustering. It defines how the similarity of two data points (x, y) is calculated and it will influence the shape of the clusters.

https://github.com/pengdsci/STA551/blob/main/w08/img/w08-kMeans-gif.gif

Here are a few sites you check for these "distances".

- https://elki-project.github.io/algorithms/distances
- https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6853338
- https://cran.r-project.org/web/packages/SimilarityMeasures/SimilarityMeasures.pdf

In the next few sections, we will describe each of these algorithms.

13.1 K-means Clustering

K-means algorithm is an iterative algorithm that partitions the data set into K *pre-defined*, distinct, and non-overlapping subgroups (clusters) where each data point belongs to only one group. Data points within each subgroup are **similar** while data points across the subgroups are "different** according to a selected dissimilarity measure used in the algorithm.

In other words, k-means clustering consists of defining clusters so that the total intra-cluster variation (known as a total within-cluster variation) is minimized. There are several k-means algorithms available. The standard algorithm is to define the total within-cluster variation as the sum of squared (SS) distances (Euclidean distances) between data points and the corresponding centroid. To be more specific, let x_i be the data point in cluster k, denoted by C_k and μ_k is the center of cluster C_k (i.e. the mean value of the points when Euclidean distance is used). The within-cluster SS is defined by

$$W(C_k) = \sum_{x_i \in C_k} (x_i - \mu_i)^2$$

> Each observation (x_i) is assigned to a given cluster such that the sum of squares (SS) distance of the observation to their assigned cluster centers μ_k is a minimum.

We define the total within-cluster variation as follows

$$\text{TW} = \sum_{k=1}^{k} W(C_l) = \sum_{i=1}^{k} \sum_{x_k \in C_k} (x_i - \mu_k)^2$$

> The total within-cluster sum of square measures the compactness (i.e. goodness) of the clustering and we want it to be as small as possible.

Two fundamental questions to answer are: (1) how many initial clusters should be selected; (2) how to choose the initial "centers".

K-means clustering algorithm works in the following three steps.

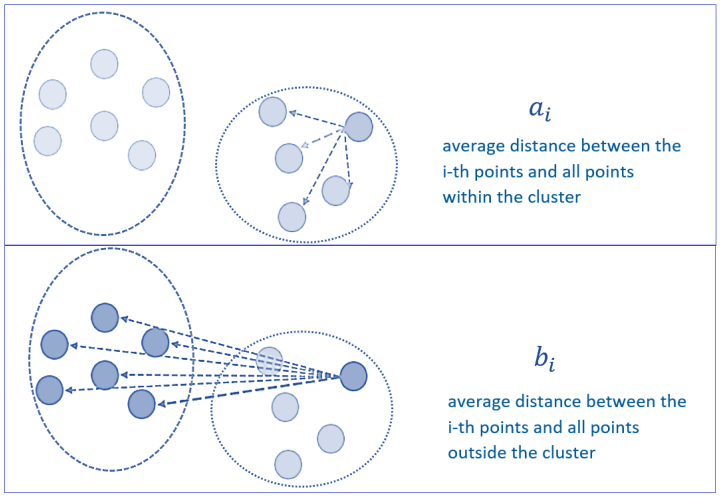


Figure 13.1: Illustration of cluster analysis.

13.1.1 Select A Pre-determined Number of Classes

Several algorithms can be used to find the optimal number of clusters. Elbow and Silhouette algorithms are commonly used and are implemented in R.

Elbow Method

The Elbow method gives us an idea of what a good k number of clusters would be based on the sum of squared distance (SSE) between data points and their assigned clusters' centroids. We pick k at the spot where SSE starts to flatten out and form an elbow. We'll use the geyser dataset and evaluate SSE for different values of k and see where the curve might form an elbow and flatten out.

Elbow is one of the most famous methods for selecting the right value of k. We also perform the hyper-parameter tuning to choose the best value of k. The Elbow method is an empirical method to find out the best value of k.

Silhouette Method

Silhouette Method uses a similarity measure (Silhouette coefficient) that is defined in the following

$$S_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$$

where S_i is the silhouette coefficient of the data point i; a_i is the average distance between i and all the other data points in the cluster to which i belongs, and b_i is the average distance from i to all clusters to which i does not belong.

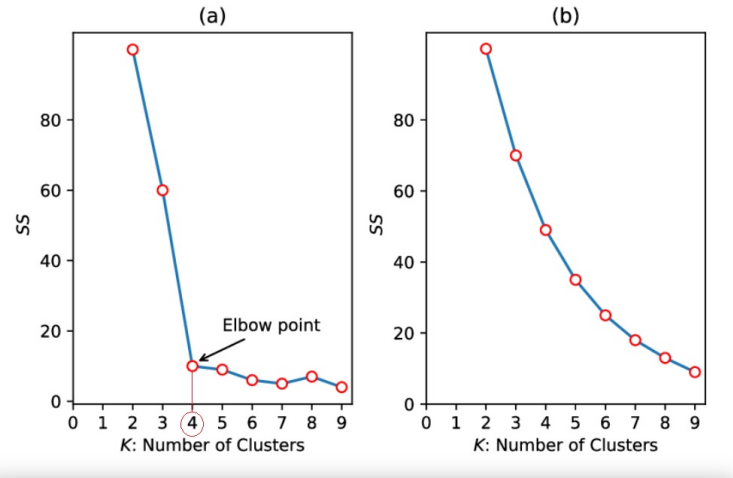


Figure 13.2: Figure 3. (a) A visual curve with an explicit elbow point. (b) A visual curve being fairly smooth with an ambiguous elbow point.

We can plot the Silhouette coefficient against the pre-determined clusters k. The plot of the silhouette is between -1 to 1.

A high average silhouette width indicates good clustering. The average silhouette method computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximizes the average silhouette over a range of possible values for k.

Observe the plot and choose the k value that is closer to 1 as the optimal number of clusters.

13.1.2 Initialize Centroids.

Initialize centroids by first shuffling the data set and then randomly selecting K data points for the centroids without replacement.

13.1.3 Update Centroids

Updating centroids is an iterative process:

1. Compute the sum of the squared distance between data points and all (initial) centroids. Assign each data point to the closest cluster (centroid).
2. Compute the (updating) centroids for the clusters by taking the average of all data points that belong to each cluster.

Keep iterating until there is no change to the centroids. i.e., the assignment of data points to clusters isn't changing.

The following figure illustrates how to find the updated centroids immediately after the initial centroids.

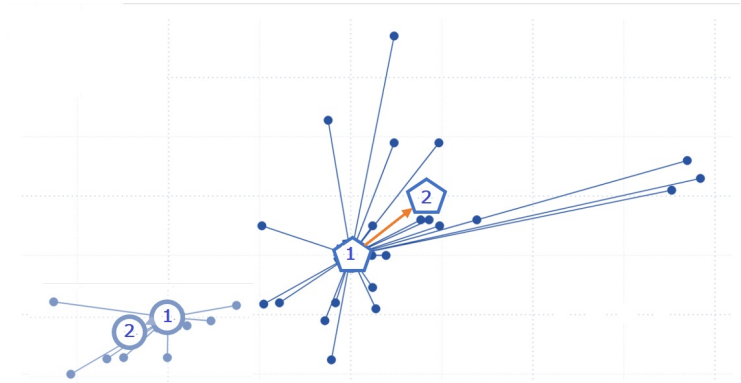


Figure 13.3: Updating centroids in the process of finding the final centroids

13.1.4 Some Remarks on K-means

1. K-means clustering assumes numerical features since the Euclidean distance is used to define similarity measures.

2. K-Means clustering performs well only for a convex set of clusters and not for non-convex sets.

3. Recent development allows categorical feature variables with non-Euclidean distance.

4. The k-means algorithm does not guarantee finding the optimal solution. k-means is a fairly simple sequence of tasks and its clustering quality depends a lot on two factors: the number of k clusters and initial centroids.

13.1.5 Case Study

For the illustrative purpose, we only use two numerical variables in a simple data set that is publically available in Github https://raw.githubusercontent.com/satishgunjal/datasets/master/Mall_Customers.csv.

```
df = read.csv("https://raw.githubusercontent.com/satishgunjal/datasets/master/Mall_Customers.csv'
## rename the two variables and then subset the data
names(df)[names(df)=="Annual.Income..k.."] = "AnnualIncome"
names(df)[names(df)=="Spending.Score..1.100."] = "SpendingScore"
clust.data = df[, c("AnnualIncome", "SpendingScore")]
scaled.data = as.data.frame(scale(clust.data)[,1:2])
```

```
distance = get_dist(scaled.data)
fviz_dist(distance, gradient = list(low = "yellow", mid = "orange", high = "darkred"),
```

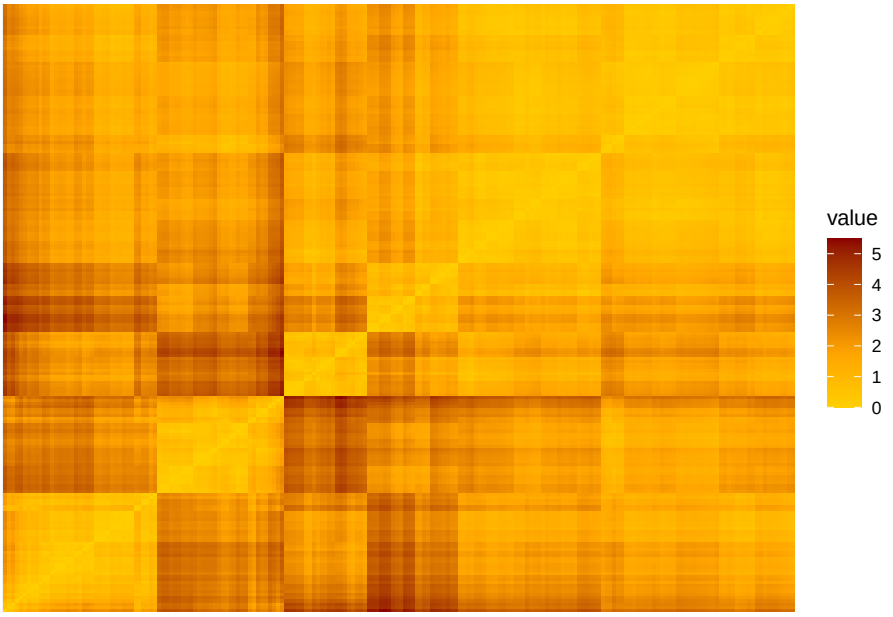


Figure 13.4: Heatmap representation of potential clusters

The above heatmap indicates that different clusters exist in this data (based on the two numerical variables).

- The syntax of **kmeans()** is given in the following code chunk.

```
k2 <- kmeans(x = scaled.data,
             centers = 2,
             iter.max = 10,
             nstart = 25,
             algorithm = "Hartigan-Wong",
             trace = FALSE)
```

- Determination of optimal class.

We use the elbow method to find the optimal number of clusters.

```
wss = NULL
K = 15
for (i in 1:K){
  wss[i] = kmeans(scaled.data, i, 1 )$tot.withinss
 }
```

```
## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1

## Warning: 1
```

```
## elbow plot
plot(1:K, wss, type ="b",
          col= "blue",
          xlab="Number of Clusters",
          ylab = "WSS",
          main = "Elbow Plot for Selecting Optimal Number of Clusters")
```

From the above elbow plot, it seems that the optimal number of clusters is 5. So select k - 5 hereafter.

- Cluster the data with 5 centroids

We will cluster the data into 5 groups and then add the cluster ID to the data. Since only two continuous feature variables were used to cluster the data. After we added the cluster ID to the data, we use color coding to make a scatter plot and view the clusters.

```
k5 <- kmeans(x = scaled.data,
             centers = 5,
             iter.max = 10,
             nstart = 25,
             algorithm = "Hartigan-Wong",
```

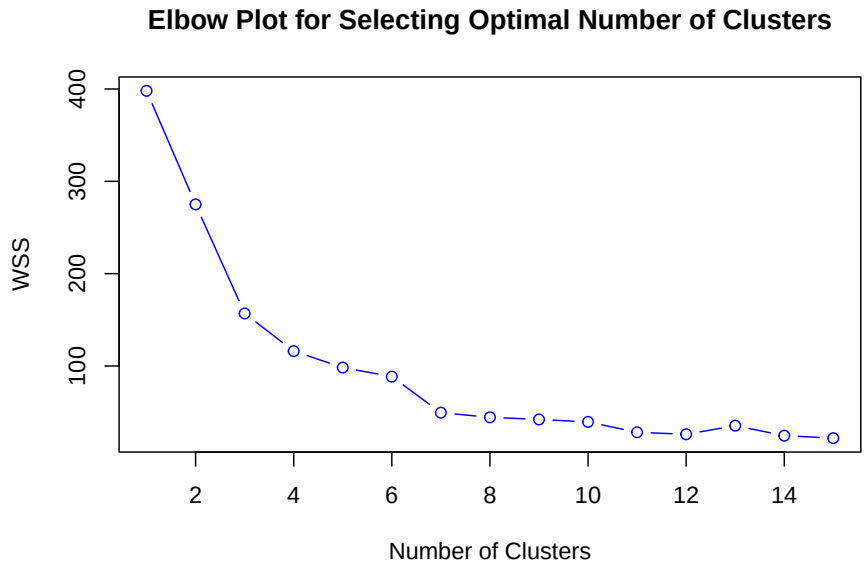


Figure 13.5: Elbow plot for optimal number of clusters.

```
              trace = FALSE)
scaled.data$group = k5$cluster
### Plot the clusters
# Scatter plot
plot(scaled.data$AnnualIncome, scaled.data$SpendingScore,
     pch = 19,
     col = factor(scaled.data$group),
     xlab ="Spending Score",
     ylab = "Annual Income",
     main = "Clustering Performance Visual Check")

# Legend
legend("topright",
       legend = levels(factor(scaled.data$group)),
       pch = 19,
       col = factor(levels(factor(scaled.data$group))))
```

13.2 Hierarchical Clustering

provided a solid introduction to one of the most popular clustering methods. Hierarchical clustering is an alternative approach to k-means clustering for iden-

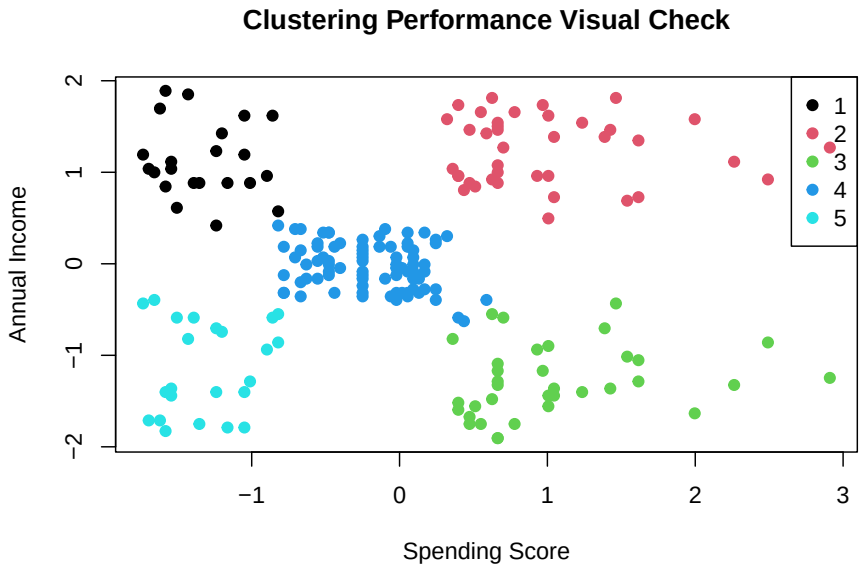


Figure 13.6: Final cluster results: visual inspection

tifying groups in the dataset. It does not require us to pre-specify the number of clusters to be generated as is required by the k-means approach. Furthermore, hierarchical clustering has an added advantage over K-means clustering in that it results in an attractive tree-based representation of the observations, called a dendrogram.

13.2.1 Types of Hierarchical Clustering

There are two types of hierarchical clustering: agglomerative and divisive.

- **Agglomerative***: An agglomerative approach begins with each observation in a distinct (singleton) cluster, and successively merges clusters until a stopping criterion is satisfied.
- **Divisive**: A divisive method begins with all patterns in a single cluster and performs splitting until a stopping criterion is met.

As an example, we look at how agglomerative clustering works using five data points in the following figure.

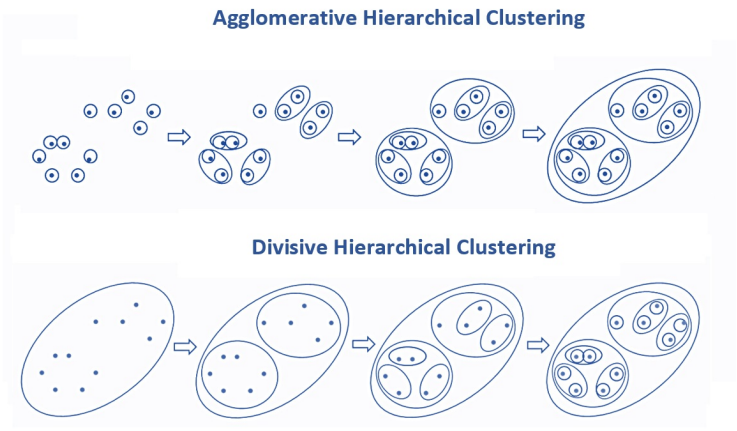


Figure 13.7: Illustration of types of hierarchical clustering.

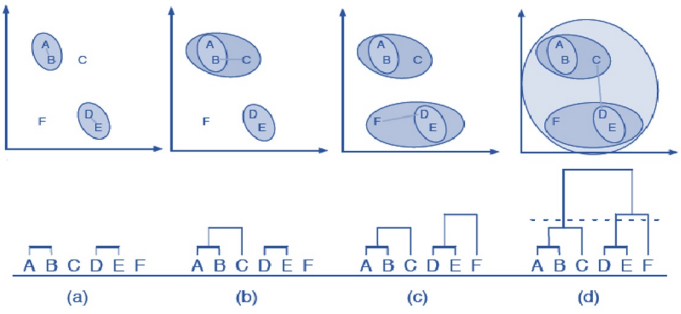


Figure 13.8: Illustration of steps of agglomerative hierarchical clustering.

13.2.2 Case Study I – Clustering with Two Features

We still use the same data set that we used in the previous case study of K-means clustering but will include **age** variable in the data frame for following hierarchical clustering.

```
df = read.csv("https://raw.githubusercontent.com/satishgunjal/datasets/master/Mall_Customers.csv"
## Rename the two variables and then subset the data
names(df)[names(df)=="Annual.Income..k.."] = "AnnualIncome"
names(df)[names(df)=="Spending.Score..1.100."] = "SpendingScore"
hierarch.data = df[, c("Age", "AnnualIncome", "SpendingScore")]
```

13.2.2.1 Pre-processing Operations for Clustering

There are a couple of things you should take care of before starting.

Scaling is imperative that we normalize the scale of feature values in order to start with the clustering process. This is because each observation's feature values are represented as coordinates in n-dimensional space (n is the number of features) and then the distances between these coordinates are calculated. If these coordinates are not normalized, then it may lead to false results. R has functions **scale()** and **normalize()**.

Missing Value imputation is also important to deal with missing/null/inf values in your data set beforehand. There are many ways to deal with such values, one is to either remove them or impute them with mean, median, mode, or use some advanced regression techniques. R has many packages and functions to deal with missing value imputations like **impute()**.

13.2.2.2 Hierarchical Clustering with R

There are different functions available in R for computing hierarchical clustering. The commonly used functions are:

- **hclust()** [in stats package] and **agnes()** [in cluster package] for agglomerative hierarchical clustering (HC).
- **diana()** [in cluster package] for divisive HC.

```
scales.hierarch = as.data.frame(hierarch.data)
distance <- dist(scales.hierarch, method = "euclidean")
# Hierarchical clustering using Complete Linkage
hc1 <- hclust(distance, method = "complete" )
# Plot the obtained dendrogram
plot(hc1, cex = 0.6, labels = FALSE, hang = -1, xlab = "",
     main = "Dendrogram: hierarchical clustering")
rect.hclust(hc1, k = 5, border = 2:9)
```

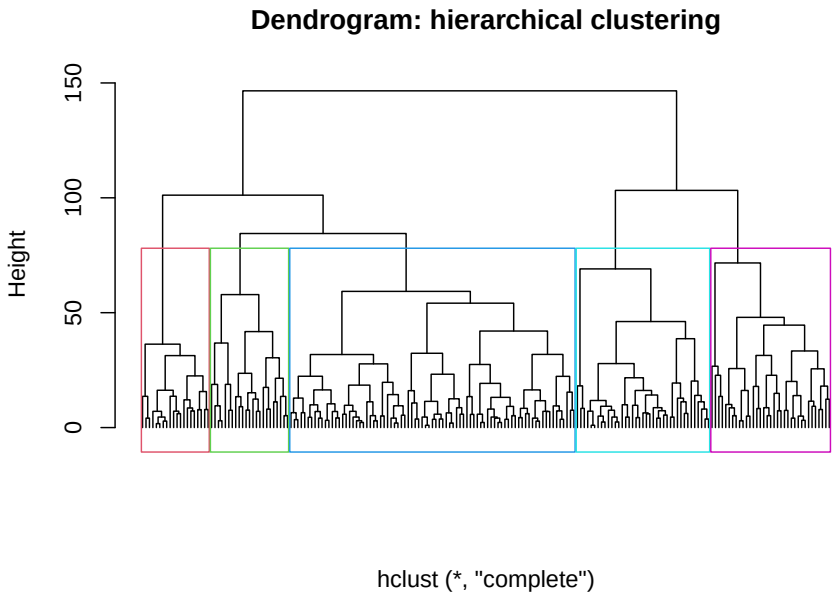


Figure 13.9: Dendrogram: hierarchical clustering

```
# Figured this out by coloring the labels white to the background
avg_dend_obj <- as.dendrogram(hc1)
labels_colors(avg_dend_obj) <- "white"
plot(avg_dend_obj, cex = 0.6,
     labels = FALSE,
     hang = -1,
     xlab = "",
     ylab= "Height",
     main = "Dendrogram: hierarchical clustering: No X-Labels")
```

```
## Warning in plot.window(...): "labels"
## Warning in plot.window(...): "hang"
## Warning in plot.xy(xy, type, ...): "labels"
## Warning in plot.xy(xy, type, ...): "hang"
## Warning in axis(side = side, at = at, labels = labels, ...): "hang"
## Warning in axis(side = side, at = at, labels = labels, ...): "hang"
## Warning in title(...): "labels"
## Warning in title(...): "hang"
```

```
rect.hclust(hc1, k = 5, border = 2:9)
```

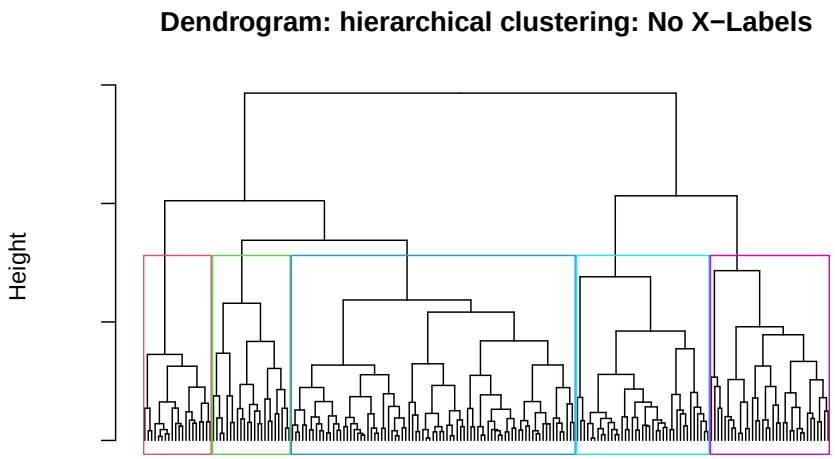


Figure 13.10: Dendrogram: hierarchical clustering: No X-Labels

13.2.2.3 Determination Optimal Number of Clusters

The determination of the optimal number of clusters is an important and challenging problem. In hierarchical clustering, different similarity measures impact the number of optimal clusters. We will not discuss this topic in detail. To know more about this topic, you are referred to the following article with examples in R https://www.jstatsoft.org/article/view/2194/798.

We can use the same elbow and silhouette methods to plot

```
fviz_nbclust(scales.hierarch, FUN = hcut, method = "wss")
```

```
fviz_nbclust(scales.hierarch, FUN = hcut, method = "silhouette")
```

13.2.2.4 Extracting Cluster ID

The above-elbow plot indicates that choosing 4 clusters is appropriate. Next, we perform a 4-cluster analysis and extract the cluster ID to add them to the data frame.

```
hc4 <- hclust(distance, method = "complete" )
group = cutree(hc4, k = 5)
```

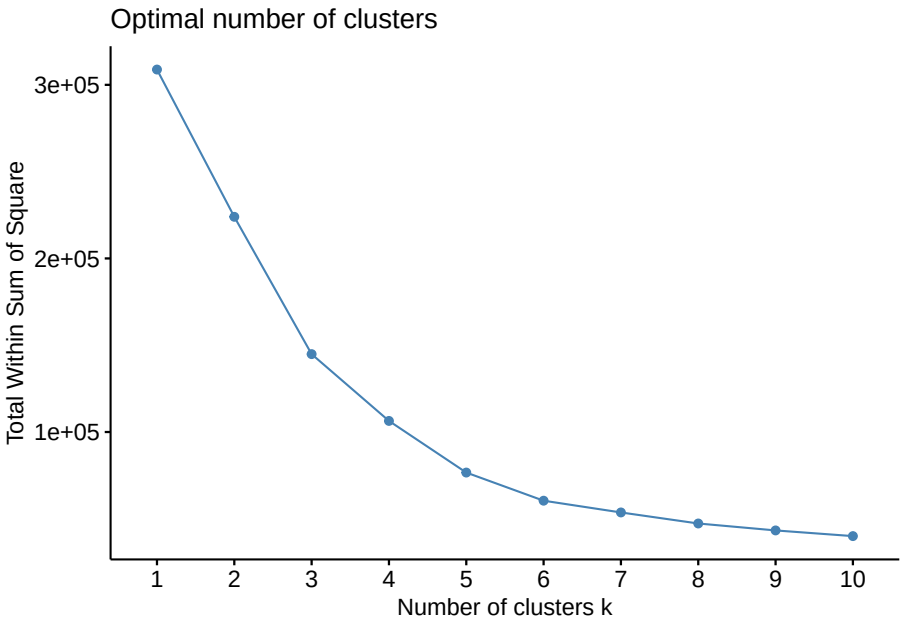


Figure 13.11: Elbow plot: Optimal number of clusters

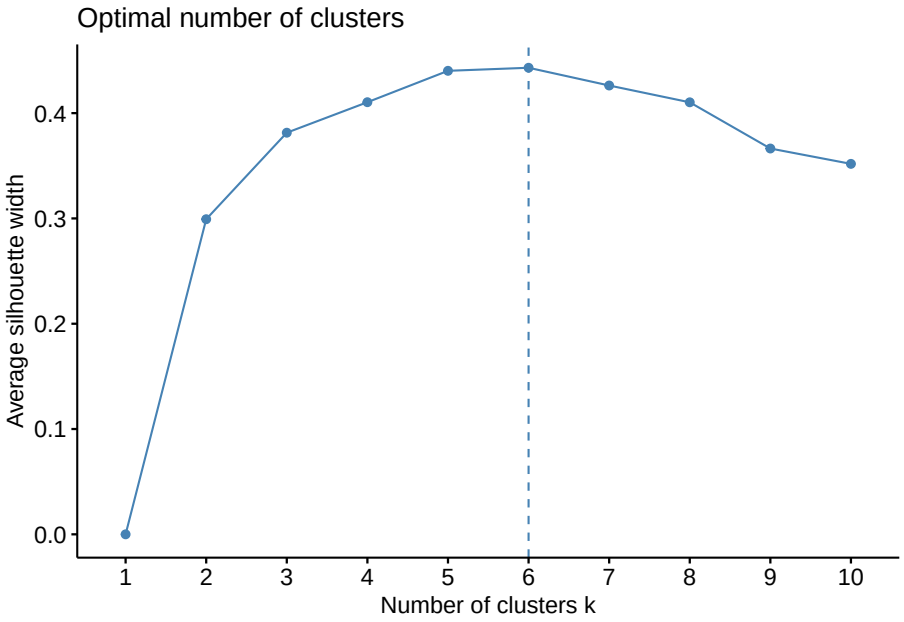


Figure 13.12: Silhouette plot: Optimal number of clusters

```
scales.hierarch$group = group
##
plot(scales.hierarch$AnnualIncome, scales.hierarch$SpendingScore,
     pch = 19,
     col = factor(scales.hierarch$group),
     xlab ="Spending Score",
     ylab = "Annual Income",
     main = "Hierarchical Clustering Performance Visual Check")

## Legend
legend("topright",
       legend = levels(factor(scales.hierarch$group)),
       pch = 19,
       col = factor(levels(factor(scales.hierarch$group))))
```

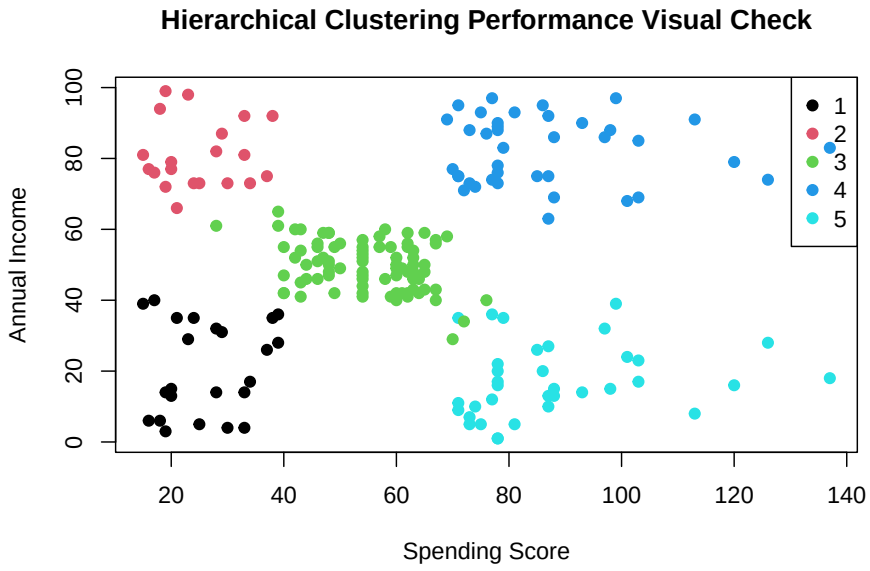


Figure 13.13: Visual check the resulting clusters obtained from agglomerative hierarchical clustering.

13.3 Case Study II: Multi-feature Clustering

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica, and Iris versicolor). These measures were used to create a linear discriminant model to classify the species. The dataset is often used in data mining, classification, and clustering examples and to test algorithms.

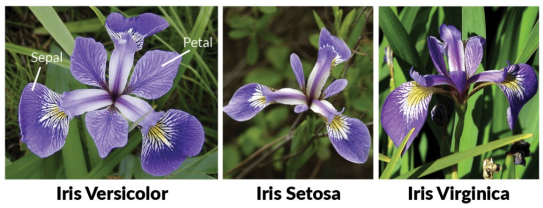


Figure 13.14: Iris data set: variables illustration - pedal and sepal

This 100-year-old data set has been included in the R base package. The first few records of the data set are displayed in the following table.

```
pander(head(iris))
```

SepalLength	SepalWidth	PetalLength	PetalWidth	Classification
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa

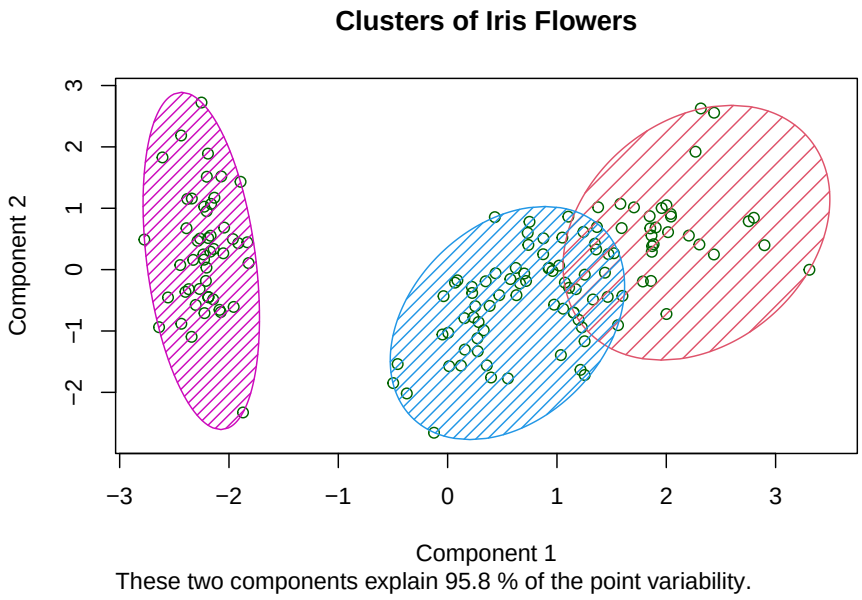
We use **k-means** method to perform cluster analysis on the **iris data** with the four numerical feature variables.

```
myClusteredIris = iris
# We start with 3 clusters since we know there are 3 species in the data.
# In practice, we need relatively try different number of clusters and then
# use the elbow plot to determine the best number of clusters.
km.iris <- kmeans( x = myClusteredIris[, -5] , centers = 3)
clust.ID <- km.iris$cluster        # extracting cluster IDs

table(clust.ID)                    # frequency of clusters
```

```
## clust.ID
##  1  2  3
## 50 62 38
```

Since this clustering task involves 4 numerical feature variables, we cannot create a 2D plot to show the clustering performance with all four original feature variables. However, we can so-called PCA (to be discussed in the next section) to create two new feature variables and then plot the new features to show the performance of the clustering algorithm.

```
clusplot(iris[, -5],
 clust.ID,
 lines = 0,
 shade = TRUE,
 color = TRUE,
 labels = 1,
 plotchar = FALSE,
 span = TRUE,
 main = paste("Clusters of Iris Flowers")
)
```



13.3.1 Memory Usage of Clustering

One of the potential issues in clustering analysis is the use of memory. If the data size is too large,

13.4 Dimensionality Reduction Algorithms

Like clustering methods, dimension reduction seeks and explores the inherent structure in the data, but in this case in an unsupervised manner or to summarize or describe data using less information.

This can be useful to visualize high-dimensional data or to simplify data which can then be used in a supervised learning method. Many of these methods can be adapted for use in classification and regression. The following are the frequently used algorithms.

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- `Independent Component Analysis (ICA)`

In this section, we introduce the most commonly used PCA.

13.4.1 The Logic of PCA

We use a two-variable animation as an example to illustrate the idea of principal component analysis (PCA).

https://github.com/pengdsci/STA551/blob/main/w08/img/w08-PCA-Animation-01.gif

The above animated graph shows that two or more numerical feature variables are highly correlated, the PCA can be used to aggregate the information in the correlated feature variables by transforming them to a set of uncorrelated **new feature variables** such that the majority of the total information is captured by the first few new feature variables.

13.4.2 Case Study - Iris Data

We have used the well-known Iris Data set in clustering algorithms. The data set has 4 correlated numerical variables(sepal width and length, petal width and length) and a categorical variable. The four variables measure the size of the flowers. We use PCA to see whether reducing the number of feature variables for related modeling.

13.4.2.1 Fitting PCA model to Iris data

We want to PCA method to reduce the dimensions from 4 (numerical variables) to a smaller number. The R function **prcomp()** to the factor loadings associated with the four numerical variables.

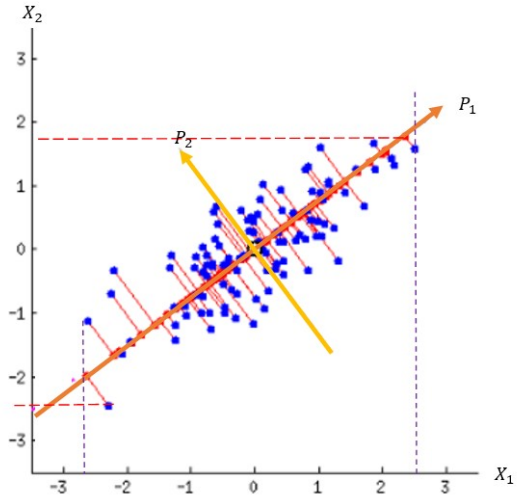


Figure 13.15: Graphical interpretation of PCA with two correlated variables

Table 13.2: Factor loadings of the PCA

	PC1	PC2	PC3	PC4
SepalLength	0.51	-0.45	0.71	0.21
SepalWidth	-0.30	-0.89	-0.32	-0.10
PetalLength	0.58	-0.03	-0.20	-0.79
PetalWidth	0.57	-0.04	-0.59	0.57

```
log.iris = log(iris[,-5])    # drop the categorical variable in the original
                             # data set and transform all numerical to the
                             # log-scale
ir.pca <- prcomp(log.iris, center = TRUE, scale = TRUE)
# summary(ir.pca)[6]    # use the command to explore the possible information
                        # available in the output of the summary.
```

In the above R function, three arguments are explained in the following

```
log.iris = log of the four variables
cater = TRUE, this means the variables are centered, i.e.,  you move the origin of the original c
scale = TRUE, divide the difference between the value of each variable and its mean by the standa
```

Next, we find the factor loading of the above fitted PCA. We can write an explicit system of linear transformation by using the loadings.

```
kable(round(ir.pca$rotation, 2), caption="Factor loadings of the PCA")
```

The explicit expression of the predictive system of PC is given by

Table 13.3: The importance of each principal component

	PC1	PC2	PC3	PC4
Standard deviation	1.708715	0.9563817	0.3700768	0.1693209
Proportion of Variance	0.729930	0.2286700	0.0342400	0.0071700
Cumulative Proportion	0.729930	0.9585900	0.9928300	1.0000000

$$PC_1 = 0.50Sepal.Length - 0.30Sepal.Width + 0.58Petal.Length + 0.57Petal.Width$$
$$PC_2 = -0.45Sepal.Length - 0.89Sepal.Width - 0.03Petal.Length - 0.04Petal.Width$$
$$PC_3 = 0.71Sepal.Length - 0.33Sepal.Width - 0.22Petal.Length - 0.58Petal.Width$$
$$PC_4 = 0.19Sepal.Length - 0.09Sepal.Width - 0.79Petal.Length + 0.58Petal.Width$$

The magnitude of factor loadings indicates the amount of information that original variables contribute to the corresponding principal components. For example, the absolute value of loadings associated with petal width and length and sepal length in PC_1 is greater than or equal to 0.5. We can simply call PC_1 the **size** of iris flowers. Similarly, sepal length and width are major contributors to PC_2, we can name PC_2 as **sepal size**.

13.4.2.2 Optimal number of PCs to be retained

The object of PCA is to reduce the dimension without losing a significant amount of information. In PCA, we look at how much total variation is captured by each principal component. Most of the libraries that are capable of performing PCA automatically rank the PCA based on the variation captured by each principal component.

The following summary table gives the importance of the principal components.

```
kable(summary(ir.pca)$importance, caption="The importance of each principal component")
```

From the above table, we can see that the first PC explains about 73.33% of the variation. But we first two principal components explain about 96% of the total variation. In the data analysis, you only need to use the first two PCs that lose about 4% of the information.

We can also make a scree plot as a visual tool to show the number of principal components to retain for future analysis.

```
screeplot(ir.pca,
          type = "lines",
          main = "Scree Plot of PCA Iris Flower Sizes")
```

Note that the vertical axis in the above scree plot uses the variances of PCs. The standard deviation was used in the above summary table.

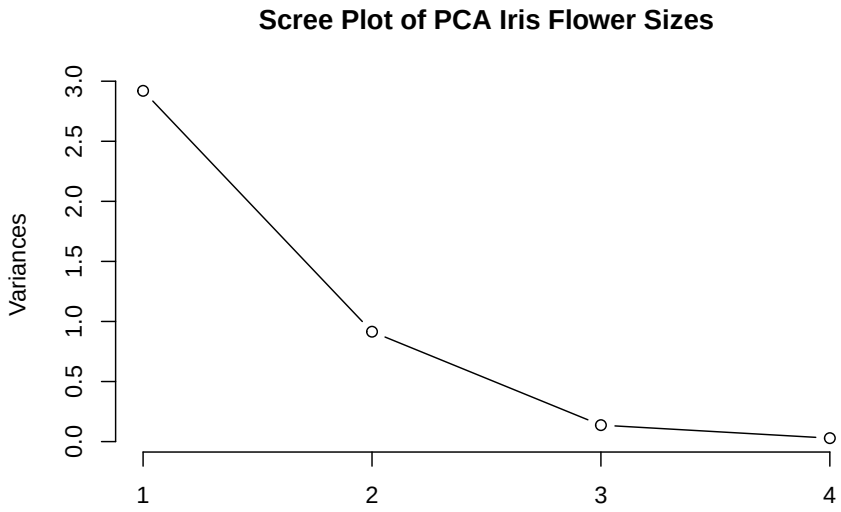


Figure 13.16: Scree plot of PCA on Iris Data

13.4.2.3 Extracting PC Scores

The predictive principle scores are values of the newly transformed variables. We can choose the first few principal components to use as response variables to do relevant modeling.

The command `ir.pcs$x` extracts the PC scores from the PCA procedure. These scores are the values of the new transformed variables. They can be used as response or predictor variables in statistical models. The following table shows the

```
kable(ir.pca$x[1:15,], caption = "The first 15 PC scores transformed from the original variable.
```

As the final step, we rename the two PCs and then add the two new variables to the original data set for future analysis. Since PC_1 captures variation of both sepal and pedal, we rename PC_1 as **iris.size**. Similarly, we rename PC_2 as **sepal.size**.

```
my.final.iris.data = iris
my.final.iris.data$iris.size = ir.pca$x[, 1]
my.final.iris.data$sepal.size = ir.pca$x[, 2]
## write the final data set to a local folder
#write.csv(my.final.iris.data, file = "C:\\Users\\75CPENG\\OneDrive - West Chester University of
```

Table 13.4: The first 15 PC scores transformed from the original variable. In the analysis, you want to either the first PC or the first two PCs.

PC1	PC2	PC3	PC4
-2.399323	-0.4230132	0.1808988	0.0148983
-2.222208	0.6710008	0.3295067	0.0675933
-2.578374	0.4014949	-0.0005677	0.0597146
-2.449092	0.6619725	-0.0853303	-0.1406184
-2.528843	-0.5370943	0.0176988	-0.0342582
-1.832157	-1.3192660	-0.2595561	0.1570885
-2.476608	0.0698647	-0.5143622	0.1176323
-2.342356	-0.1817091	0.1237008	-0.0855882
-2.537388	1.2245319	-0.1337017	-0.0648975
-2.619625	0.4888347	0.6474674	-0.4456031
-2.242377	-0.9560140	0.3182291	-0.0342892
-2.425334	-0.0555452	-0.1031337	-0.2316215
-2.693003	0.7631906	0.6418480	-0.3595538
-3.323662	1.1230276	0.1721670	-0.1966056
-2.367240	-1.6608246	0.5762004	0.3128922

The following scree shot shows the final data file was saved in a local folder and the two renamed principal components were added to the final data set.

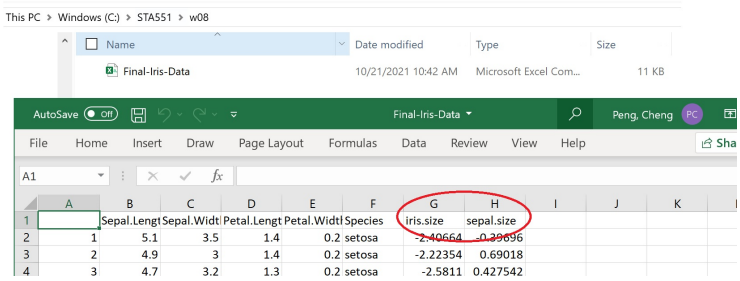


Figure 13.17: Screenshot of the final iris data set with new variables defined based on the principal components

Chapter 14

Bootstrap Algorithms and Applicatications

The bootstrap method is a data-based simulation method for statistical inference. The method assumes that

- The sample is a random sample representing the population;
- The sample size is large enough such that the empirical distribution can be close to the true distribution.

14.1 Basic Idea of Bootstrap Method.

The objective is to estimate a population parameter such as mean, variance, correlation coefficient, regression coefficients, etc. from a random sample without assuming any probability distribution of the underlying distribution of the population.

For convenience, we assume that the population of interest has a cumulative distribution function $F(x:\theta)$, where θ is a vector of the population. For example, You can think about the following distributions

- **Normal distribution**: $N(\mu, \sigma^2)$, the distribution function is given by

$$f(x:\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

  where $\theta = (\mu, \sigma)$. Since the normal distribution is so fundamental in statistics, we use the special notation for the cumulative distribution $\phi_{\mu,\sigma^2}(x)$ or simply $\phi(x)$. The corresponding probability function

- **Binomial distribution**: $Binom(n, p)$, the probability distribution is given by

$$P(x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}, x = 0, 1, 2, \cdots, n-1, n.$$

where $\theta = p$. *Caution*: n is NOT a parameter!

We have already learned how to make inferences about population means and variances under various assumptions in elementary statistics. In this note, we introduce a **new approach** to making inferences only based on a given random sample taken from the underlying population.

As an example, we focus on the population mean. For other parameters, we can follow the same idea to make bootstrap inferences.

14.1.1 Sampling True Population - Monte Carlo Sampling

We have introduced various study designs and sampling plans to obtain random samples from a given population with the distribution function $F(x : \theta)$. Let μ be the population mean.

- **Random Sample**. Let

$$\{x_1, x_2, \cdots, x_n\} \to F(x : \theta)$$

be a random sample from population $F(x : \theta)$.

- **Sample Mean**. The point estimate is given by

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n}$$

- **Sampling Distribution of $\hat{\mu}$**. In order to construct the confidence interval of μ or make hypothesis testing about μ, we need to know the sampling distribution of $\hat{\mu}$. From elementary statistics, we have the following results.
  - $\hat{\mu}$ is normally distributed if (1). n is large; or (2). the population is normal and population variance is known.
  - the standardized $\hat{\mu}$ follows a t-distribution if the population is normal and population variance is unknown.
  - $\hat{\mu}$ is **unknown** of the population is not normal and the sample size is not large enough.
- In the last case of the previous bullet point, we don't have the theory to derive the sampling distribution based on a **single** sample. However, if the sampling is not too expensive and time-consuming, we take following the sample study design and sampling plan to repeatedly take a large

number, 1000, samples of the same size from the population. We calculate the mean of each of the 1000 samples and obtain 1000 sample means $\{\widehat{\mu}_1, \widehat{\mu}_2, \cdots, \widehat{\mu}_{1000}\}$. Then the empirical distribution of $\widehat{\mu}$.

The following figure depicts the process of how to sample the true population and approximate the sampling distribution of the point estimator of the population parameter.

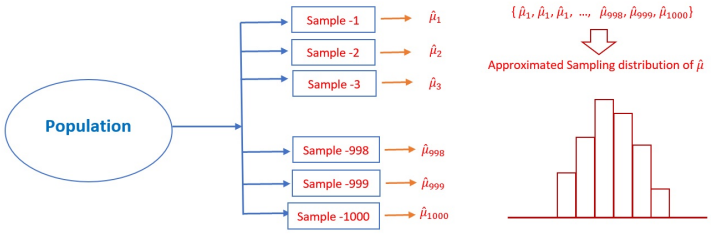


Figure 14.1: Steps for estimating the sampling distribution of a point estimator of the population parameter

14.1.2 Sampling Random Sample - Bootstrap Sampling

Sampling the true population can be very expensive in practice! That means the method of Monte Carlo sampling is infeasible from a practical perspective. The question is whether there are ways to estimate the sampling distribution of the sample means from **a single given random sample**? The answer is YES under the assumption the sample yields a valid estimation of the original population distribution.

- **Bootstrap Sampling** With the assumption that the sample yields a good approximation of the population distribution, we can take bootstrap samples from the **actual** sample. Let

$$\{x_1, x_2, \cdots, x_n\} \rightarrow F(x : \theta)$$

be the actual random sample taken from the population. A **bootstrap sample** is obtained by taking a sample **with replacement** from the original data set (not the population!) with the same size as the original sample. Because **with replacement** was used, some values in the bootstrap sample appear once, some twice, and so on, and some do not appear at all.

- **Notation of Bootstrap Sample**. We use $\{x_1^{(i*)}, x_2^{(i*)}, \cdots, x_n^{(i*)}\}$ to denote the i^{th} bootstrap sample. Then the corresponding mean is called bootstrap sample mean and denoted by $\widehat{\mu}_i^*$, for $i = 1, 2, ..., n$.

- **Bootstrap sampling distribution** of the sample mean can be estimated by taking a large number, say B, of bootstrap samples. The resulting B bootstrap sample means are used to estimate the sampling distribution. Note that, in practice, B is bigger than 1000.

The above Bootstrap sampling process is illustrated in the following figure.

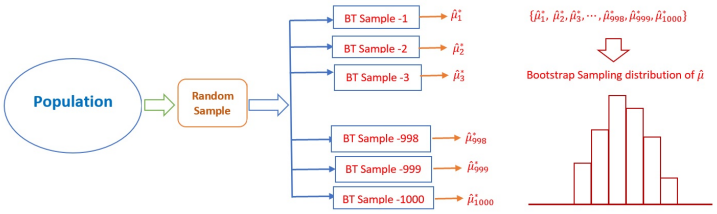


Figure 14.2: Steps for the Bootstrap sampling distribution of a point estimator of the population parameter

14.1.3 Case Study: Confidence Interval of Correlation Coefficient

There is no exact formula for the correlation coefficient of the two variables. Different approximate confidence intervals are available. One of them is based on Pearson transformation. The explicit form of the interval has the following form.

$$\left(\frac{e^{2L}-1}{e^{2L}+1}, \frac{e^{2U}-1}{e^{2U}+1}\right)$$

where

$$L = z_r - \frac{Z_{0.975}}{\sqrt{n-3}}, \quad U = z_r + \frac{Z_{0.975}}{\sqrt{n-3}}$$

and

$$z_r = \frac{\ln(1+r) - \ln(1-r)}{2} \quad \text{and} \quad z_{0.975} = 97.5 \text{ th percentile of the standard normal distribution.}$$

```
y = Carseats$Sales
x = Carseats$Price
plot(Sales ~ Price, data = Carseats, pch=19, cex=0.8)
```

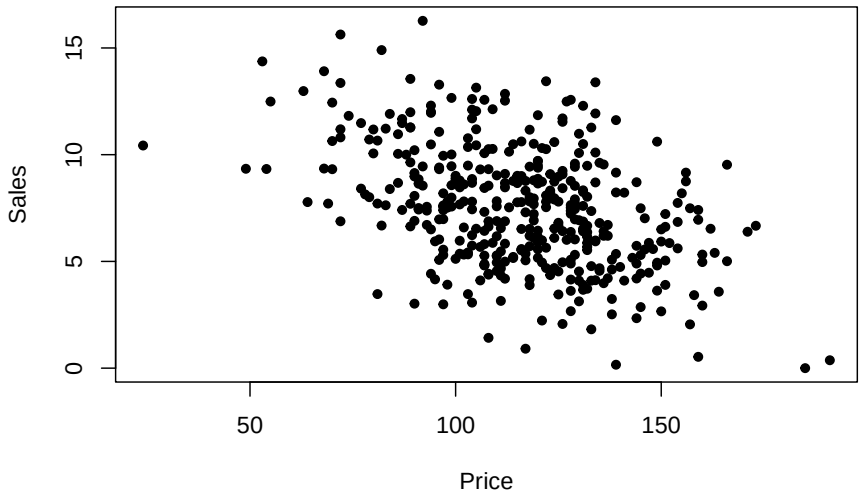


Figure 14.3: The scatter plot between sales and price in car seats data set

```
##  The following is the translation of the given formula
n = length(y)
r = cor(x,y)    # sample Pearson correlation coefficient
zr = (log(1+r)-log(1-r))/2
z.975 = qnorm(0.975)
L = zr - z.975/sqrt(n-3)
U = zr + z.975/sqrt(n-3)
LCI = (exp(2*L)-1)/(exp(2*L)+1)
UCI = (exp(2*U)-1)/(exp(2*U)+1)
CI = cbind(LCI = LCI, UCI = UCI)
### Bootstrap CI
B = 1000              # Take 1000 bootstrap sample
bt.r = c()            # empty vector to store bootstrap coefficient
for ( i in 1:B){
  bt.id = sample(1:n, n, replace = TRUE)   # bootstrapping observation IDs
  bt.x = x[bt.id]     # bootstrap x , must use the above bt.id
  bt.y = y[bt.id]     # bootstrap y , must use the above bt.id
  bt.r[i] = cor(bt.x, bt.y)
}
## 2.5% and 97.5% of the 1000 bootstrap correlation coefficients are the
## lower and upper limits of the 95% bootstrap confidence interval
```

```
bt.CI = quantile(bt.r, c(0.025, 0.975))
list(CI = CI, bt.CI = as.vector(bt.CI))
```

```
## $CI
##              LCI       UCI
## [1,] -0.5203026 -0.362724
##
## $bt.CI
## [1] -0.5214968 -0.3649663
```

14.2 Bootstrap Confidence Interval of AUC for Logistic Model

In this section, we demonstrate how to find the confidence interval of the area of the ROC curve. This confidence interval is of practical importance.

14.3 Bootstrap Sampling for Logistic Modeling

In logistic regression models, the residuals are not defined as those in the linear regression (i.e., e_i = observed y − fitted y). Bootstrapping residuals does not work for logistic regression. We can use the following two steps to take bootstrap samples:

1. Take a bootstrap sample from the observation IDs with the same size (and with replacement): $\{1, 2, 3, \cdots, n\}$. Note that some of the IDs will be sampled multiple times.

2. Using the bootstrap IDs to identify the corresponding records in the data set and defined a new data set called **Bootstrap sample**. Some of the records appear multiple times. These duplicate records with be kept in the sample. In other words, the distinct records in a bootstrap sample are less than the sample size!

Following the above steps, we can take many bootstrap samples. Recall that we can build a logistic regression for a given data set and construct an ROC curve and calculate the area under the curve. This means, if we draw $B = 1000$ bootstrap samples, we can then build 1000 logistic regression models (also called **bootstrap logistic regression models**), hence, will have 1000 **bootstrap AUCs** (one for each bootstrap logistic model).

This further means that the histogram of these **bootstrap AUCs** can be considered as the sampling distribution of the actual sample **AUC**. The **95% Bootstrap confidence interval** of **AUC** is defined as the 2.5% and 97.5% quantiles

of the **bootstrap AUCs**.

14.3.1 Case Study - Confidence Interval of AUC

We present two examples showing how to calculate the bootstrap confidence interval of the AUC using a logistic regression model. One can apply the same steps for neural net and decision tree algorithms.

14.3.1.1 Predicting Graduate Admission

We use the following admission data set to illustrate the steps.

```
admitted = read.csv("https://raw.githubusercontent.com/pengdsci/STA551/main/w09/w09-admitted.csv"
notadmitted = read.csv("https://raw.githubusercontent.com/pengdsci/STA551/main/w09/w09-notAdmitte
## add an admission status variable to both sub-samples.
admitted$status = "yes"                          # admitted
notadmitted$status = "no "                       # not admitted
admission = rbind(admitted, notadmitted)         # combining the two sub-samples
## Define an empty vector to store bootstrap AUCs.
btAUC.vec = c()
## Select the number of bootstrap samples to be generated
B = 1000
## Size of the original sample
sample.size = dim(admission)[1]
## Vector of cut-off probabilities for construct ROC
cut.off.seq = seq(0,1, length = 100)
# bootstrap procedure starts here
for (k in 1:B){
  boot.id = sample(1:sample.size, sample.size, replace = TRUE)   # Bootstrap IDs
  boot.sample = admission[boot.id,]      # Bootstrap samples
  ## Bootstrap logistic regression model is given below
  boot.logistic = glm(factor(status)~., family = binomial, data = boot.sample)
  ##
  pred.prob = predict.glm(boot.logistic, newdata = boot.sample, type = "response")
  ## vectors to store sensitivity and specificity
  sensitivity.vec = NULL
  specificity.vec = NULL
    for (i in 1:100){
   pred.status = as.numeric(pred.prob > cut.off.seq[i])
   ### components for defining various measures
   TN = sum(pred.status == 0 & boot.sample$status == "no ")
   FN = sum(pred.status == 0 & boot.sample$status == "yes")
   FP = sum(pred.status == 1 & boot.sample$status == "no ")
   TP = sum(pred.status == 1 & boot.sample$status == "yes")
```

```
   ###
   sensitivity.vec[i] = TP / (TP + FN)
   specificity.vec[i] = TN / (TN + FP)
  }
  one.minus.spec = 1 - specificity.vec
  sens.vec = sensitivity.vec
  ## A better approx of ROC, need library {pROC}
  prediction = pred.prob
  category = boot.sample$status == "yes"
  ROCobj <- roc(category, prediction, quiet = TRUE)
  btAUC.vec[k] = round(auc(ROCobj),4)
}
hist(btAUC.vec, xlab = "Bootstrap AUC", main = "Bootstrap Sampling Distribution
     of AUCs \n (Admission Prediction Model)")
```

The 95% bootstrap confidence interval of the AUC is defined to be 2.5% and 97.5% quantiles of the bootstrap AUCs.

```
pander(quantile(btAUC.vec, c(0.025, 0.975)))
```

2.5%	97.5%
0.6415	0.7497

The confidence interval of AUC can be used for variable selection: if two confidence intervals of AUC overlapped, the predictive performances of the two corresponding predictive models are not significantly different. A simpler model should be recommended for implementation.

14.3.1.2 Fraud Detection Data

We use the same fraud data that was used in an earlier mote.

```
fraud.data = read.csv("https://pengdsci.github.io/datasets/FraudIndex/fraudidx.csv")[,
## recode status variable: bad = 1 and good = 0
good.id = which(fraud.data$status == " good")
bad.id = which(fraud.data$status == "fraud")
##
fraud.data$fraud.status = 0
fraud.data$fraud.status[bad.id] = 1
```

Next, we perform bootstrap logistic regression with 1000 bootstrap samples and build 1000 bootstrap logistic regression models and calculate the AUC of the ROC of the corresponding bootstrap logistic regression models.

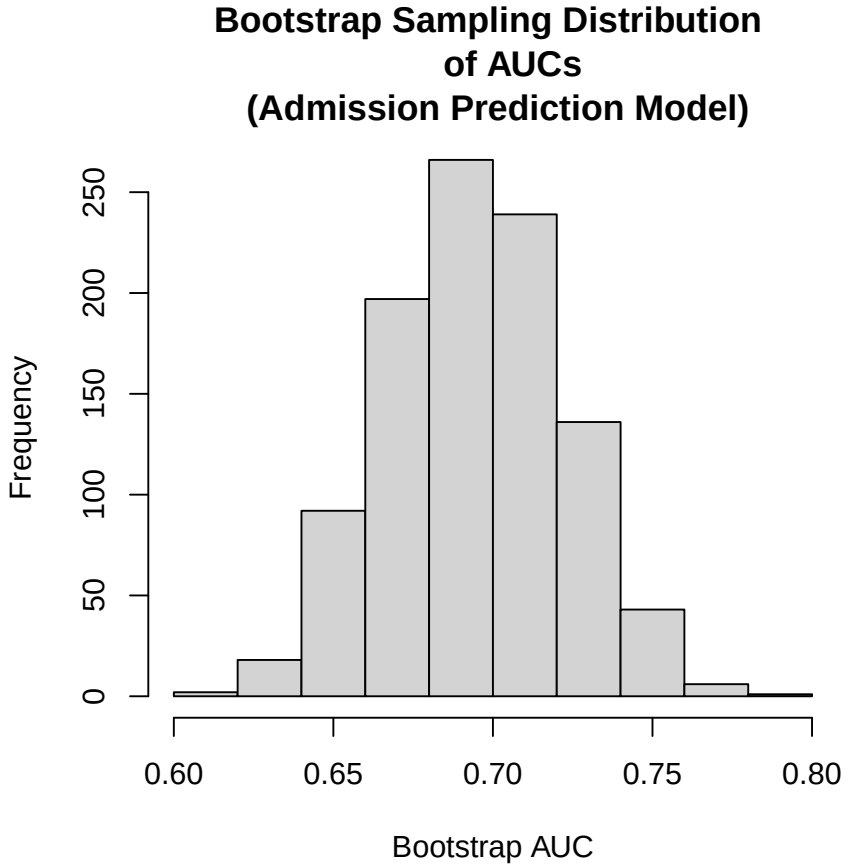


Figure 14.4: The bootstrap sampling distribution of the area under the curve of ROC (graduate admission prediction)

```
## Define an empty vector to store bootstrap AUCs.
btAUC.vec = c()
## select the number of bootstrap samples to be generated
B = 1000
## Size of the original sample
sample.size = dim(fraud.data)[1]
## Vector of cut-off probabilities for construct ROC
cut.off.seq = seq(0,1, length = 100)
# bootstrap procedure starts here
for (k in 1:B){
  boot.id = sample(1:sample.size, sample.size, replace = TRUE)   # Bootstrap IDs
  boot.sample = fraud.data[boot.id,]      # Bootstrap samples
  ## Bootstrap logistic regression model is given below
  boot.logistic = glm(factor(status) ~ index, family = binomial, data = boot.sample)
  ##
  newdata = data.frame(index= boot.sample$index)
  pred.prob = predict.glm(boot.logistic, newdata, type = "response")
  ## vectors to store sensitivity and specificity
  sensitivity.vec = NULL
  specificity.vec = NULL
  for (i in 1:100){
   pred.status = as.numeric(pred.prob > cut.off.seq[i])
   ### components for defining various measures
   TN = sum(pred.status == 0 & boot.sample$fraud.status == 0)
   FN = sum(pred.status == 0 & boot.sample$fraud.status == 1)
   FP = sum(pred.status == 1 & boot.sample$fraud.status == 0)
   TP = sum(pred.status == 1 & boot.sample$fraud.status == 1)
   ###
   sensitivity.vec[i] = TP / (TP + FN)
   specificity.vec[i] = TN / (TN + FP)
  }
  one.minus.spec = 1 - specificity.vec
  sens.vec = sensitivity.vec
  ## A better approx of ROC, need library {pROC}
  prediction = pred.prob
  category = boot.sample$fraud.status == 1
  ROCobj <- roc(category, prediction)
  btAUC.vec[k] = round(auc(ROCobj),4)
}
hist(btAUC.vec, xlab = "Bootstrap AUC",
main = "Bootstrap Sampling Distribution of AUCs \n (Fraud Detection Model)")
```

With the bootstrap AUCs, we can similarly construct the 95% bootstrap confidence interval for the AUC of the fraud detection model is $(0.9234, 0.9313)$.

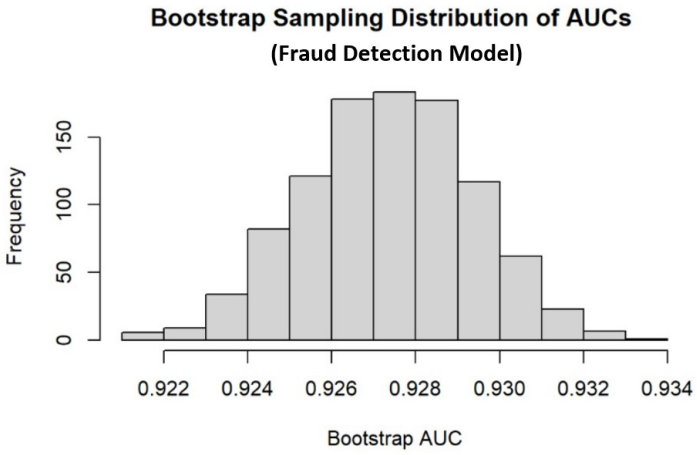


Figure 14.5: Bootstrap sampling distribution of AUC (fraud prediction)

14.4 Concepts of Ensemble Algorithms

We have taken many bootstrap samples and built a logistic regression model on each of these bootstrap samples and calculate the area under the ROC curve of associated logistic regression models. Using these bootstrap AUCs, we approximate the find the bootstrap sampling distribution of the AUC and, hence, find the confidence interval of the AUC.

We can follow the same steps to find the bootstrap confidence intervals of the AUCs of decision trees and neural net algorithms. One important application of Bootstrap in machine learning is its ability to aggregate a set of *weak* models and algorithms to make a *stronger* combined model - This is the so-called **ensemble** learning method in machine learning. One way of improving the performance of a *weak model* through an ensemble approach is to reduce the risk of **overfitting** and **underfitting** by balancing the trade-off between **bias** and **variance**,

14.4.1 Overfitting v.s. Underfitting

In predictive modeling, **bias** is a phenomenon that skews the result of an algorithm in favor of or against the ground truth. It describes how well the model matches the training data set:

- A model with a higher bias would not match the data set closely.
- A low-bias model will closely match the training data set.

Variance refers to the changes in the model when using different portions of the training data set. The variance comes from highly complex (but valid) models with a large number of features

- Models with high bias will have low variance.

- Models with high variance will have a low bias.

The terms **underfitting** and **overfitting** refer to how the model fails to match the data.

- **Underfitting** occurs when the model is too simple to be able to match the input data to the target data.
- **Overfitting** occurs when the model is highly complex but perfectly matches almost all the given data points and performs well in training data sets. However, the model would not be able to generalize the data point in the test data set to predict the outcome accurately due to high error (variation).

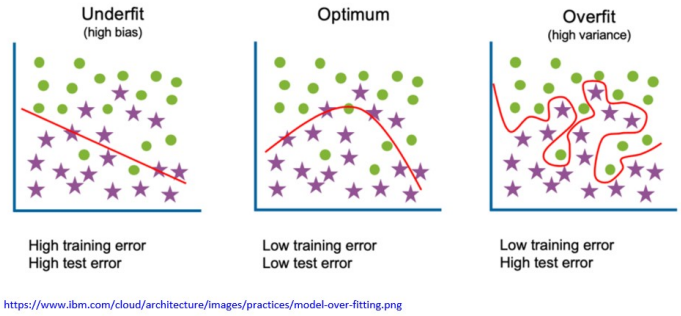


Figure 14.6: Bias, variance, overfitting and underfitting of predictive models

14.4.2 The Logic Ensemble Learning Method

Ensemble learning attests to the idea of the "wisdom of crowds," which suggests that the decision-making of a larger group of people is typically better than that of an individual expert. Similarly, ensemble learning refers to a group (or ensemble) of base learners (eg., models and algorithms), which work collectively to achieve a better final prediction.

A single model or algorithm, also known as a base or weak learner, may not perform well individually due to high variance or high bias. However, when weak learners are aggregated, they can form a strong learner (with stable performance and low variance) yielding better model performance.

Many different types of ensemble learning methods have been developed in the past few decades, Among them, boosting and Bootstrap aggregation (BAGGING) methods are commonly used in practice.

14.4.3 BAGGING Ensemble Methods

Bagging is an acronym for *Bootstrap Aggregation* and is used to decrease the variance in the prediction model (the idea we adopted when identifying the optimal cut-off scores and the confidence interval of AUC).

Bagging is a **parallel** method that fits **different**, considered learners **independently** from each other, making it possible to train them **simultaneously**.

Bagging generates additional data for training from the data set. This is achieved by **bootstrap sampling** (random sampling with replacement) from the original data set. As discussed in earlier sections, **Bootstrap Sampling** may repeat some observations in each new training data set. This guarantees that every element in **Bagging** is equally probable for appearing in a **bootstrap** data set.

These bootstrap data sets are used to train multiple models in **parallel**. The average of all the predictions from different ensemble models is calculated. The **majority vote** gained from the voting mechanism is considered when classification is made. *Bagging decreases the variance and tunes the prediction to an expected outcome*.

The following figure explains the BAGGING algorithm applied to the decision tree algorithm.

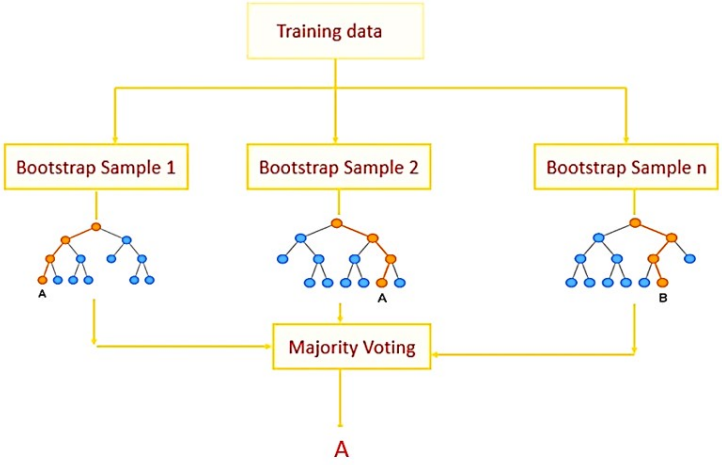


Figure 14.7: The idea of Gagging ensemble algorithm.

In summary, the bagging algorithm has three basic steps:

Bootstrapping: Bagging leverages a bootstrapping sampling technique to create diverse samples.

- **Parallel training**: These bootstrap samples are then trained independently and in parallel with each other using weak or base learners (de).

- **Aggregation**: Finally, aggregating the outputs from individual algorithms/models. In the case of using a tree algorithm (like the above figure)
  - *for regression*, an average of all terminal node weights is taken of all the outputs predicted by the individual classifiers; this is known as **soft voting**.
  - *for classification*, the class (defined with default 0.5 in each individual tree) with the highest majority of votes is accepted; this is known as **hard voting or majority voting**.

14.4.3.1 Case Study I: BAGGING Classification Trees

BAGGING is implemented in R libraries **ipred{}** and **rpart{}**. They are usually used with several other libraries such as **caret{}** and **e1071{}** to extract relevant information in the bagging algorithm.

```
Pima = read.csv("https://pengdsci.github.io/STA551/w03/AnalyticPimaDiabetes.csv")[,-1]
# We use a random split approach
n = dim(Pima)[1]  # sample size
# caution: using without replacement
train.id = sample(1:n, round(0.7*n), replace = FALSE)
train = Pima[train.id, ]    # training data
test = Pima[-train.id, ]    # testing data
```

```
##
Diabetes.bag.train <- bagging(as.factor(diabetes) ~ .,
                              data = train,
                              nbagg = 150,    # number of trees
                              coob = TRUE,
                              parms = list(loss = matrix(c(0, 10, 1, 0),
                                                         ncol = 20,
                                                         byrow = TRUE),
                                                         split = "gini"),
                              control = rpart.control(minsplit = 10,cp = 0.02))
### predict() returns either "class" or "prob" in the classification
### When specifying type = "class", the default cut-off of 0.5 is used.
pred = predict(Diabetes.bag.train, train, type = "prob")
### Optimal cut-off probability identification: no cross-validation is needed
cut.prob = seq(0,1, length = 20)
  senspe.mtx = matrix(0, ncol = length(cut.prob), nrow= 3, byrow = FALSE)
  for (i in 1:length(cut.prob)){
  # CAUTION: "pos" and "neg" are values of the label in this data set!
```

```
  # The following line uses only "pos" probability: pred[, "pos"] !!!!
  pred.out =  ifelse(pred[,"pos"] >= cut.prob[i], "pos", "neg")
  TP = sum(pred.out =="pos" & train$diabetes == "pos")
  TN = sum(pred.out =="neg" & train$diabetes == "neg")
  FP = sum(pred.out =="pos" & train$diabetes == "neg")
  FN = sum(pred.out =="neg" & train$diabetes == "pos")
  senspe.mtx[1,i] = TP/(TP + FN)                  # sensitivity
  senspe.mtx[2,i] = TN/(TN + FP)                  # specificity
  senspe.mtx[3,i] = (TP+TN)/(TP + FN + TN + FP)   # accuracy
  }
  ## A better approx of ROC, need library {pROC}
  prediction = pred[, "pos"]
  category = train$diabetes == "pos"
  ROCobj <- roc(category, prediction)
```

```
## Setting levels: control = FALSE, case = TRUE
```

```
## Setting direction: controls < cases
```

```
  AUC = auc(ROCobj)
  AUC = round(as.vector(AUC[1]),3)
  ###
  n = length(senspe.mtx[3,])
  idx = which(senspe.mtx[3,] == max(senspe.mtx[3,]))
  tick.label = as.character(round(cut.prob,2))
  ###
  par(mfrow = c(1,2))
  plot(1-senspe.mtx[2,], senspe.mtx[1,], xlim=c(0,1), ylim = c(0,1),
       xlab = "1 - specificity",
       ylab = "Sensitivity", main = "ROC (Taining Data)", type="l",cex.main = 0.8)
  legend("bottomright",c("fn = 10", "fp = 1", "cp = 0.02", paste("AUC =", AUC)),
         bty="n", cex = 0.8)

  plot(1:length(cut.prob), senspe.mtx[3,], xlab="cut-off probability",
       ylab = "accuracy", ylim=c(min(senspe.mtx[3,]),1),
              axes = FALSE,
        main="cut-off vs accuracy",
        cex.main = 0.9,
        col.main = "navy")
        axis(1, at=1:20, label = tick.label, las = 2)
        axis(2)
        points(idx, senspe.mtx[3,][idx], pch=19, col = "red")
        segments(idx , min(senspe.mtx[3,]), idx , senspe.mtx[3,][idx ], col = "red")
       text(idx, senspe.mtx[3,][idx]+0.03, as.character(round(senspe.mtx[3,][idx],4)),
            col = "red", cex = 0.8)
```

Similarly, there are several **hyperparameters** one can **tune** to find the best-

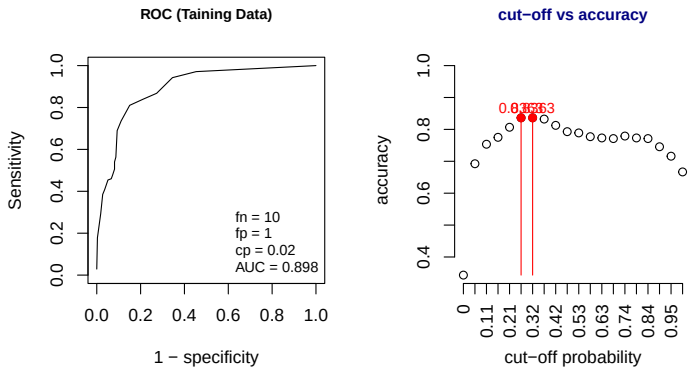


Figure 14.8: The ROC curve and the plot for optimal cut-off determination.

bootstrapped decision tree. For those who are interested in exploring more in **tunning hyperparameters**, you can write a wrapper of **bagging()** and pass hyperparameters such as **fp, fn, cp, minisplit** in the list of arguments in **rpart()** to find the best bootstrap decision tree.

14.4.4 Boosting Ensemble Methods

Boosting is a sequential ensemble method that iteratively **adjusts the weight of data points** as per the last classification. If a data point is incorrectly classified, it increases the weight of that data point. It decreases the bias and variance (hence the predictive error) and builds strong predictive models.

Data points misclassified in each iteration are spotted, and their weights are increased. The Boosting algorithm allocates weights to each resulting model during training. A model (also commonly called learner) with good training data prediction results will be assigned a higher weight. When evaluating a new learner, Boosting keeps track of the learner's errors.

The steps of boosting algorithm are summarized in the following:

1. Data points in the initial training data set are equally weighted.
2. A based model is created for the initial training data set.
3. classification Errors are counted using actual and predicted values. The data point that was incorrectly predicted is provided a higher weight.
4. a model is built on the modified data (re-weighted data points)
5. the process is iterated for multiple models and each of them corrects the previous model's errors.

6. The final model works as a strong learner and shows the weighted mean of all the models.

The following figure demonstrates the rough idea of boosting decision trees in a conceptual approach.

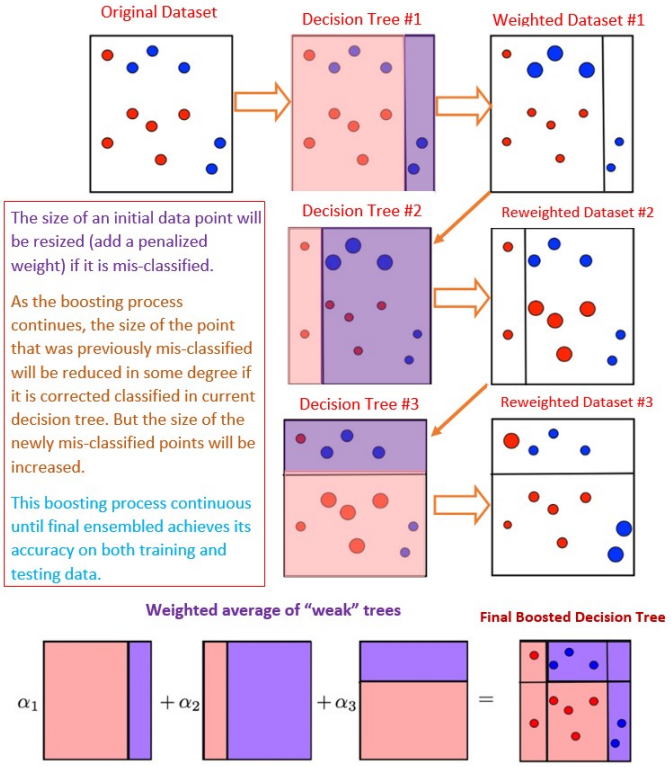


Figure 14.9: Graphical representation of boosting tree method

14.4.5 Bagging Versus Boosting

Bagging and **Boosting** have a universal similarity of being classified as ensemble methods. In addition, Bagging and Boosting

- are ensemble methods focused on getting N learners from a single learner.
- make random sampling and generate several training data sets.
- arrive upon the end decision by making an average of N learners or taking the voting rank done by most of them.
- reduce variance and provide higher stability by minimizing errors.

However, the two ensemble algorithms are very different from the technical perspective. The following table shows these structural differences.

Bagging	Boosting
Merging the same type of predictions.	Merging different types of predictions.
Decreases variance, not bias, and solves over-fitting issues.	Decreases bias, not variance.
Each model receives an equal weight.	Models are weighed based on their performance.
Models are built independently.	New models are affected by a previously built model's performance.
Training data subsets are drawn randomly with a bootstrap sample.	Every new subset comprises the elements that were misclassified by previous models.
Usually applied where the classifier is unstable and has a high variance.	Usually applied where the classifier is stable and simple and has a high bias.

The boosting ensemble algorithms have various new development in recent years. In general, they are more mathematically and pragmatically demanded in implementation. We will not go into details about any of boosting algorithms and implement them.

To conclude, we use the following figure to show the role of mathematical tools and thinking in the evolution of tree-based algorithms: the more mathematical tools you have, the more powerful models you are capable of developing!

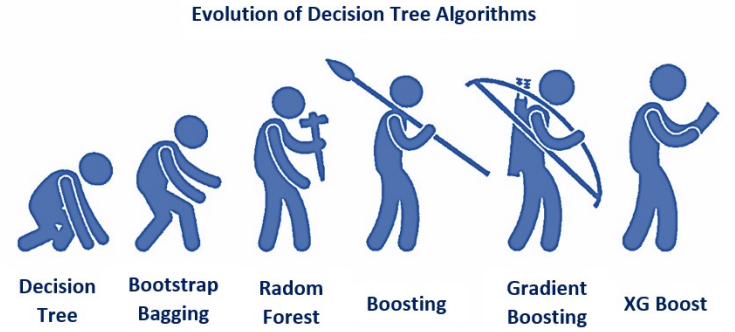


Figure 14.10: Evolution of tree-based algorithms with various level of technical tools.

Chapter 15

Algorithms for Anomaly Detection

Credit card fraud is a type of identity theft that occurs when someone illegally uses another person's credit card or account information for an unauthorized transaction. Fraud can happen as a result of a stolen, misplaced, or counterfeit credit card.

According to Nilson Report [https://nilsonreport.com/content_promo.php?id_promo=16], card fraud losses worldwide reached $28.65 billion in 2020. By 2025, the United States is projected to reach $12.5 billion in card fraud losses.

Figure 15.1: The trend of worldwide card fraud loss.

Among all card fraud, credit card fraud contributes a significant portion. Unlike the process of identifying credit default in which we have a lot of information about cardholders, in the fraud domain, we have no information about fraudsters

except for the amount, location, and time of the fraudulent transactions. This makes the detection and identification of credit card fraud more challenging.

There are different types of credit card fraud. The most frequently occurring types of credit card fraud include lost and stolen, counterfeit, card-not-present (CNP), account takeover, application fraud, card-never-received, etc. The different strategies of prevention for different types of fraud have been developed by financial industries over the years. These include educating cardholders on how to protect their personal information, increasing security measures such as using chip-enabled cards and enhanced web portals for online transactions, monitoring card shipment, etc.

Fraud detection, on the other hand, is a more complex and difficult process. This is where analytics come into play.

In the rest of this note, we will

- overview of the statistical models and machine learning algorithms that are used in credit card fraud detection;
- introduce the statistical foundation of the proposed algorithm;
- detail the proposed new algorithms based on sequential data;
- evaluate the performance of the proposed algorithms
- introduce the potential improvement and generalization of the proposed algorithms.

15.1 Anomaly Detection Use Case: Fraud Operation

In essence, identifying fraud is an anomaly detection process that identifies unusual patterns that do not fit normal behavior in the data generation process. In addition to fraud detection, anomaly detection has many applications in business such as intrusion detection (identifying strange patterns in network traffic), health monitoring (spotting a malignant tumor in an MRI scan), fault detection in operating environments, etc.

Three different types of anomalies can be found in credit card fraud.

Global Anomalies – if a data point is too far from the rest, it falls into the category of point anomalies (outliers). For example, if all historical fuel costs are below $75, but the most recent transaction is $149.50. The most recent transaction is abnormal.

Contextual Anomalies – If the event is anomalous in specific circumstances (context), then we have contextual anomalies. As data becomes more and more complex, it is vital to use anomaly detection methods for the context. This anomaly type is common in sequence data. For example, - a card was used for fuel roughly 4-5 times a month historically, but the same card was used at gas pumps 5 times within the last 3 hours.

Collective Anomalies. The collective anomaly denotes a collection of anomalous concerning the multiple feature variables, but not individual objects. For example: if one card was skimmed at a pump, the fraudsters then make many duplicate cards and sell them to card other fraudsters. The counterfeit cards may be used in very different geographic locations at approximately the "same time". Each transaction associated with this card may be normal. However, we will see the abnormality if we look at these transactions collectively.

Sometimes, people break anomaly detection algorithms down into two subclasses: outlier detection and novelty detection.

- **Outlier detection**: the input data set contains examples of both standard events and anomaly events. These algorithms seek to fit regions of the training data where the standard events are most concentrated, disregarding, and therefore isolating the anomaly events. Such algorithms are often trained in an unsupervised fashion (i.e., without labels). We sometimes use these methods to help clean and pre-process datasets before applying additional machine-learning techniques.
- **Novelty detection**: Unlike outlier detection, which includes examples of both standard and anomaly events, novelty detection algorithms have only the standard event data points (i.e., no anomaly events) during training time. During training, these algorithms use only labeled examples of standard events (supervised learning). At the time of testing/prediction, novelty detection algorithms must detect when an input data point is an outlier.

Various machine learning algorithms and statistical models could be used for credit card fraud detection dependent on the amount of information on the types of fraud. But they are broadly classified into supervised and unsupervised methods. Some of the classical methods are outlined in the following subsections.

15.2 Supervised and Unsupervised Anomaly Detection

Depending on whether the labels are available, anomaly detection techniques can be categorized into one of the following three modes:

15.2.1 Supervised Anomaly Detection

Techniques trained in supervised mode assume the availability of a training data set that has labeled instances for normal as well as anomaly classes. A typical approach in such cases is to build a predictive model for normal vs. anomaly classes. Any unseen data instance is compared against the model to determine which class it belongs to.

Two major issues arise in supervised anomaly detection.

- **Imbalance Labels**: The anomalous instances are far fewer compared to the normal instances in the training data. Issues that arise due to imbalanced class distributions have been addressed in the data mining and machine-learning literature
- **Inaccuracy of Label**: obtaining accurate and representative labels, especially for the anomaly class is usually challenging. This is particularly true in the world of credit card fraud operations.

Other than these two issues, the supervised anomaly detection problem is similar to building predictive models such as logistic regression and decision tree algorithms.

15.2.2 Semi-Supervised Anomaly Detection.

These types of methods usually assume that the training data has labeled instances for only the normal class. Since they do not require labels for the anomaly class, they are more widely applicable than supervised techniques. The typical approach used in such techniques is to build a model for the class corresponding to normal behavior and use the model to identify anomalies in the test data.

A limited set of anomaly detection techniques exist that assume the availability of only the anomaly instances for training. Such techniques are not commonly used, primarily because it is difficult to obtain a training data set that covers every possible anomalous behavior that can occur in the data.

Sometimes, we have to use supervised and unsupervised methods to label some unlabeled examples to reach the minimum sample size for using supervised anomaly detection methods.

15.2.3 Unsupervised Anomaly Detection.

These types of methods operate in unsupervised mode and do not require training data, and thus are most widely applicable. The techniques in this category make the implicit assumption that normal instances are far more frequent than anomalies in the test data. If this assumption is not true then such techniques suffer from a high false alarm rate.

Many semi-supervised techniques can be adapted and used in an unsupervised mode by using a sample of the unlabeled data set as training data. Such adaptation assumes that the test data contains very few anomalies and the model learned during training is robust to these few anomalies.

In the following sections, we only select two representative anomaly detection algorithms: outlier detection and novelty detection.

15.3 Local Outlier Factor Method: Outlier Detection

The LOF [proposed by Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jorg Sander in 2000] is the most well-known local anomaly detection algorithm whose idea is carried out in many nearest-neighbor-based algorithms.

15.3.1 Definitions of Some `Distances`

Before introducing the steps for calculating the LOF score, we define the following technical terms

Normal Distance (ND) - any of the valid "statistical distances" such as Euclid, Minkowski, Manhattan, etc.

k-distance (kD): For the pre-selected k, k-distance is defined to be the distance of a (new) point to its kth neighbor. For example, if k was 3, the k-distance of A, denoted by `k-distance(A)`, would be the distance of point A to its `third closest` point which is D (B is the first closest neighbor, C is the second closest neighbor), see the following Figure 3.

Reachability Distance (RD) is defined to be the maximum distance of two points and the k-distance of the `second` point. For example, the reachability distance between A and D is given by

```
Reachability-Distance(A, D) = max{k-distance (D), normal_distance
(A, D)}
```

Where `normal-distance(A, D)` could be any "statistical distance" that is used in `k-distance(D)`.

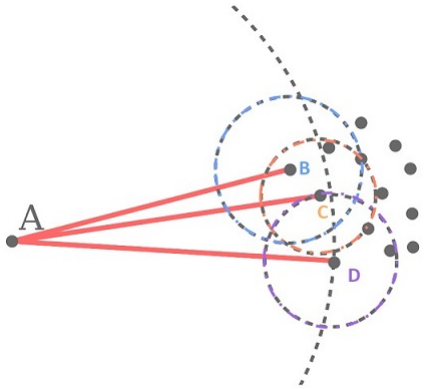


Figure 15.2: Illustration of calculating LOF score.

Local Reachability Density (LRD) refers to how far we need to go from the point we are at to reach the next point or set of points. For example, For all

k =3 closest neighbors of A, the reachability distances are calculated and the values found are summed and divided by the value of k =3. When the inverse of this value is taken, we calculate the density we are looking for.

```
Local-reachability-density(A) = 1 / ( sum ( Reachability_Distance
(A , n ) ) / k)
```

n is the number of points/neighbors centered at point A.

Local Outlier Factor (LOF) Score

The local reachability densities found are compared to the local reachability densities of A's nearest k neighbors. The density of each neighbor is summed up and divided by the density of A. The value found is divided by the number of neighbors i.e. k.

```
LOF(A) =[(LRD(1st. neighbor) + LRD(2nd. neighbor) + ... + LRD(kth.
neighbor))/LRD(A)]/k
```

Use of LOF Score - A value of approximately 1 indicates that the object is comparable to its neighbors. A value below 1 indicates a denser region (which would be an inlier), while values significantly larger than 1 indicate outliers.

- **LOF(k) ~ 1** means Similar density as neighbors,
- **LOF(k) < 1** means Higher density than neighbors (Inlier),
- **LOF(k) > 1** means Lower density than neighbors (Outlier)

15.3.2 A Toy Example

The calculation of the LOF score is not difficult. However, it involves multiple steps and several "distance"-based measures. In this section, we use a toy example with 4 points depicted in the following figure.

The following figure shows the steps for calculating the LOF score for each of the four points in the toy data set.

We can see that the LOF score of point c is 2. Point c is a potential outlier.

15.3.3 Implementation of LOF in R

Several R packages have a function to calculate LOF scores. We use **lof()** in **{dbscan}** to calculate LOF scores of data points in the well-known **iris** data.

lof() can calculate the LOF score in a high dimensional space. The original iris data has 4 numerical variables (sepal and petal widths and lengths). We will calculate the LOF scores based on these variables (in 4-dimensional space).

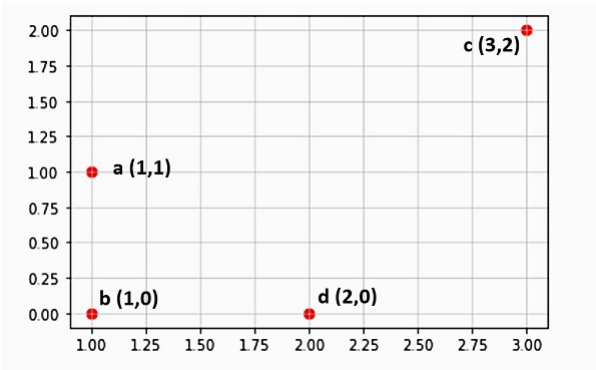


Figure 15.3: Toy data set.

Steps of Manual Calculation of Local Outlier Factor Score

1. Normal Distance (Manhattan) (ND_M)

ND_M (a, b) = 1
ND_M (a, d) = 2
ND_M (b, d) = 1
ND_M (a, c) = 3
ND_M (b, c) = 4
ND_M (c, d) = 3

2. k-Distance (k = 2)

kD_2 (a) = 2
kD_2 (b) = 1
kD_2 (c) = 3
kD_2 (d) = 2

3. Neighborhood with k = 2

N_2(a) = {b, d}
N_2(b) = {a, d}
N_2(c) = {a, d}
N_2(d) = {a, b}

4. Reachability Distance

RD (a, b) = max {kD_2 (b), ND (a, b)} = max {1,1} = 1
RD (a, d) = max {kD_2 (d), ND (a, d)} = max {2,1} = 2
RD (b, a) = max {kD_2 (a), ND (b, a)} = max {2,1} = 2
RD (b, d) = max {kD_2 (d), ND (b, d)} =max {2,1} = 2
RD (c, a) = max {kD_2 (a), ND (C, a)} = max {2,3} = 3
RD (c, d) = max {kD_2 (d), ND (C, d)} = max {2,3} = 3
RD (d, a) = max {kD_2 (a), ND (d, a)} = max {2,2} = 2
RD (d, b) = max {kD_2 (b), ND (d, b)} = max {1,1} = 1

5. Local Reachability Density

LRD(a) = 1 / (RD (a, b) + RD (a, d)) / 2) = 2/3
LRD(b) = 1 / (RD (b, a) + RD (b, d)) / 2) = 1/2
LRD(c) = 1 / (RD (c, a) + RD (c, d)) / 2) = 1/3
LRD(d) = 1 / (RD (d, a) + RD (d, b)) / 2) = 2/3

6. Local Outlier Factor Score

LOF(a) = ((LRD(b) + LRD(d)) / LRD(a)) / 2 = [(1/2 + 2/3) / (2/3)] / 2 = 7/8 = 0.875
LOF(b) = ((LRD(a) + LRD(d)) / LRD(b)) / 2 = [(2/3 + 2/3) / (1/2)] / 2 = 4/3 = 1.333
LOF(c) = ((LRD(a) + LRD(d)) / LRD(c)) / 2 = [(2/3 + 2/3) / (1/3)] / 2 = 4/2 = 2
LOF(d) = ((LRD(a) + LRD(b)) / LRD(d)) / 2 = [(2/3 + 1/2) / (2/3)] / 2 = 7/8 = 0.875

Figure 15.4: Steps for calculating LOF scores.

To visualize the LOF score, we also perform a PCA and use the first two PCs (which account for about 95% of the total variation) to calculate LOF scores.

A Cautionary Note on LOF Scores with PCA - The purpose of calculating LOF scores based on the first two-component analysis is to visualize the outliers in a 2-dimensional plot. The original variables must not be scaled in the PCA to obtain comparable LOF scores. The translation of the original variable will give the same LOF score.

```
log.iris = log(iris[,-5])   # drop the categorical variable in the original
                            # data set and transform all numerical to the
                            # log-scale
ir.pca <- prcomp(log.iris, center = TRUE, scale = FALSE)
# use the first two PCs to define a data frame for LOF
pca.iris = data.frame(ir.pca$x[, 1:2])
### Calculate the two LOF scores with the original variables and PCs respectively.
lof.pca <- lof(pca.iris, minPts = 30)  #minPts = k value
lof.orig <- lof(log.iris, minPts = 30)
## 2D plot of LOF score based on PCs
plot(pca.iris, pch = "x",    # point symbol
     main = "LOF Based on PCA",
     #asp = 1,
     cex = 0.5)                # aspect ratio - ratio of 'y/x'
points(pca.iris,
       cex = (lof.pca-1)*1.5,      # point size according to the LOF score
       pch = 21,
       col = "purple")
text(pca.iris[lof.pca > 1.8,],
     labels = round(lof.pca, 1)[lof.pca > 1.8],
     pos = 1,   # 1, 2, 3 and 4 => below, left , above, and right
     cex = 0.7,
     col = "blue")
```

```
plot(lof.pca, lof.orig, pch ="x",
     main = "LOF Scores Comparison: PCA vs Original Variable",
     xlab = "PCA LOF Score",
     ylab = "Original LOF Score",
     cex = 0.6)
points(lof.pca[lof.pca > 1.8], lof.orig[lof.pca > 1.8],
       pch = 21,
       cex = lof.pca*1.5,
       col = "purple")
text(lof.pca[lof.pca > 1.8], lof.orig[lof.pca > 1.8],
     labels = round(lof.pca, 1)[lof.pca > 1.8],
     pos = 1,   # 1, 2, 3 and 4 => below, left , above, and right
     cex = 0.7,
     col = "blue")
```

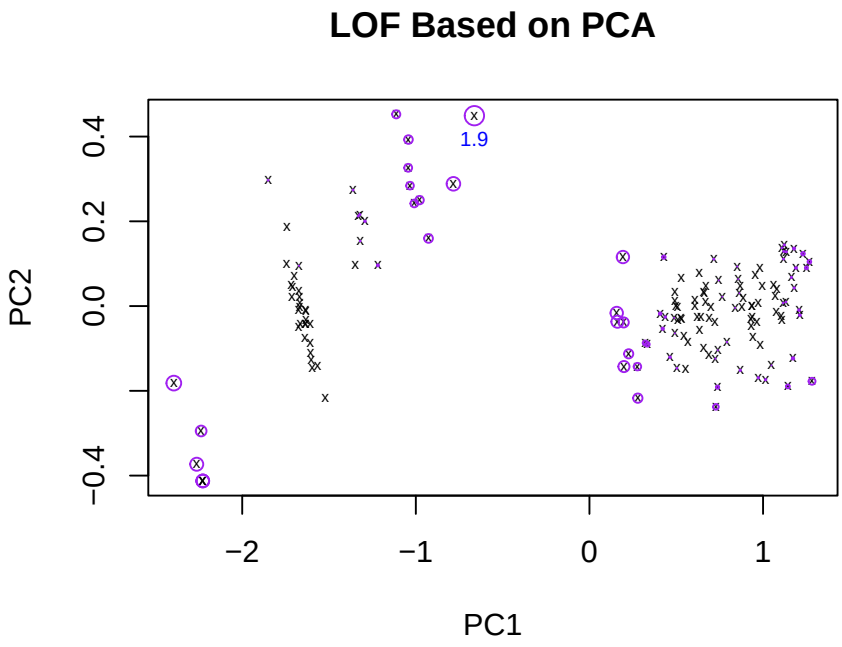


Figure 15.5: LOF scores based on the first 2 principal components.

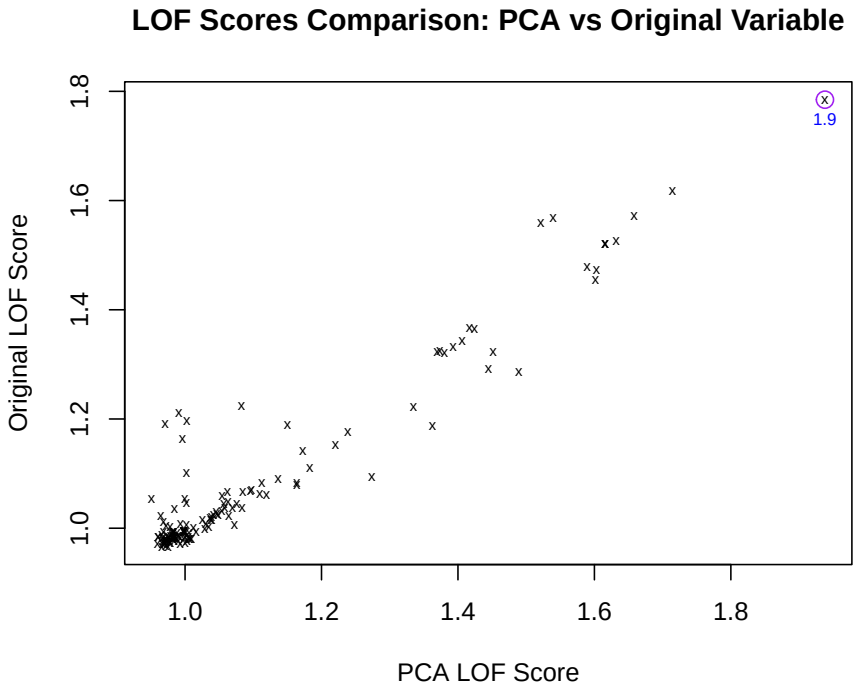


Figure 15.6: Scatter plot of LOF scores based on the original variables and the first 2 PCs.

The above two figures show that the LOF method identifies the same outliers based on the original four variables (in the 4-dimensional feature space) and the first two PCs (2-dimensional space). No variable scaling was used in the PCA.

15.4 One-class SVM: Novelty Detection

Many machine learning models throw inaccuracies in the modeling of outlier elements. So it becomes a most basic requirement for us to determine if the new observation comes under the same existing distribution or if the new observation should be determined as different. In other words, there is only one class in the practical problem. The methods introduced in the following subsections are based on the non-linear kernel function. We will not introduce the mathematics behind these methods. But I will give a brief description of how the kernel function works.

15.4.1 Kernel Function

A kernel function is a weighing function. It defines an inner product (dot product) based on transformed feature variables. It can map a lower-dimensional nonlinear classification problem to a higher-dimensional space to obtain a linear classification problem.

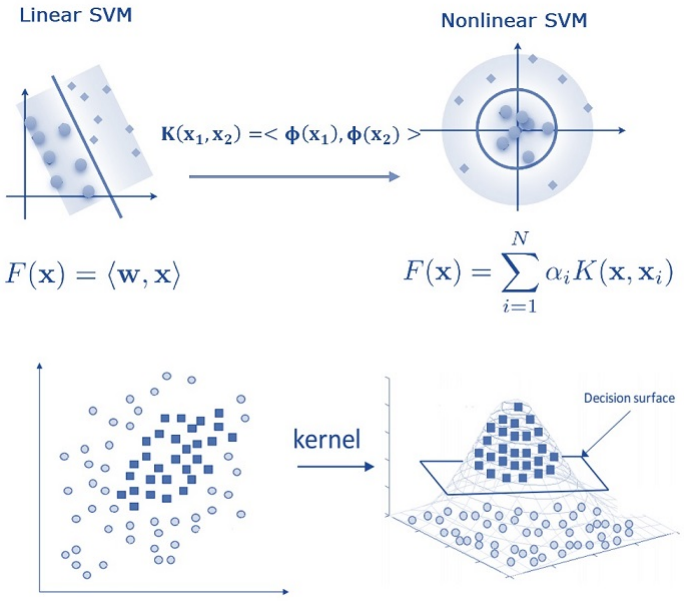


Figure 15.7: Kernel function.

There are different kernel functions used in various applications. The com-

monly used kernel functions are linear, quadratic, exponential, sigmoid, Gaussian radial-based function (RBF), etc.

15.4.2 Description of Support Vector Data Description (SVDD)

Support Vector Data Description (SVDD, developed by Tax and Duin in 2004) is a one-class classification technique that is useful in applications where data that belong to one class are abundant but data about any other class are scarce or missing. SVDD can

- Identifies minimum radius hyper-sphere around "normal" data
- Works on multivariate data
- Does not require the assumption of normality
- Fits flexible surfaces using a kernel function
- Minimizes the chance of accepting outliers

It creates a minimum-radius hyper-sphere around the training data set and scores new observations by calculating the distance to the hyper-sphere center. Observations with distances greater than the minimum radius are flagged as anomalies. It has been used in the areas such as fraud detection, equipment health monitoring, and process control where the majority of the data belong to one class.

SVDD's foundation is built based on the primal-dual algorithms (a method for solving linear programs inspired by the Ford–Fulkerson). It uses a kernel-based approach to define a score function for each observation and obtain a decision boundary on whether new incoming data is an outlier.

To avoid mathematical expression, the following figure illustrates the

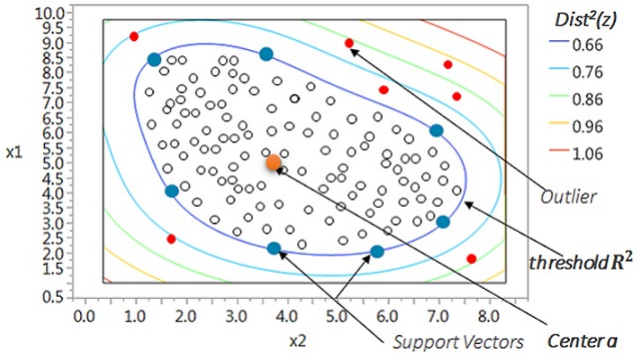


Figure 15.8: Illustration of SVDD.

15.4.3 One-Class Support Vector Machine (OC-SVM)

The OC-SVM formulation is equivalent to the SVDD formulation when the **RBF kernel** function is used.

In this subsection, we use the R function **svm()** in package **{e1071}** to perform one-class support vector machine analysis using the iris data. We need to define a training data set. For simplicity, we use all setosa records to train the OC-SVM and then use the entire iris data to calculate the accuracy metrics.

```
data(iris)
df <- iris
##
df <- subset(df ,  Species=='setosa')          # choose only one of the classes
x <- subset(df, select = -Species)             # make x variables
y <- df$Species                                # make y variable(dependent)
model <- svm(x, y, type ='one-classification') # train an one-classification model
```

```
## Warning in Ops.factor(yorig, ret$fitted): '-' not meaningful for factors
```

```
# print(model)
summary(model) #print summary
```

```
##
## Call:
## svm.default(x = x, y = y, type = "one-classification")
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.25
##          nu:  0.5
##
## Number of Support Vectors:  27
##
##
##
##
## Number of Classes: 1
```

```
# Test on the whole set
pred <- predict(model, subset(iris, select = -Species)) #create predictions
actual = (iris[,5]== 'setosa')   # actual labels
confusion.matrix = table(pred, actual)
confusion.matrix
```

```
##        actual
## pred    FALSE TRUE
```

```
##   FALSE   100   25
##   TRUE      0   25
```

Question: Can we mimic the case study in LOF to use the first two principal components to perform OC-SVM?