MACHINE LEARNING MADE EASY With R

Intuitive Step by Step Blueprint for Beginners

Dr. N.D Lewis

ISBN-13: 978-1546483755
ISBN-10: 1546483756

Contents

Finally, A Blueprint for Machine Learning with R!

Machine Learning Made Easy with R offers a practical tutorial that uses hands-on examples to step through real-world applications using clear and practical case studies. Through this process it takes you on a gentle, fun and unhurried journey to creating machine learning models with R. Whether you are new to data science or a veteran, this book offers a powerful set of tools for quickly and easily gaining insight from your data using R.

NO EXPERIENCE REQUIRED: **Machine Learning Made Easy with R** uses plain language rather than a ton of equations; I'm assuming you never did like linear algebra, don't want to see things derived, dislike complicated computer code, and you're here because you want to try successful machine learning algorithms for yourself.

YOUR PERSONAL BLUE PRINT: Through a simple to follow intuitive step by step process, you will learn how to use the most popular machine learning algorithms using R. Once you have mastered the process, it will be easy for you to translate your knowledge to assess your own data.

THIS BOOK IS FOR YOU IF YOU WANT:

- Focus on explanations rather than mathematical derivation.
- Practical illustrations that use real data.
- Worked examples in R you can <u>easily</u> follow and <u>immediately</u> implement.
- Ideas you can actually use and try out with your own data.

TAKE THE SHORTCUT: **Machine Learning Made Easy with R** was written for people who want to get up to speed as quickly as possible. In this book, you will learn how to**:**

- **Unleash** the power of Decision Trees.
- **Develop** hands on skills using k-Nearest Neighbors.
- **Design** successful applications with Naive Bayes.
- **Deploy** Linear Discriminant Analysis.
- **Explore** Support Vector Machines.
- **Master** Linear and logistic regression.
- **Create** solutions with Random Forests.
- **Solve** complex problems with Boosting.
- **Gain** deep insights via K-Means clustering.
- **Acquire** tips to enhance model performance.

QUICK AND EASY: For each machine learning algorithm, every step in the process is detailed, from preparing the data for analysis, to evaluating the results. These steps will build the knowledge you need to apply them to your own data science tasks. Using plain language, this book offers a simple, intuitive, practical, non-mathematical, easy to follow guide to the most successful ideas, outstanding techniques and usable solutions available using R.

GET STARTED TODAY! Everything you need to get started is contained within this book. **Machine Learning Made Easy with R** is your very own hands on practical, tactical, easy to follow guide to mastery.

Buy this book today and accelerate your progress!

Preface

THIS book is about understanding and then hands use of machine learning algorithms for prediction and classification; more precisely, it is an attempt to give you the tools you need to build machine learning models easily and quickly using R. The objective is to provide you the reader with the necessary knowledge to do the job, and provide sufficient illustrations to make you think about genuine applications in your own field of interest. I hope the process is not only beneficial but enjoyable.

Caution!

On its own, this text won't turn you into a machine learning guru any more than a few dance lessons will turn you into the principal dancer with the Royal Ballet in London. But if you're a working professional, economist, business analyst or just interested in trying out new machine learning ideas, you will learn the basics, and get to play with some cool tools. Once you have mastered the fundamentals, you will be able to use these ideas using your own data.

If you are looking for detailed mathematical derivations, lemmas, proofs or implementation tips, please do not purchase this book. It contains none of those things.

You don't need to know complex mathematics, algorithms or object-oriented programming to use this text. It skips all that stuff and concentrates on sharing code, examples and illustrations that gets practical stuff done.

Before you buy this book, ask yourself the following tough questions. Are you willing to invest the time, and then work through the examples and illustrations required to take your knowledge to the next level? If the answer is yes, then by all means click that buy button so I can purchase my next cappuccino.

A Promise

No matter who you are, no matter where you are from, no matter your background or schooling, you have the ability to master the ideas outlined in this book. With the appropriate software tool, a little persistence and the right guide, I personally believe the techniques outlined in this book can be successfully used in the hands of anyone who has a real interest.

When you are done with this book, you will be able to implement one or more of the ideas I've talked about in your own particular area of interest. You will be amazed at how quick and easy the techniques are to develop and test. With only a few different uses you will soon become a skilled practitioner.

I invite you therefore to put what you read in these pages into action. To help you do that, I've created **"12 Resources to Supercharge Your Productivity in R"**, it is yours for **FREE**. Simply go to *http://www.AusCov.com* and download it now. It is my gift to you. It shares with you 12 of the very best resources you can use to boost your productivity in R.

Now, it's your turn!

Dr. N.D. Lewis

Chapter 1

Introduction to Machine Learning

WE are living in the "*quantum data generation age*". Your computer tracks websites visited, social media collects data on your interests and friendships; even your smartphone knows your location and favorite route to work. Retailers, corporations, pharmaceutical companies, local government and national governments are collecting, storing and analyzing more data than ever.

This introductory chapter:

- Outlines who this book is for.
- Answers the question "What is Machine Learning?"
- Provides an overview of the machine learning process.
- Explains how you can get the most out of this book.
- Provides information on getting and using R.

Buried deep in the ever-growing mountain of electronic digits are complex relationships, new insights and groundbreaking discoveries. Machine learning algorithms are designed to help people like you extract this knowledge.

Who is this Book for?

It's no accident that the words simple, easy and gentle appear so often in this text. I have shelves filled with books about machine learning, statistics, computer science, and econometrics. Some are excellent, others are good, or at least useful enough to keep. But they range from very long to the very mathematical. I believe many working professionals want something short, simple with practical examples that are easy to follow and straightforward to implement. In short, a very gentle intuitive introduction to applied machine learning.

Also, almost all advice on machine learning comes from academics; this comes from a practitioner. I have been a practitioner for most of my working life. I enjoy boiling down complex ideas and techniques into applied, simple and easy to understand language that works. Why spend five hours ploughing through technical equations, proofs and lemmas when the core idea can be explained in ten minutes and deployed in fifteen?

I wrote this book because I don't want you to spend your time struggling with the mechanics of implementation or theoretical details. That's why we have Ivy league (in the US) or Russell Group (in the UK) professors. Even if you've never attempted to forecast anything, you can easily make your computer do the grunt work. This book will teach you how to apply the very best R machine learning tools for prediction and classification.

What is Machine Learning?

Machine learning is a collection of algorithms that generate insight from data. That insight might be used by humans or other machines to make a decision. For example, a team of primary care researchers and computer scientists at Nottingham University used machine learning algorithms for cardiovascular disease risk assessment. The machine learning algorithms were better than experienced medical doctors at gauging heart attack risk:

> "*Machine-learning significantly improves accuracy of cardiovascular risk prediction, increasing the number of patients identified who could benefit from preventive treatment, while avoiding unnecessary treatment of others.*"

Machine learning tasks

Machine learning is used for two primary tasks. The first task, is forecasting future outcomes. For example, we might want to optimize the traffic light sequences at a busy intersection. A machine learning algorithm might be developed to predict the five minute ahead traffic flow.

The second task, is classification of objects into specific classes. For example, coffee beans are usually classified into one of four grades (Specialty, Premium, Exchange, Standard). An industrial scale purchaser of beans, might develop a machine learning algorithm to automatically grade bean quality.

Figure 1.1 illustrates the typical workflow used to develop machine learning models. The sample data is collected, features determined, and a machine learning algorithm selected.

The selected model is then trained on a specially selected subset of the data, referred to as the training set or train sample. If the classification or predictive performance is acceptable, the model is further validated on an independent test set. This is a sub-sample of the original sample data not previously seen by the model.

The entire process is iterative. Typically, many iterations are required before a final model is selected. This model (or models) is then deployed on new, previously unseen, real world data.

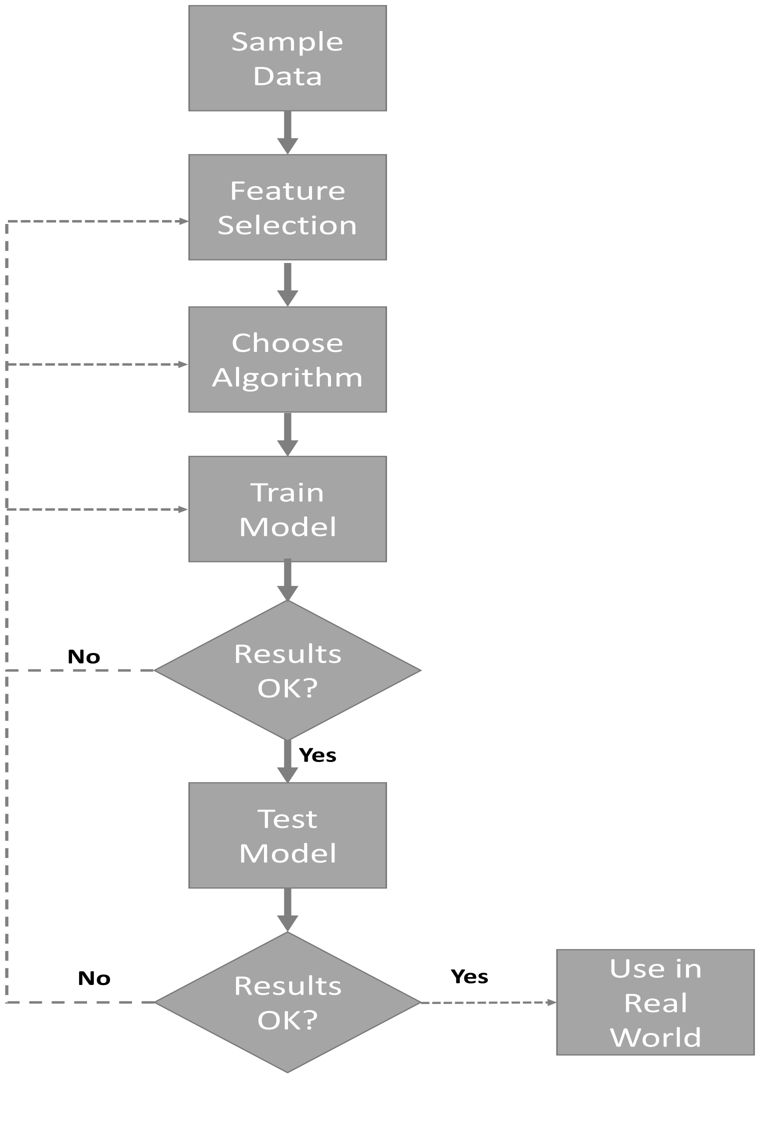


Figure 1.1: Overview of the machine learning process

Prediction accuracy versus interpretability

Whilst the primary interest of machine learning is accurate prediction or classification, other empirical sciences such as econometrics focus primarily on interpreting model parameters, often in the light of prior economic theory.

Machine learning models often produce results which cannot be matched by traditional empirical techniques. Unfortunately, meaningful interpretation of model parameters is often infeasible. For the most part, you will find a clear trade-off between predictive accuracy and interpretability.

How to Get the Absolute Most Possible Benefit from this Book

I want you to get the absolute most possible benefit from this book in the minimum amount of time. You can achieve this by typing in the examples, reading the reference material and most importantly experimenting. This book will deliver the most value to you if you do this.

It is always a good idea to study how other users and researchers have applied a technique in actual practice. This is primarily because practice often differs substantially from the classroom or theoretical text books. To this end and to accelerate your progress, numerous applications of machine learning use are given throughout this text. In addition, at the end of each chapter is a list of selected readings. The reading provides additional details on the applications discussed in the chapter. Each article was included because they are very applied, written in a style that is understandable, and throw additional light onto how algorithms are used to solve real life challenges.

Successfully applying machine learning algorithms requires work, patience, diligence and most importantly experimentation and testing. By working through the numerous application, examples and suggested readings, you will broaden your knowledge, deepen you intuitive understanding and strengthen your practical skill set.

Getting R

R is a free software environment for statistical computing and graphics. It is available for all the major operating systems. Due to the popularity of Windows, examples in this book use the Windows version of R. You can download a copy of R from the R Project for Statistical Computing.

Learning R

This text is not a tutorial on using R. However, as you work through the examples you will no doubt pick up many tips and tricks. If you are new to R, or have not used it in a while, refresh your memory by reading the amazing free tutorials at `http://cran.r-project.org/other-docs.html`. You will be "up to speed" in record time!

Using Packages

If a package mentioned in the text is not installed on your machine you can download it by typing `install.packages("package_name")`. For example, to download the `fpc` package you would type in the R console:

```
install.packages("fpc")
```

Once the package is installed, you must call it. You do this by typing in the R console:

```
require(fpc)
```

The `fpc` package is now ready for use. You only need type this once, at the start of your R session.

NOTE... ✍

If you are using Windows you can easily upgrade to the latest version of R using the `installr` package. Enter the following:

```
install.packages("installr")
installr::updateR()
```

Effective Use of Functions

Functions in R often have multiple parameters. The examples in this text focus primarily on the key parameters required for rapid model development. For information on additional parameters available in a function, type in the R console `?function_name`. For example, to find out about additional parameters in the `prcomp` function, you would type:

```
?prcomp
```

Details of the function and additional parameters will appear in your default web browser. After fitting your model of interest, you are strongly encouraged to experiment with additional parameters.

I have also included the `set.seed` method in the R code samples throughout this text to assist you in reproducing the results exactly as they appear on the page.

NOTE... ✍

Can't remember what you typed two hours ago! Don't worry, neither can I! Provided you are logged into the same R session you simply need to type:

```
history(Inf)
```

It will return your entire history of entered commands for your current session.

Summary

I have found, over and over, that an individual who has exposure to a broad range of modeling tools and applications will run circles around the narrowly focused genius who has only been exposed to the tools of their particular discipline. Knowledge of how to build and apply machine learning models using R will add considerably to your own personal toolkit.

The master painter Vincent van Gough once said

> "*Great things are not done by impulse, but by a series of small things brought together.*"

Now let's get started!

NOTE... ✍

As you use the ideas in this book successfully in your own area of expertise, write and let me know. I'd love to hear from you. Email or visit `www.AusCov.com`.

Suggested Reading

- **Machine Learning for Heart Attack Risk:** Weng SF, Reps J, Kai J, Garibaldi JM, Qureshi N (2017) Can machine-learning improve cardiovascular risk prediction using routine clinical data? PLoS ONE 12(4): e0174944. https://doi.org/10.1371/journal.pone.0174944

- **Machine Learning Overview:**

  - Hooker, Giles, and Cliff Hooker. "Machine Learning and the Future of Realism." arXiv preprint arXiv:1704.04688 (2017).
  - Kavakiotis, Ioannis, et al. "Machine Learning and Data Mining Methods in Diabetes Research." Computational and Structural Biotechnology Journal (2017).
  - Libbrecht, Maxwell W., and William Stafford Noble. "Machine learning applications in genetics and genomics." Nature Reviews Genetics 16.6 (2015): 321-332.
  - Stanisavljevic, Darko, and Michael Spitzer. "A Review of Related Work on Machine Learning in Semiconductor Manufacturing and Assembly Lines." (2016).

Other

R user groups are popping up everywhere. Look for one in you local town or city. Join it! Here are a few resources to get you started:

- For my fellow Londoners check out: `http://www.londonr.org/`.
- A global directory is listed at: `http://blog.revolutionanalytics.com/local-r-groups.html`.

- Another global directory is available at: http://r-users-group.meetup.com/.
- Keep in touch and up to date with useful information in my FREE newsletter. Sign up at www.AusCov.Com.

Chapter 2

Decision Trees

DECISION trees are one of the simplest forms of decision model. They use sample data features to create decision rules that form a treelike structure. Decision trees have their origin in the way humans make decisions. They are popular because they present information visually in an easy to understand tree format.

Decision trees can be used for both regression and classification problems. They reduce a data sample to a manageable set of rules which can be used to make a classification decision or generate a prediction. In this chapter, you will:

- Gain an intuitive overview of how a decision tree works.
- Discover the divide and conquer approach to decision tree construction.
- Explore the role of information, and walk through an applied example of it's calculation.
- Examine how decision trees have been applied in real world applications ranging from the design of intelligent shoes, to predicting traffic accidents.
- Delve into their use for identifying counterfeit currency using R.

Before jumping into the details, let's begin by discussing what decision tree analysis involves.

Understanding Decision Trees

In the 1990 movie *Cadillac Man*, fast-talking used car salesman Joey O'Brien finds himself in a tight spot. The Mafia want their money back, his teenage daughter goes missing, mistress one is madly in love with him, as is mistress two but not her crazy husband, and his ex-wife is on his tail demanding alimony. To make matters worse, his boss tells him he has two days to sell 12 cars or he loses his job!

Suppose you are asked to build an automatic car buying system to replace Joey after he has been fired, pummeled by the Mob or crazy husband, and sentenced to a stint in the "clink" for non-payment of alimony. The goal is to make a "`Yes`" or "`No`" decision as vehicles are presented to you.

Fortunately, from Joey's note book you discover that purchase decisions were made using three features - road tested miles driven, odometer miles, and age of the vehicle in years.

How can you use this information to create a decision rule? One solution, shown in Figure 2.1, is to create a hierarchy of rules, and use these to guide the `Yes`/`No` purchase decision. For example, if the vehicle has less than 100 road tested miles the decision is "`No`". However, if the vehicle has more than 100 road tested miles, and the odometer is less than 50,000 miles, the decision is "`Yes`".

As illustrated in Figure 2.1, a decision tree performs multistage hierarchical decision making. It classifies data by creating a hierarchy of rules about the features. The tree is made up of leaves which contain both a feature and a decision rule. The goal is to generate a hierarchy of decision rules that correctly classify the data.

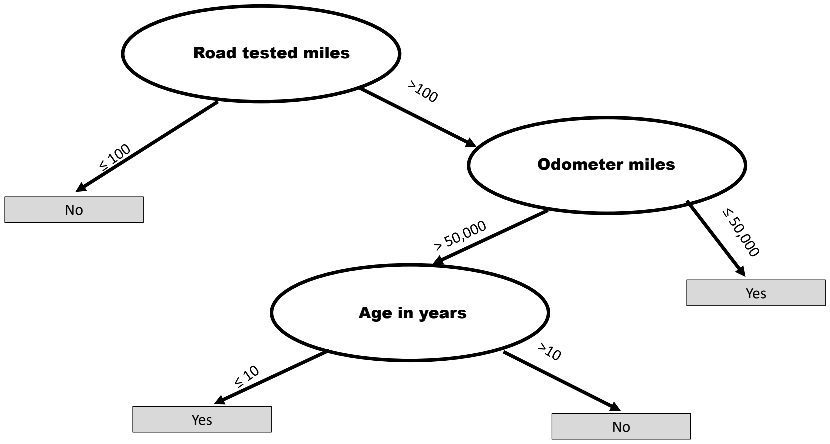


Figure 2.1: Basic decision tree

NOTE... ✍

Decision tree rules are generated by recursively partitioning the data.

Classifying a new example

Once the decision tree has been constructed, classifying a test item is straightforward. True to its name, it selects a class outcome by descending the tree of possible decisions. A new data item is assigned to a class by following the path of rules from the topmost node (root node), to a node without children (leaf). When a leaf node is reached, the class label associated with the leaf node is then assigned to the item.

As an illustration, suppose we are asked to make a decision about a vehicle. The first question, asked at the root node of the tree, is "`How many road tested miles?`" Suppose the vehicle has been tested for 125 miles. We descend down the `>100` branch to the next node. That node asks about odometer miles. Suppose the value is 43,000. Then we descend, down the left branch to the "`Age in years node`". The car is nine years old, so we end up in the `Yes` leaf, and purchase the vehicle.

The Divide and Conquer Algorithm

Decision trees are built by adding features and an associated rule to nodes incrementally. They are typically built from the root downwards using a recursive divide-and-conquer algorithm. As illustrated in Figure 2.2, the basic approach starts at the top node of the tree, where the data is split using a rule derived by one of the features.

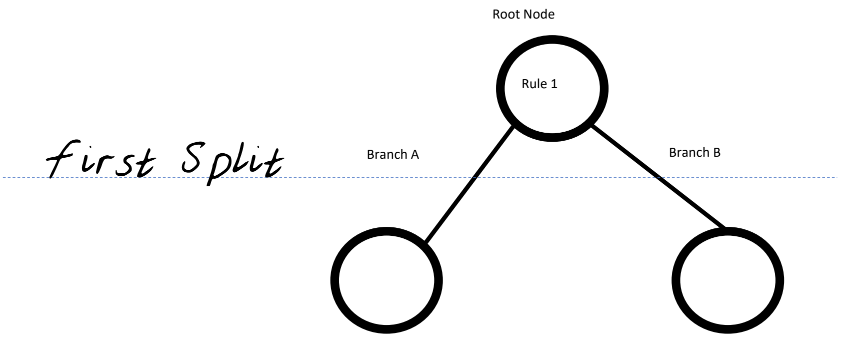


Figure 2.2: Root node and first split of a decision tree

The goal is to choose a feature and rule that splits the data into correctly classified examples. This involves identifying the feature that is the most "useful" for classification, and then deriving a decision rule using the feature.

All features are tested to see which one is the best for splitting the data. Ideally, a single, simple rule would perfectly split the training examples into their classes. However, in most cases, this is not the case, and a rule is selected that separates the examples as cleanly as possible. In Figure 2.2 this is identified as `Rule 1`.

After this stage, the tree consists of two new branches with associated nodes, and one root node. As shown in Figure 2.3, the branch nodes are then assessed. If the partition is "pure" in the sense that all examples in a node belong to the same class then stop. Label each leaf node with the name of the class.

For each branch the process is repeated to build the tree, partitioning the samples to further improve the classification purity of the data. The process continues until a stopping criteria is reached, or the nodes classify all cases in their partition correctly. A typical stopping criteria is to stop splitting the data when new feature-rule combinations increase the purity of the subsets by less than more a small amount.

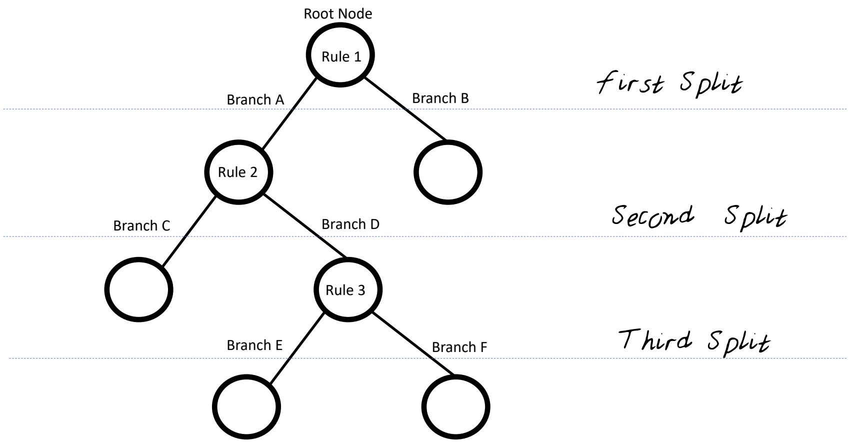


Figure 2.3: Decision tree construction

NOTE... ✍

The Divide and Conquer Algorithm grows a tree by recursively splitting the data into meaningful subsets. Each new rule progressively refines the classification in a hierarchical manner.

Understanding Entropy

Decision tree algorithms require criteria by which to split the data. In practice, different algorithms vary in the metric they choose for this task. A popular metric is based on the entropy of a sample.

Entropy is a measure of randomness in a system. For a probability distribution with k classes, it is defined as:

$$Entropy = \sum_k -p_k \log_2 p_k \tag{2.1}$$

where p_k is the probability an item belongs to the kth class.

NOTE... ✍

The maximum value of entropy depends on the number of classes. In the binary case the maximum occurs at 1. For four classes it occurs at 2, and for 16 classes the maximum is attained at 4.

Entropy for a binary random variable

To better understand entropy, let's take a look at binary classification. The entropy function for a binary random variable **x** is illustrated in Figure 2.4. Notice that:

- It reaches its maximum value of 1 when the probability is 0.5. In other words the state of randomness or uncertainty reaches a maximum when $p(x = 1) = 0.5$, and there a 50-50 chance of either $x = 1$ or $x = 0$.

- The function reaches a minimum at 0 when $p(x = 1) = 1$ or $p(x = 1) = 0$. In other words, when we have complete certainty.

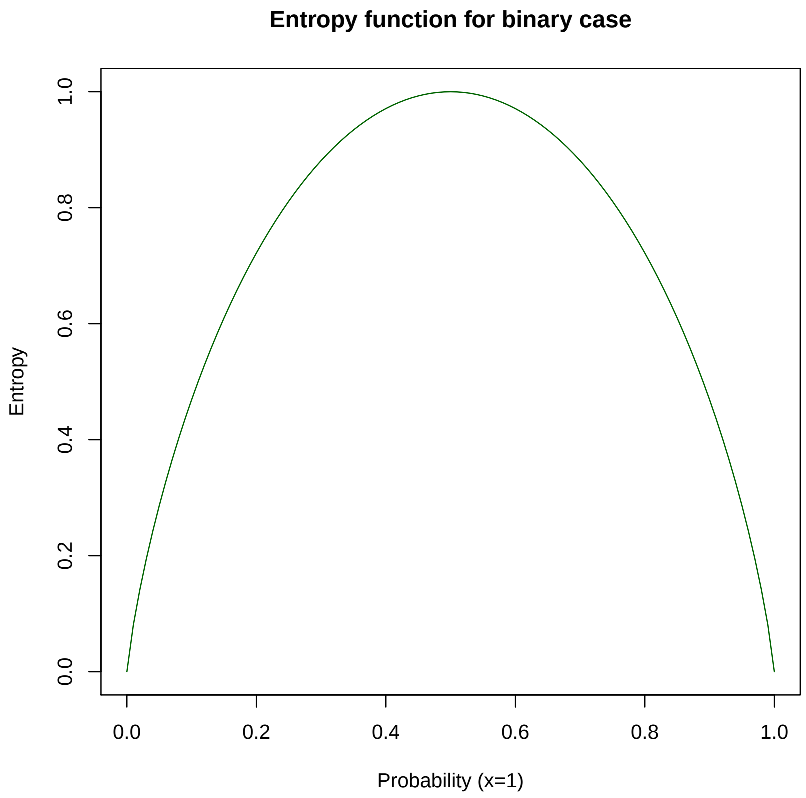


Figure 2.4: Binary entropy function

Importance of Entropy

Entropy is important for decision tree splits because it can be used to calculate the homogeneity (sameness) of a sample. In the case of binary classification, the sample is completely homogeneous when entropy is zero. If there is equal uncertainty about which class an observation belongs to, it has an entropy of one. Therefore, decision trees choose a split for a given partition that minimizes entropy in the partition's distribution of labels

Calculating Entropy - A Step by Step Example

Let us work through a simple example. Suppose you have been asked by Joey O'Brien's ex-boss to build a decision tree to help consumers decide whether to test drive a car. You are given 29 observations as shown in Table 2. It contains three features and the target variable `Test Drive`.

The first vehicle had positive reviews from consumers on the internet, was fuel efficient, and the interior cab exhibited low noise at high speed. It was taken for a test drive. The second vehicle, had the same features as the first vehicle, but was not taken for a test drive.

As illustrated in Table 1, of the 29 cars in the sample, 24 received positive reviews from consumers on the internet, 19 are fuel efficient, and 20 have low interior noise on the motorway.

Feature	Yes	No	Total
Positive Web Review	24	5	29

Feature	Yes	No	Total
Fuel Economy	19	10	29

Feature	Yes	No	Total
Low Noise	20	9	29

Table 1: Summary of counts on the features

Obs	Positive Web Review	Fuel Economy	Low Noise	Test Drive
1	Yes	Yes	Yes	Yes
2	Yes	Yes	Yes	No
3	Yes	Yes	Yes	No
4	Yes	Yes	Yes	Yes
5	Yes	Yes	Yes	No
6	Yes	Yes	Yes	Yes
7	Yes	Yes	Yes	No
8	Yes	Yes	Yes	Yes
9	Yes	Yes	Yes	Yes
10	Yes	Yes	Yes	Yes
11	Yes	Yes	Yes	Yes
12	Yes	Yes	Yes	Yes
13	Yes	Yes	Yes	Yes
14	Yes	Yes	Yes	No
15	Yes	Yes	Yes	Yes
16	Yes	Yes	Yes	Yes
17	Yes	Yes	Yes	Yes
18	Yes	Yes	Yes	Yes
19	Yes	Yes	Yes	No
20	Yes	No	Yes	No
21	Yes	No	No	No
22	Yes	No	No	Yes
23	Yes	No	No	Yes
24	Yes	No	No	No
25	No	No	No	No
26	No	No	No	No
27	No	No	No	No
28	No	No	No	Yes
29	No	No	No	No

Table 2: Test drive data

Step 1: Calculate the Entropy of the sample

The first step is to calculate the entropy for the sample given that our target variable is `Test Drive`. It is calculated using equation 2.1 as:

$$E_{sample} = [-p_{drive}(x = yes)\log_2 p_{drive}(x = yes)]$$

$$- [p_{drive}(x = no)\log_2 p_{drive}(x = no)]$$

The probabilities can be calculated empirically from the sample using the relative frequencies:

$$p_{drive}(x = yes) = \frac{16}{29} = 0.552$$

and

$$p_{drive}(x = no) = \frac{13}{29} = 0.448$$

Therefore:

$$E_{sample} = [-0.552 \times \log_2 \{0.552\}] - [0.448 \times \log_2 (0.448)]$$

$$= 0.473 + 0.519 = 0.992$$

The value equals 0.992, and makes sense because the test drive examples are almost a 50-50 split; so, the entropy should be close to 1.

Step 2: Calculate Entropy of features

The next step is to calculate the weighted entropy given the class for each of the features. After splitting using feature i, a fraction p_i^L of the data goes to the left with entropy $E_i(D_i^L)$; a fraction p_i^R goes to the right node with entropy $E_i(D_i^R)$.

The average entropy after splitting can be calculated as:

$$E_i(D_i^L D_i^R) = p_i^L \times E_i(D_i^L) + p_i^R \times E_i(D_i^R)$$

Notice that $E_i(D_i^L D_i^R)$ is the entropy of feature i, D_i is the sample which has been partitioned into $D_i^L$and D_i^R due to some split decision.

To illustrate this idea, consider the feature `Fuel Economy`. Given the class (`Test Drive`). It is distributed as follows:

		`Fuel Economy`	
		No	Yes
`Test Drive`	No	7	6
	Yes	3	13

The weights p_i^L and p_i^R can be estimated as relative frequencies from the data:

$$p_{fuel}^L(x = no|Test\ Drive) = \frac{7+3}{29} = 0.345$$

and:

$$p_{fuel}^R(x = yes|Test\ Drive) = \frac{6+13}{29} = 0.655$$

Next, we calculate the entropy on each branch of the split:

$$E_{fuel}(D_{fuel}^L) = \left[-\frac{7}{10} \times \log_2 \left\{\frac{7}{10}\right\}\right] - \left[\frac{3}{10} \times \log_2 \left(\frac{3}{10}\right)\right]$$

$$= 0.881$$

and

$$E_{fuel}(D_{fuel}^L) = \left[\frac{13}{19} \times \log_2 \left\{\frac{13}{19}\right\}\right] - \left[\frac{6}{19} \times \log_2 \left(\frac{6}{19}\right)\right]$$

$$= 0.90$$

Therefore:

$$E_{fuel}(D_{fuel}^L D_{fuel}^R) = (0.345 \times 0.881) + (0.655 \times 0.90)$$

$$= 0.893$$

Figure 2.5 visualizes the situation.

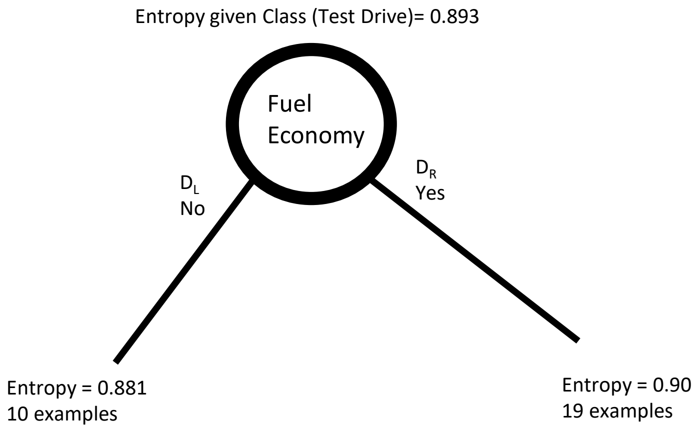

Entropy given Class (Test Drive)= 0.893
Fuel Economy
D_L No
D_R Yes
Entropy = 0.881 10 examples
Entropy = 0.90 19 examples

Figure 2.5: Characteristics of selected root node

Using a similar method we find $E_{web}(D^L_{web}D^R_{web}) = 0.914$, and $E_{noise}(D^L_{noise}D^R_{noise}) = 0.929$. It appears `Low Noise` has the most uncertainty (it is the closest to 1). On the other hand, `Fuel Economy` has the most certainty.

Step 3: Determine the root node using information gain

Information gain is the expected reduction in entropy caused by partitioning the sample according to a given feature. It measures the difference between the entropy of the parent node and the expected entropy of the children. The feature with the highest information gain is chosen as the root node.

For the root node, we can calculate it as the difference between the sample entropy and the entropy of a feature given the class. For the feature `Fuel Economy`, the information gain is:

$$E_{sample} - E_{fuel} = 0.992 - 0.893 = 0.099$$

This tells us the expected reduction in 'uncertainty' if we choose `Fuel Economy` as our root node is 0.099.

For the feature `Positive Web Review`, it is:

$$E_{sample} - E_{web} = 0.992 - 0.914 = 0.078$$

And for the feature `Low Noise`, we have:

$$E_{sample} - E_{noise} = 0.992 - 0.929 = 0.063$$

The information gain (reduction in uncertainty) is greatest for `Fuel Economy`, and so it is selected as the root node.

The process is repeated for subsequent nodes, and the tree stops growing when the information gain is less than a specific threshold, i.e. 0.05.

NOTE... ✍

Decision tree algorithms use a number of impurity measures. Other popular metrics include the gini-index and distance-based metrics.

Advantages of Decision Trees

Decision tress capture every step in the decision-making process. This is because each path through a tree consists of a combination of rules about features which together help distinguish between classes. In other words, the rules provide an explanation for an items classification.

Because a decision tree is built as a series of test questions and conditions, it can be designed to be easy-to-understand, with intuitively clear rules. Large trees can be presented as sets of "`if then`" rules to improve human readability.

Decision trees are a non-parametric method. This means they don't make assumptions about the probability distribution of the attributes or classes. They can handle categorical and numerical data, and are robust to outliers in the sample. In addition, they require very little data preparation. There is no need to normalize the data or create dummy variables; and they can work with missing values.

Decision trees are computationally inexpensive, and work well with large data-sets. Finally, once constructed, they classify new items quickly.

Practical Application of Decision Trees

Decision tree algorithms have been successfully deployed in a wide variety of areas. We look at three interesting applications - intelligent shoes, the analysis of Micro Ribonucleic Acid, and comparison of different decision tree algorithms for analyzing traffic accidents.

Intelligent Shoes

Researchers Zhang et al develop a wearable shoe (`SmartShoe`) to monitor physical activity in stroke patients. The data set consisted of 12 patients who had experienced a stroke.

Supervised by a physical therapist, `SmartShoe` collected data from the patients using eight posture and activity classes. These included sitting, standing, walking, ascending stairs, descending stairs, cycling on a stationary bike, being pushed in a wheelchair, and propelling a wheelchair.

Patients performed each activity from between 1- 3 minutes. Data was collected from the `SmartShoe` every 2 seconds. Features were computed using the collected sample data. Half the feature sample was selected at random for the training set. The remainder used for validation. The C5.0 algorithm was used to build the decision tree.

The researchers constructed decision trees for each individual; and they also built decision trees for the entire group.

The classification performance for five patients are shown in Table 3. For patient 1, the individual tree had a classification accuracy of 96.5%. The accuracy of the group decision tree for the same patient was 87.5%.

As might be expected, the decision trees designed for a specific patient performed better than the tree fitted to the group. For example, the accuracy of patient 5's individual tree was 98.4%, however the group tree accuracy was 75.5%. The group models were trained using data from multiple subjects, and therefore might be expected to have lower overall predictive accuracy than a patient specific decision tree.

Patient	1	2	3	4	5
Individual	96.5%.	97.4%	99.8%	97.2%	98.4
Group	87.5%	91.1%	64.7%	82.2%	75.5

Table 3: Zhang et al's decision tree performance metrics

Micro Ribonucleic Acid

MicroRNAs (miRNAs) are non-protein coding Ribonucleic acids (RNAs) that attenuate protein production in P bodies. Scholars Williams et al develop a MicroRNA decision tree. For the training set the researchers used known miRNAs associated from various plant species for positive controls and non-miRNA sequences as negative controls.

The typical size of their training set consisted of 5294 cases using 29 attributes. The model was validated by calculating sensitivity and specificity based on leave-one-out cross-validation.

After training, the researchers focus on attribute usage information. Table 4 shows the top ten attribute usage for a typical training run. The researchers report that other training runs show similar usage. The values represent the percentage of sequences that required that attribute for classification. Several attributes, such as DuplexEnergy, minMatchPercent, and C content, are required for all sequences to be classified. Note that G% and C% are directly related to the stability of the duplex.

Usage	Attribute
100%	G%
100%	C%
100%	T%
100%	DuplexEnergy
100%	minMatchPercent
100%	DeltaGnorm
100%	G + T
100%	G + C
98%	duplexEnergyNorm
86%	NormEnergyRatio

Table 4: Top ten attribute usage for one training run of the classifier reported by Williams et al.

An interesting question is if all miRNAs in each taxonomic category studied by the researchers are systematically excluded from training while including all others, how well does the predictor do when tested on the excluded category? Table 5 provides the answer. The ability to correctly identify known miRNAs ranged from 78% for the Salicaceae to 100% for seven of the groups shown in Table 5.

Traffic Accidents

Researchers de Oña et al investigate the use of decision tress for analyzing road accidents on rural highways in the province of Granada, Spain. Regression-type generalized linear models, Logit models and Probit models have been the techniques most commonly used to conduct such analyses.

Taxonomic Group	Correctly classified	% of full set excluded
Embryophyta	94%	9.16%
Lycopodiophyta	100%	2.65%
Brassicaceae	100%	20.22%
Caricaceae	100%	0.05%
Euphorbiaceae	100%	0.34%
Fabaceae	100%	27.00%
Salicaceae	78%	3.52%
Solanaceae	93%	0.68%
Vitaceae	94%	4.29%
Rutaceae	100%	0.43%
Panicoideae	95%	8.00%
Poaceae	100%	19.48%
Pooideae	80%	3.18%

Table 5: Results from exclusion of each of the 13 taxonomic groups by Williams et al.

Three decision tree models are developed using three different algorithms (CART, ID3 and C4.5). Nineteen independent variables, reported in Table 6, are used to build the decision trees.

Accident type	Age
Safety barriers	Cause
Lane width	Lighting
Number of injuries	Number of Occupants
Pavement width	Pavement markings
Shoulder type	Sight distance
Atmospheric factors	Day of week
Paved shoulder	Gender
Time	Vehicle type
Month	

Table 6: Variables used from police accident reports by de Oña et al

The accuracy results of their analysis are shown in Table 7. Overall, the decision trees showed modest improvement over chance.

CART	C4.5	ID3
55.87	54.16	52.72

Table 7: Accuracy results (percentage) reported by de Oña et al

Even though de Oña et al tested three different decision tree algorithms (CART, ID3 and C4.5), their results led to a very modest improvement over chance. This will also happen to you on very many occasions. Rather than clinging on to a technique (because it is the latest technique or the one you happen to be most familiar with), the professional seeks out and tests alternative methods. In this text, you have many of the best machine learning techniques at your fingertips. If decision trees don't "cut it" try something else!

Example - Identifying Counterfeit Currency

In Horatio Alger's classic *Timothy Crump's Ward: A Story of American Life*, Abel Harding, a kind-hearted Baker, receives a dollar in payment for goods in his store. He gives it to his own daughter. Delighted with the cash, the child persuades her mother, Mrs. Harding, to take her shopping for a new doll. In the store, Mrs. Harding encounters a slight problem as she attempts to pay with her daughter's dollar:

> The shopman took it in his hand, glanced at it carelessly at first, then scrutinized it with increased attention.
>
> "What is the matter?" inquired Mrs. Harding. "It is good, isn't it?"
>
> "That is what I am doubtful of," was the reply.
>
> "It is new."
>
> "And that is against it. If it were old, it would be more likely to be genuine."
>
> "But you wouldn't condemn a bill because it is new?"
>
> "Certainly not; but the fact is, there have been lately many cases where counterfeit bills have been passed, and I suspect this is one of them. However, I can soon ascertain."
>
> "I wish you would," said the baker's wife. "My husband took it at his shop, and will be likely to take more unless he is put on his guard."

The shopman sent it to the bank where it was pronounced counterfeit.

In the 1860's when Horatio Alger was writing, counterfeit currency was a serious issue. It remains with us today. In this section, we build a decision tree to help identify counterfeit currency.

Step 1 – Collecting and Exploring the Data

The `banknote` data-frame in the `mclust` package contains measurements made on genuine and counterfeit Swiss 1000 franc bank notes. Table 8 provides details of the features, and class variable.

Variable	Description	Type
`Status`	Genuine or counterfeit	Class
`Length`	Length of bill (mm)	Feature
`Left`	Width of left edge (mm)	Feature
`Right`	Width of right edge (mm)	Feature
`Bottom`	Bottom margin width (mm)	Feature
`Top`	Top margin width (mm)	Feature
`Diagonal`	Length of diagonal (mm)	Feature

Table 8: Features and Class in `banknote`

Load the data, and take a look at the first few observations using the `head` function:

```
data("banknote",package="mclust")
head(banknote)
```

```
 Status  Length  Left  Right  Bottom  Top  Diagonal
1 genuine  214.8 131.0 131.1     9.0   9.7     141.0
2 genuine  214.6 129.7 129.7     8.1   9.5     141.7
3 genuine  214.8 129.7 129.7     8.7   9.6     142.2
4 genuine  214.8 129.7 129.6     7.5  10.4     142.0
```

```
5 genuine   215.0 129.6 129.7    10.4   7.7      141.8
6 genuine   215.7 130.8 130.5     9.0  10.1      141.4
```

The first six banknotes are all genuine. The first example has a length of 214.8, and diagonal of 141.0.

Let's take a look at the last few observations via the `tail` function:

```
tail(banknote)
```

```
          Status Length  Left Right Bottom  Top Diagonal
195  counterfeit  214.9 130.3 130.5   11.6 10.6    139.8
196  counterfeit  215.0 130.4 130.3    9.9 12.1    139.6
197  counterfeit  215.1 130.3 129.9   10.3 11.5    139.7
198  counterfeit  214.8 130.3 130.4   10.6 11.1    140.0
199  counterfeit  214.7 130.7 130.8   11.2 11.2    139.4
200  counterfeit  214.3 129.9 129.9   10.2 11.5    139.6
```

The final few observations are all counterfeit. The last example, has a length of 214.3, and a diagonal of 139.6.

Let's use the table function to check on the distribution of the class variable:

```
table(banknote$Status)
```

```
counterfeit     genuine
        100         100
```

As expected the sample consists of 200 examples composed of 100 counterfeit and 100 genuine banknotes.

Figure 2.6 shows a correlation plot of the features; `Left` and `Right` have a relatively high correlation of 0.74; and `Diagonal` is negatively correlated with all the features except `Length`.

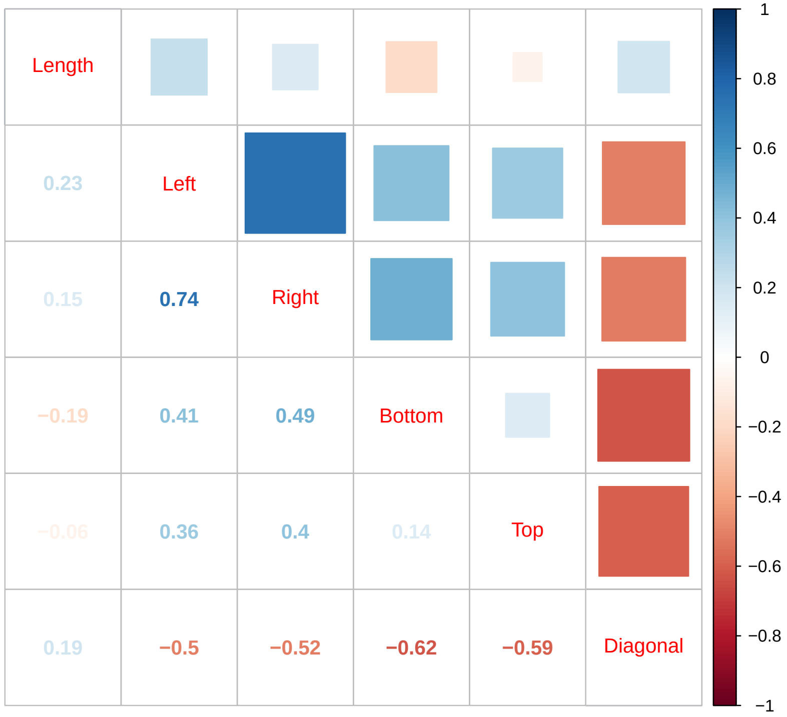


Figure 2.6: Correlation plot of the features

Step 2 – Preparing the Data

The sample consists of measurements made on 100 genuine and 100 counterfeit notes. We select 150 at random without replacement for the training set via the `sample` function. The remaining observations are used for the test set:

```
set.seed(2018)
N=nrow(banknote)
train <- sample(1:N, 150, FALSE)
```

Note that the R object `train` contains the row numbers of the train set data, and not the actual observations. To see this, use the `head` function:

```
head(train)
[1]  68  93  12  39 199  59
```

This informs us that the first randomly selected example is from row 68 of `banknote`. The second randomly selected example is from row 93 of `banknote`, and so on.

Step 3 - Train Model using Train Set

We will use the C5 algorithm to build our decision tree. It uses entropy for splits. The package `C50` contains the decision tree fitting function `C5.0`. Here is how to use it with our data:

```
library(C50)
fitc<- C5.0(Status ~.,
data = banknote[train,] )
```

Here are a few observations on this set up:

- The `C5.0` function takes a standard R formula, followed by the training data.
- The formula tells R to build a model with `Status` as the class variable. The argument "`~.`" instructs R to use all of the features in `banknote`.
- Details of the fitted decision tree are stored in the R object `fitc`.

Viewing the decision tree

We can use the `plot` function to visualize the decision tree:

```
plot(fitc)
```

You should see Figure 2.7. The root node contains the feature `Diagonal`, with the first split occurring at 140.6. The second node is the feature `Bottom`, and the final node is `Length`. Notice the decision tree did not use all of the attributes, only those deemed to be most informative.

The leaf nodes indicate the number of observations, and the proportion correctly classified. For example, `Diagonal>140.6` results in 75 observations in Node 7, all of which appear to be correctly classified.

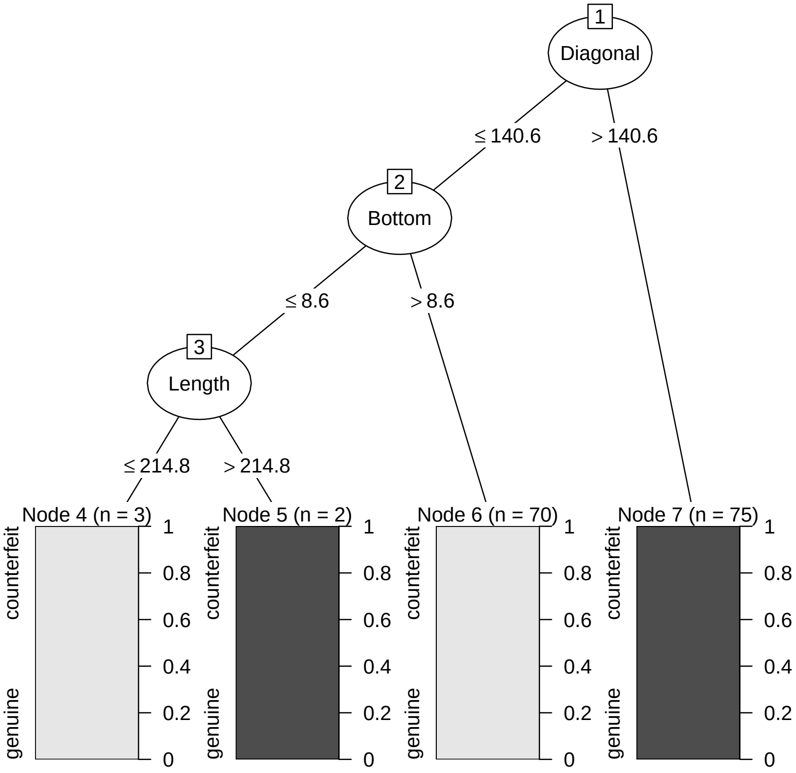


Figure 2.7: Decision tree using `banknote`

View rules

Since a decision tree is built as a series of test questions and conditions, we can view the actual rules as a series of "`if then`" statements. With a large tree, this can improve readability. To do this, simply refit the model via the `C5.0` function with the added argument `rules=TRUE`:

```
fitc_rules<- C5.0(Status ~.,
data = banknote[train,],
rules=TRUE)
```

Now, to see the rules use the `summary` function:

```
summary(fitc_rules)

Rules:

Rule 1: (70, lift 2.0)
        Bottom > 8.6
        Diagonal <= 140.6
        ->  class counterfeit

Rule 2: (44, lift 2.0)
        Length <= 214.8
        Diagonal <= 140.6
        ->  class counterfeit

Rule 3: (75, lift 1.9)
        Diagonal > 140.6
        ->  class genuine

Rule 4: (30, lift 1.9)
        Length > 214.8
        Bottom <= 8.6
        ->  class genuine
```

The rules are now visible, and pretty easy to understand. The first rule states `if Bottom > 8.6 and Diagonal` $\leq$ `140.6` then the classification is `counterfeit`. The fourth rule informs us that `if Bottom` $\leq$ `8.6 and Length > 214.8` then the classification is `genuine`.

Step 4 - Evaluate Model Performance

To explore how well the decision tree classified the training sample we can use a combination of the `predict` and `table` functions. The `predict` function takes the fitted tree as the first argument, followed by the sample data. We set the third argument `type="class"` to return the class labels:

```
predc_train<-predict(fitc,
newdata=banknote[train,],
type = "class")
```

The prediction results are stored in the R object `predc`. Use the `head` function to take a look at the first few values:

```
head(predc_train)
[1] genuine     genuine     genuine
[4] genuine     counterfeit genuine
Levels: counterfeit genuine
```

We see that the first four train set banknotes are classified by the decision tree as `genuine`. However, the fifth note is classified as `counterfeit`.

How well did the decision tree classify the training sample? We can use the `table` function to create a simple confusion matrix:

```
table(banknote$Status[train],
predc_train,dnn=c( "Observed Class",
"Predicted Class" ))


              Predicted Class
Observed Class counterfeit genuine
   counterfeit          73       0
   genuine               0      77
```

Wow! The decision tree classifies all the examples correctly. This is great! But before we get too excited, we have to remember that this is on the training set. Models often fit training set data well, only to crash and burn when applied to the test sample.

Test set performance

Let's see how well the model performed on the training sample. We follow the steps outlined previously. Pass `fitc` (our fitted decision tree model) along with the test set sample to the `predict` function.

```
predc<-predict(fitc,
newdata=banknote[-train,],
type = "class")
```

Notice we specify the test sample using the argument `-train`.

Now use the `table` function to display the confusion matrix:

```
table(banknote$Status[-train],
predc,dnn=c( "Observed Class",
"Predicted Class" ))

                 Predicted Class
Observed Class counterfeit genuine
   counterfeit          24       3
   genuine               0      23
```

The test sample contains 50 observations. The decision tree classifies all but 3 of these correctly. It is interesting to observe it classified all of the genuine notes correctly. But, was fooled by 3 counterfeit notes, classifying them as genuine. Nevertheless, a 94% classification accuracy rate, without having to transform the data to extract "relevant" features, is a nice place from which to begin.

Step 5 - Improving Model Performance

The decision tree `fitc` correctly classified 94% of the test set observations. This is rather high. How can we push the number even higher?

One approach that often works well is to select an alternative splitting criterion. Three impurity measures or splitting criteria that are commonly used in binary decision trees are

Gini impurity, Entropy and Deviance. The `tree` package lets you to use the Deviance or Gini metric. Let's give it a go:

```
library(tree)
fit<- tree(Status ~.,
data = banknote[train,],
split="deviance")
```

The `tree` function is similar to `C5.0`, however it allows you to set the splitting criterion via the `split` argument. The options are `"deviance"` and `"gini"`.

NOTE... ✍

Gini is calculated as $1 - \sum_k p_k^2$

Viewing the decision tree

The decision tree can be visualized via the `plot` and `text` function. Here is how:

```
plot(fit); text(fit)
```

Figure 2.8 shows the tree. For this illustration, by using deviance we end up with a simpler tree structure. As with Figure 2.7, `Diagonal` is selected as the root node. However, only `Bottom` makes the cut as an additional feature in the tree.

However, since both branches of `Bottom` lead to the `counterfeit` classification, the decision tree breaks down into the simple rule of thumb - `If Diagonal <140.65`, the banknote is fake.

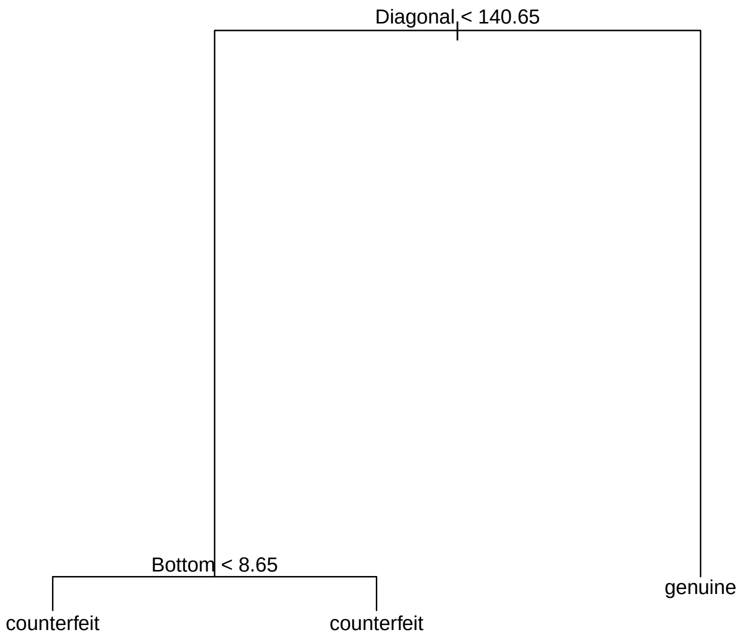


Figure 2.8: Decision tree with deviance as the split criterion

Train set performance

A nice feature of the `tree` function is that classification performance can be viewed using the `summary` function:

```
summary(fit)
```

```
Classification tree:
tree(formula = Status ~ .,
data = banknote[train, ],
split = "deviance")

Variables actually used in tree construction:
```

```
[1] "Diagonal" "Bottom"
Number of terminal nodes:  3
Residual mean deviance:  0.04578 = 6.73 / 147
Misclassification error rate: 0.01333 = 2 / 150
```

The first few lines remind you of the formula used to fit the tree. This is followed by the details of the variables used in tree construction, and the number of leaf (terminal) nodes. The model has a residual deviance of 0.045, with a misclassification error rate of just over 1.33%. In other word 2 of the 150 training examples were misclassified.

Test set performance

To see the test set performance, pass the `predict` function the fitted model `fit`, and the test set data as follows:

```
pred<-predict(fit,
newdata=banknote[-train,])
```

Take a look at the last five predictions:

```
tail(pred,5)
    counterfeit genuine
178         1.0     0.0
187         0.6     0.4
190         1.0     0.0
195         1.0     0.0
```

The row numbers refer to the original row number in the `banknote` dataframe. So, the first observation was item 178 in the original `banknote` dataframe. The columns report probabilities. So, according to the decision tree, the first observation in the list above has a 100% probability of being counterfeit. The second observation has a 60% probability. In both cases, we would classify the observations as counterfeit.

Now let's calculate the confusion matrix. For each row we take the column name of the predicted class, and store the result in the R object `pred.class`:

```
pred.class <- colnames(pred)[max.col(pred,
ties.method = c("random"))]
```

Take a look at the first few predictions, to ensure they match with our previous observations:

```
tail(pred.class,5)
[1] "counterfeit" "counterfeit" "
   counterfeit" "counterfeit" "counterfeit"
```

Yep, as expected the last few observations are classified as `counterfeit`.

Next, we use the `table` function to create the confusion matrix:

```
table(banknote$Status[-train],
pred.class,
dnn=c( "Observed Class",
"Predicted Class" ))


                Predicted Class
Observed Class counterfeit genuine
   counterfeit          27       0
   genuine               0      23
```

The model correctly classifies all of the test set examples. The decision tree has achieved a classification accuracy of 100%.

In this illustration, the use of an alternative split criteria improved the test set performance of the decision tree. However, at the outset of the model building process, there is little guidance on which split method to use. The best you can do is to experiment to see what works best for your data.

Limitations of Decision Trees

Although decision trees have proven to be a robust and powerful machine learning tool, they have several limitations. One of the key being instability. If the sample data is changed slightly, the decision tree can change a lot. This hampers interpretation.

Overfitting

Another weakness of decision trees is they are prone to over fitting. This occurs when statistically insignificant patterns end up influencing classification results. This is most likely to occur when the data contain a small number of examples, or if the data is very noisy.

A while back researchers M. Bohanec and I. Bratko studied the role of pruning a decision tree for better decision making. They found that pruning can reduce the risk of over fitting because it results in smaller decision trees that exploit fewer features. Pruning a fully developed tree using a statistical test of redundancy may reduce the likelihood of overfitting.

Another way to reduce the likelihood of overfitting, is to stop growing the tree before it perfectly classifies the training data.

Simplicity of the Decision Boundary

Traditional decision trees create a decision rule one attribute at a time. The decision rule partitions the sample via a series of axis parallel splits into rectangular regions as shown in Figure 2.9. This limits the complexity of the decision boundary that can be constructed.

Oblique decision trees replace single feature axis parallel splits by creating rules based on a weighted combination of features. They can be used to produce a richer set of decision boundaries.

Figure 2.10, illustrates an oblique decision tree to classify a given vehicle silhouette as one of four types of vehicle (van, bus, opel, saab). The root node rule is a combination of three features `Max.L.Ra`, `Sc.Var.Maxis` and `Elong`. The decision rule is:

$$6.4 - 1.42 \times Max.L.Ra + 0.03 \times Sc.Var.Maxis - 0.12 \times Elong < 0$$

It is clearly a more complex decision rule than that produced by an axis parallel split tree.

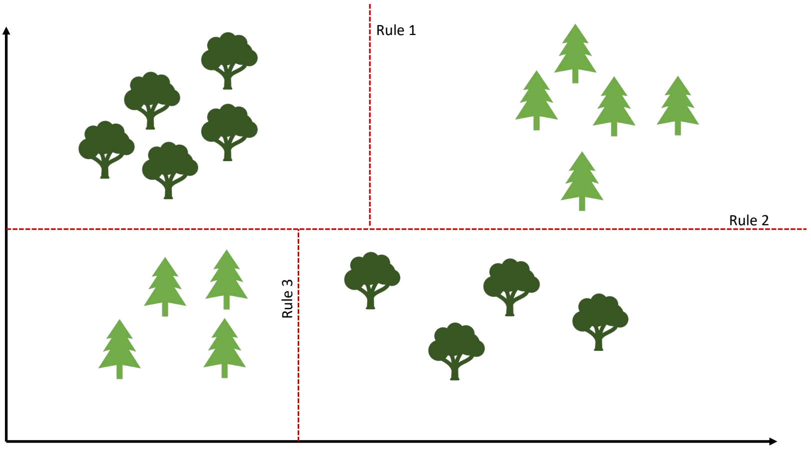


Figure 2.9: Axis parallel splits of a traditional decision tree

NOTE... ✍

Oblique decision trees can be estimated via the `oblique.tree` function in the `oblique.tree` package.

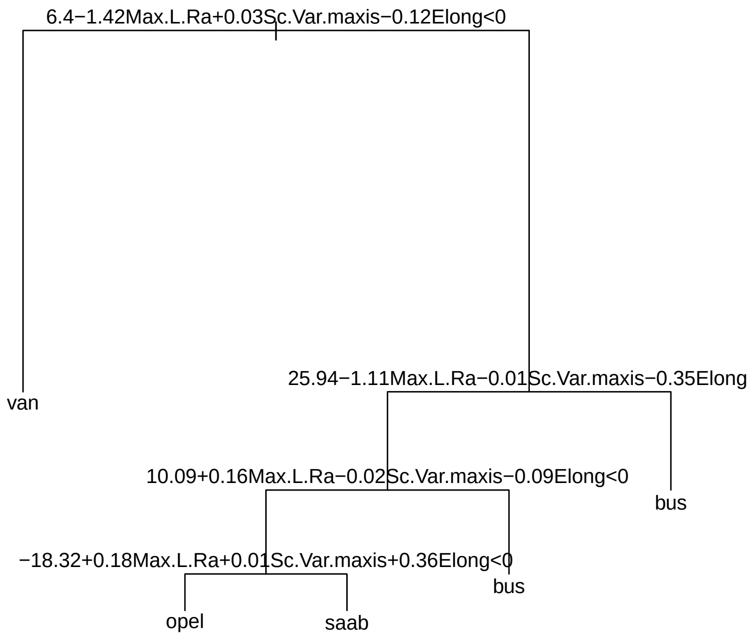

6.4−1.42Max.L.Ra+0.03Sc.Var.maxis−0.12Elong<0
van
25.94−1.11Max.L.Ra−0.01Sc.Var.maxis−0.35Elong
10.09+0.16Max.L.Ra−0.02Sc.Var.maxis−0.09Elong<0
bus
−18.32+0.18Max.L.Ra+0.01Sc.Var.maxis+0.36Elong<0
bus
opel
saab

Figure 2.10: Oblique decision tree

Local Optimality

As illustrated in Figure 2.9, decision trees use recursive partitioning in a forward stepwise search. This results in the best possible choice at each node. However, it does not take into consideration whether those choices remain optimal in later stages. In other words, a traditional decision tree makes a locally optimal decision at each node rather than finding the globally optimal tree.

Although the approach is known to be an efficient heuristic, it is possible that a rule developed at an earlier node may be sub-optimal in the overall scheme of things.

It turns out that finding the globally "optimal tree", if one exists, is computationally intractable (or NP-hard, technically speaking). One solution is to construct decision trees via an approximately globally optimal method. Evolutionary algorithms offer one alternative way to do this.

NOTE... ✍

A decision tree using an evolutionary algorithm can be built via the `evtree` package.

Summary

In this chapter, we discovered one of the most popular tools in machine learning - the decision tree. It is a particularly useful tool for discovering rules, and is especially valuable if the rules are required to be interpretable by humans.

We explored how decision trees have been used in practice and built our own models in R. We also examined the impact of different split metrics and assessed their impact on overall performance.

In the next chapter, we discuss another very simple, yet powerful classification algorithm, k-Nearest Neighbors.

Suggested Reading

- **Intelligent Shoes:** Zhang, Ting, et al. "Using decision trees to measure activities in people with stroke." Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE. IEEE, 2013.

- **Micro Ribonucleic Acid:** Williams, Philip H., Rod Eyles, and George Weiller. "Plant MicroRNA prediction by supervised machine learning using C5.0 decision trees." Journal of nucleic acids 2012 (2012).

- **Traffic Accidents:** de Oña, Juan, Griselda López, and Joaquín Abellán. "Extracting decision rules from police accident reports through decision trees." Accident Analysis & Prevention 50 (2013): 1151-1160.

Other

- **Distance Based Measures:** R. L. De M´antaras, "A distance-based attribute selection measure for decision tree induction," Mach. Learn., vol. 6, no. 1, pp. 81–92, 1991.

- **Information Gain/ Gain Ratio:**
  - J. R. Quinlan, C4.5: programs for machine learning. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.
  - Quinlan, K., and Ron Kohavi. "Decision Tree Discovery." Data Mining, 1999.

- **Gini Index**: Breiman, Leo, et al. Classification and regression trees. CRC press, 1984.

- **Pruning:** Bohanec M, Bratko I (1994) Trading accuracy for simplicity in decision trees, Machine Learning 15: 223–250.

Chapter 3

k-Nearest Neighbors

The k-Nearest Neighbors algorithm(KNN) is one of those machine learning algorithms that is very simple to understand and works incredibly well in practice. It is a non-parametric algorithm. Non-parametric simple means that the algorithm makes no assumptions about the probability distribution generating the sample data. This is different from many of the other algorithms we discuss in this book. For example, linear regression assumes the model error is Gaussian. Real world data tends to violate theoretical assumptions, and therefore non-parametric algorithms like KNN come in handy.

In this chapter, you will

- Gain clarity on how KNN works, in theory and practice works.
- Learn about distance metrics and how to use them.
- Walk through a step by step example of KNN calculations.
- Delve into their use for classifying animal and human behavior.
- Build a KNN model to identify wine cultivars.

We begin by outlining the basic idea behind KNN.

Understanding Nearest Neighbors

The basic idea is illustrated in Figure 3.1. We would like to predict the class of the empty oval. One simple way to do this is to look around for objects that are close to the empty oval, and choose the most frequent as the predicted class. This is precisely what KNN does.

KNN uses a local neighborhood to obtain a classification prediction. The assumption is that items in this neighborhood are likely to be similar to the empty oval. It searches for items closest to the empty oval. The size of the local neighborhood to be searched is determined by the parameter k.

Suppose we set k = 3. This tells the algorithm to use the 3 closet items to the empty oval. In Figure 3.1 the three nearest neighbors are all solid ovals. All three are from the same class, therefore the prediction is solid oval.

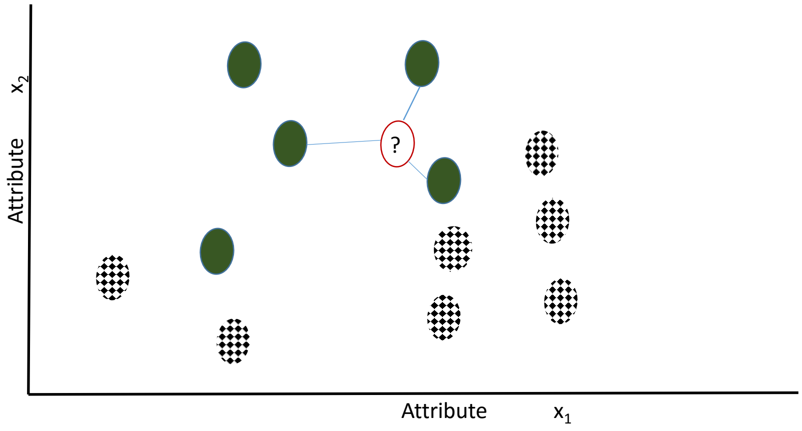


Figure 3.1: Simple illustration of KNN

Measuring Closeness with Distance Metrics

KNN calculates the distance between the points that require classification and its neighbors. The predicted class is then determined by a majority vote. How can we measure the "*closeness*" of a neighbor?

The distance between observations i and j can be measured in many ways. Euclidean distance is often used for continuous features. It is calculated as:

$$\mathcal{D}(x_i, x_j) = \sqrt{\sum_{m=1}^{n} (x_{im} - x_{jm})^2} \tag{3.1}$$

where n is the number of features. Other popular metrics include the Manhattan distance:

$$\mathcal{D}(x_i, x_j) = \sum_{m=1}^{n} |x_{im} - x_{jm}|; \tag{3.2}$$

The Minkowski metric:

$$\mathcal{D}(x_i, x_j) = \left(\sum_{m=1}^{n} (|x_{im} - x_{jm}|)^q \right)^{\frac{1}{q}};$$

And the Canberra metric:

$$\mathcal{D}(x_i, x_j) = \sum_{m=1}^{n} \frac{|x_{im} - x_{jm}|}{|x_{im}| + |x_{jm}|} \tag{3.3}$$

Both Euclidean distance and the Manhattan distance can be regarded as special cases of Minkowski distance. The Euclidean distance results from the selection $q = 2$, the Manhattan distance for the parameter value $q = 1$. The Canberra distance is a weighted version of the Manhattan distance.

NOTE... ✍

If you change the distance function, you change how data points are classified.

An Example to Boost Your Understanding

Abalone are, mostly sedentary, marine snails and belong to a group of sea critters (phylum Mollusca) which include clams, scallops, sea slugs, octopuses and squid. They cling to rocks while waiting for their favorite food of kelp to drift by. Once the seaweed is spotted they hold it down, with what can only be described as a “foot”, and chew on it vigorously (for a snail) with radula - a rough tongue with many small teeth.

Given the healthy diet of fresh kelp, it is little wonder these mollusks are considered delicious raw with a little lemon, or gently pan-fried tossed in garlic and butter. They are deemed to be so flavorful that the demand for consumption outstripped supply over many years.

Today, the white abalone is officially listed as an endangered species. Other species remain plentiful, are harvested regularly to the delight of foodies, restaurant goers and seafood chefs.

Tasmania supplies around a quarter of the yearly world supply of abalone. The majority of which are either blacklip or greenlip abalone. Let’s walk through KNN for classification of blacklip and greenlip abalone.

Suppose we are given some standardized data points on age, weight and type of abalone, as illustrated in Figure 3.2. Our goal is to find the class label for the new point denoted with a question mark.

Sample observations

The sample observations are shown in Table 9. A total of eleven labeled observations are available, and our task is to predict the class of the 12th observation, which has a standardized age of 0.807 and a standardized weight of -0.963.

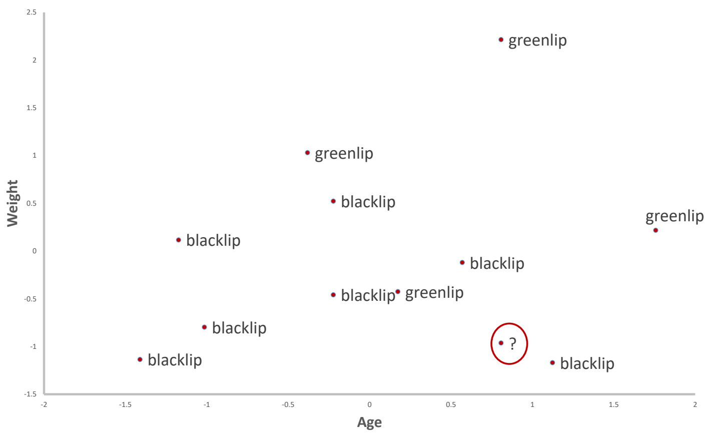
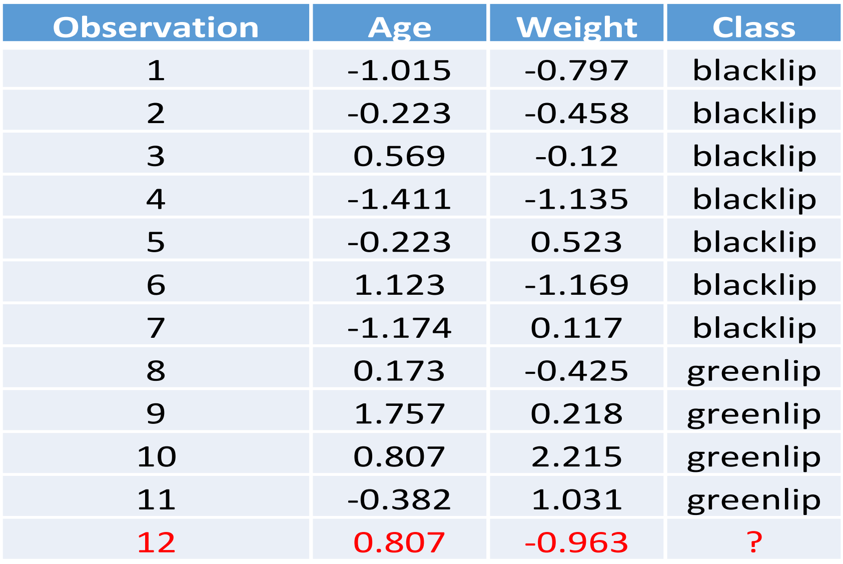


Figure 3.2: Abalone illustration



Observation	Age	Weight	Class
1	-1.015	-0.797	blacklip
2	-0.223	-0.458	blacklip
3	0.569	-0.12	blacklip
4	-1.411	-1.135	blacklip
5	-0.223	0.523	blacklip
6	1.123	-1.169	blacklip
7	-1.174	0.117	blacklip
8	0.173	-0.425	greenlip
9	1.757	0.218	greenlip
10	0.807	2.215	greenlip
11	-0.382	1.031	greenlip
12	0.807	-0.963	?

Table 9: Attribute and class for each observation

Calculate distance

To illustrate the calculation, I use Euclidean distance. So for example, the distance between new observation x_{12} and the 11th observation x_{11} is calculated as:

$$\mathcal{D}(x_{12}, x_{11}) = \sqrt{(x_{12\,1} - x_{11\,1})^2 + (x_{12\,2} - x_{11\,2})^2}$$

$$= \sqrt{(0.807 - (-0.382))^2 + (-0.963 - 1.031)^2}$$

$$= \sqrt{1.414 + 3.976}$$

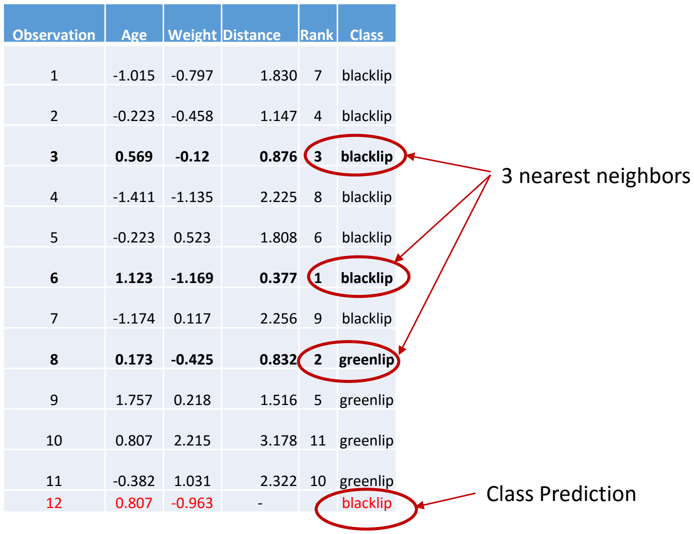
$$= \sqrt{5.390}$$

$$= 2.322$$

Classification prediction

Table 10 shows the distances for all eleven observations where the closest observation has a rank of 1.

Let's set k= 3, to use the three closest observations. Doing this, we see the nearest neighbor is observation 6 which is classified as blacklip, followed by observation 8 which is classified as greenlip, followed by observation 3 which is classified as blacklip. Using these three neighbors and majority voting, the new observation is classified as blacklip.



Observation	Age	Weight	Distance	Rank	Class
1	-1.015	-0.797	1.830	7	blacklip
2	-0.223	-0.458	1.147	4	blacklip
3	**0.569**	**-0.12**	**0.876**	**3**	**blacklip**
4	-1.411	-1.135	2.225	8	blacklip
5	-0.223	0.523	1.808	6	blacklip
6	**1.123**	**-1.169**	**0.377**	**1**	**blacklip**
7	-1.174	0.117	2.256	9	blacklip
8	**0.173**	**-0.425**	**0.832**	**2**	**greenlip**
9	1.757	0.218	1.516	5	greenlip
10	0.807	2.215	3.178	11	greenlip
11	-0.382	1.031	2.322	10	greenlip
12	0.807	-0.963	-		blacklip

3 nearest neighbors
Class Prediction

Table 10: Euclidean distances and rank by closeness

Advantages of k-Nearest Neighbors

KNN is very simple to understand. It is easy to implement. Because the process is transparent, it is also very easy to explain to professionals not versed in machine learning or mathematics.

It is also an instance based learning algorithm. Instance based learning algorithms memorize the sample data. There is no explicit training or model. Any generalization beyond the training sample is delayed until a query is made to the system by an new instance (data sample).

Practical Application of k-Nearest Neighbors

Figuring out species type is a common problem in science. KNN has been used successfully for this task. We outline the approach a group of researchers used to classify plant species.

Whether you prefer to watch other humans, or the animals. Human and animal behavior is fascinating. We also discuss how KNN has been used to successfully identify behavior.

Leaf Classification

In Lisa D. Lee's compelling *Mystery of the Jaguar Moon*, an investigative team of scientists explore an ancient Mayan temple located in the Yucatan Peninsula of Mexico. Federico, one of the team members, takes care to document the plant life:

> "*Federico diligently photographed and made detailed drawing of each leaf, flower, root...Most were depicted with a dizzying array of color and intricacy. They were by far more intriguing than the digital photos because of the human interpretation and character inherent in them.*"

Leaf morphology is useful for species identification, that is why Federico took such painstaking care. Of course, if the team of scientists had included a Statistician, it might have been suggested that KNN help assist in the task.

In the real world, researchers at Xiamen University in China, collected 350 images from seven plant species. KNN was one of the machine learning algorithms used for classification. The researchers extracted four features based on leaf tooth morphology.

The training sample consisted of half of the images selected at random without replacement. The remaining images formed the test sample.

Classification of the KNN model was performed using the two nearest neighbors (k=2). The researchers observed a classification accuracy of 72.3%. This was not statistically different from the classification performance of a much more complex neural network.

Classifying Human Activity

A triaxial accelerometer is an electromechanical device that measures acceleration forces across three axis. They can be used to detect activity such as running and walking. Alessio Martinelli, Simone Morosi, and Enrico Del Re use accelerometer data to classify movements such as walking, walking up stairs and walking down stairs.

The sample data was collected from 10 individuals wearing a belt mounted smart phone that generated triaxial accelerometer signals.

The signals were processed via a Fast Fourier Transform and low-pass filtering to extract 24 features. Three KNN models were tested (k=1, k=10 and k=100) with the goal of classifying walking (`walk`), walking up-stairs (`up`), walking down-stairs(`down`).

Performance was assessed using 10-fold cross validation. The confusion matrix of the best performing KNN is shown below:

		Predicted Class		
		`walk`	`up`	`down`
Actual Class	`walk`	**94.6%**	4.6%	0.7%
	`up`	7.7%	**89.7%**	2.6%
	`down`	5.7%	9.9%	**84.4%**

The class specific accuracy is highest (94.6%) for walking; and the KNN model has an average classification accuracy of 89.5%. The researchers comment:

> "*...the KNN model has obtained very good accuracy results in classifying the testing movements.*"

Identifying Animal Behavior

One of the neat things about KNN is that it often works well with very little, if any data preprocessing. Owen Bidder from the College of Science at Swansea University in Wales, collected raw triaxial accelerometer data. He then used KNN to predict behavior from this data:

> "*The purpose of this study was to illustrate that the KNN method could be used to identify automatically the behavioural modes of animals equipped with accelerometers recording at high sample rates, and that this approach is applicable for large, complex datasets.*"

Accelerometer data was collected on various animals and humans engaged in motion based activities. For example, the human accelerometer data consisted of four classes - `Lying, Walk, Run`, and `Crawl`.

Each sample consisted of training and testing segments equivalent to 1000 and 2000 data points respectively. The raw data for all three axis of acceleration were fed into the KNN algorithm, and it's classification performance assessed. Twenty-one nearest neighbors (k=21) were used in the KNN algorithm.

The classification accuracy of the model is given in Table 11.

Item	Accuracy %
Badger	0.71
Camel	0.82
Comorantl	0.77
Cheetah	0.77
Dingo	0.83
Kangaroo	0.91
Wombat	0.76
Human	0.95

Table 11: KNN classification result

There is some variability in performance, with accuracy highest in humans and lowest in the badger. Nevertheless, this level of performance derived directly using raw data is very encouraging. Owen Bidder and his co-researchers conclude:

> "*Our results show that animal behavioural modes can indeed be successfully identified automatically using the KNN method and that, with a mean Accuracy of 78%, they are comparable to results gained using more complex automated methods.*"

Example - Identifying Wine Cultivars

The quality of a fine wine is a complex combination of many individual constituents, some originating in the grapes, and others produced during fermentation. Certain individuals possess an unnatural ability to identify a fine wine and its vineyard. Take for example, Richard Pratt, the priggish middle aged potbellied epicurean, in Roald Dahl's *Taste*.

At an upscale dinner party, Mike Schofield, a working-class man, who has made some money on the stock market, makes a high-risk bet. He has purchased a fine wine from a little-known vineyard, and challenges Richard Pratt, the famous gourmet, to identify the source.

The stakes are high, if Mike wins he acquires two houses; If he loses, his daughter becomes Richard Pratt's wife.

The room goes quiet. Total silence. Everyone watching Richard. He pauses, picks up the wine glass, examines the color, sniffs the aroma, tastes the wine, closes his eyes. Then slowly, very slowly a sly smile creeps across his face...

In this section, we develop a KNN model to classify wine cultivars. The hope is that such systems can outperform the Richard Pratt's of this world.

Step 1 – Collecting and Exploring the Data

The `wine` dataset from the `rebmix` package contains 13 features derived from chemical analyses of three types of wine from the same region in Italy, see Table 12.

Item	Description	Type
`Alcohol`	continuous.	Feature
`Malic.Acid`	continuous.	Feature
`Ash`	continuous.	Feature
`Alcalinity.of.Ash`	continuous.	Feature
`Magnesium`	continuous.	Feature
`Total.Phenols`	continuous.	Feature
`Flavanoids`	continuous.	Feature
`Nonflavanoid.Phenols`	continuous.	Feature
`Proanthocyanins`	continuous.	Feature
`Color.Intensity`	continuous.	Feature
`Hue`	continuous.	Feature
`OD280.OD315.of.Diluted.Wines`	continuous.	Feature
`Proline`	integer.	Feature
`Cultivar`	1, 2 or 3	Class

Table 12: The `wine` dataset

The class variable represents three different cultivars. The sample contains 178 examples:

```
data("wine",package ="rebmix")
```

Figure 3.3 shows a bar-plot for the class variable `wine$Cultivar`. Clearly, the sample is somewhat unbalanced. However, the class distributions are not extreme, and will likely not cause an issue in our analysis.

Figure 3.4 presents the correlation plot for the attributes. The pairwise correlations range from strongly positive to mildly negative.

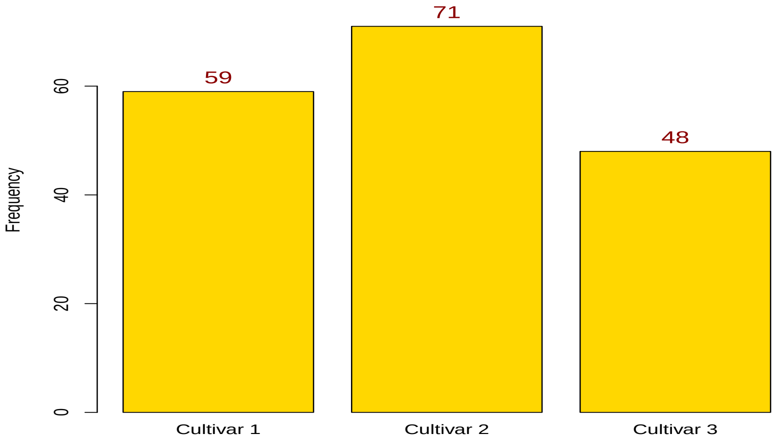


Figure 3.3: Bar-plot of target variable - `wine$Cultivar`

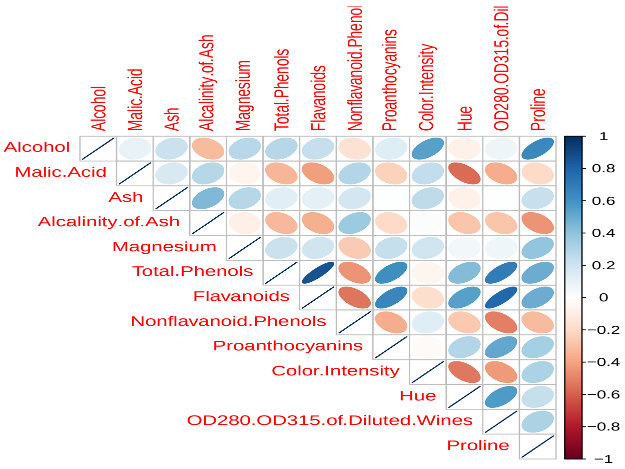


Figure 3.4: Correlation plot of `wine` features

Step 2 – Preparing the Data

Earlier we saw that Owen Bidder successfully performed KNN classification using features obtained from raw triaxial accelerometer data. The performance of KNN depends critically on the distance metric. For this reason, you should typically normalize or scale the features so that they are approximately in the same range. If you don't do this, features with larger ranges will be treated as more important.

Figure 3.5, illustrates the situation for the `wine` features. Clearly, `Proline` has a much larger range than the other features, and therefore dominates the box-plot.

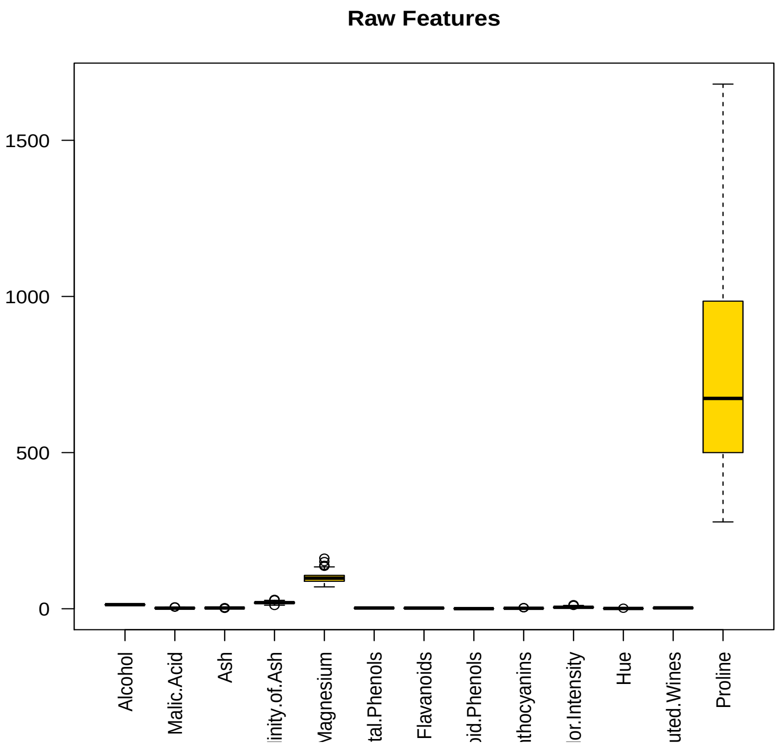


Figure 3.5: Box-plot of raw `wine` features

Let's transfer the features to the R object `data_sample`. We can scale the features to have a mean of zero and a standard deviation of 1, using the `scale` function:

```
data_sample<-wine[,1:13]
data_sample<-scale(data_sample)
```

Figure 3.6 shows the box-plot of the scaled features. They appear more balanced, with no single feature dominating the plot.

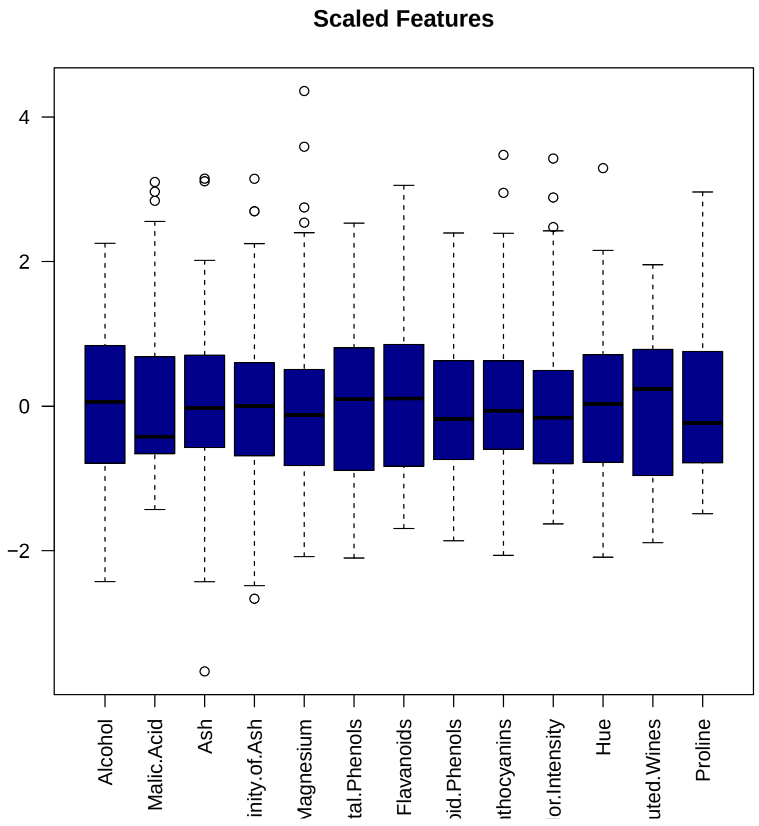


Figure 3.6: Box-plot of scaled `wine` features

NOTE... ✍

Irrelevant and noisy features should be removed from the sample. They add random variations to the distance measures, and therefore are likely to hamper classification performance.

Train and test samples

A random sample, without replacement, can be specified via the `sample` function, and setting the `replace` argument to `FALSE`. We will select 89 observations without replacement for our training sample:

```
set.seed(2016)
n=nrow(data_sample)
train <- sample(1:n, 89, replace = FALSE)
```

The first line uses the `set.seed` function for reproducibility. The object `n` contains the total number of observations in the sample (178); and `train` contains the row numbers of the randomly selected training sample.

To view the first few values in `train` use the `head` function:

```
head(train)
[1]  33  26 149  24  84  21
```

The first randomly selected example is row 33 of `wine`. The second, from row 26, and so on.

Step 3 - Train Model using Train Set

There are numerous options for estimating a KNN model in R. We use the `knnVCN` function from the `knnGarden` package. This package allows us to specify various distances metrics. Here is how to fit a KNN using the nearest two neighbors:

```
require(knnGarden)
fit1<-knnVCN(data_sample[train,],
wine$Cultivar[train],
data_sample[-train,],
K = 2,
method = "canberra")
```

Let's take a moment to run through this code. The first argument of `knnVCN` receives the train set features. This is followed by the class variable. The parameter `K` controls the number of neighbors used in the analysis. We set `K=2`. The final line specifies the distance metric, we use Canberra distance given in equation 3.3. The R object `fit1` contains the fitted model.

Step 4 - Evaluate Model Performance

The predicted values for the test set data are stored in `fit1$TstXIBelong`. We use these values alongside a confusion matrix to evaluate performance. The `table` function helps out here:

```
tab1<-table(fit1$TstXIBelong,
wine$Cultivar[-train])
tab1
      1  2  3
  1 27  2  0
  2  0 34  3
  3  0  1 22
```

The confusion matrix informs us that the KNN algorithm classifies the majority of cases correctly. The error rate is:

$$Error\ rate = \frac{2+3+1}{89} = 6.74\%$$

And the overall classification accuracy using the test set data is:

$$Accuracy = 1 - error\ rate = 93.26\%$$

Step 5 - Improving Model Performance

Our initial model has a high classification accuracy. How might we improve it further? One solution is to try an alternative distance matrix. At the outset of a study using KNN, it is unclear which distance metric will deliver the optimal performance. Unfortunately, there are no reliable rules of thumb to follow, so the best path is usually experimentation.

For illustration, we the fit the exact same model as `fit1`, except with Euclidean distance:

```
fit2<-knnVCN(data_sample[train,],
wine$Cultivar[train],
data_sample[-train,],
K = 2,
method = "euclidean")

tab2<-table(fit2$TstXIBelong,
wine$Cultivar[-train])

tab2
```

```
     1  2  3
  1 27  3  0
  2  0 34  1
  3  0  0 24
```

In the case, the model misclassified 4 observations on the test set sample. The overall test set accuracy is a little higher at 95.5%

Changing k

The number of neighbors is a key parameter in the KNN algorithm. Its optimal value is often determined by experimentation. Suppose we take `fit2` and use the three nearest neighbors. What impact will this have on test set performance? Let's find out:

```
fit3<-knnVCN(data_sample[train,],
wine$Cultivar[train],
data_sample[-train,],
K = 3,
method = "euclidean")
tab3<-table(fit3$TstXIBelong,
wine$Cultivar[-train])

tab3
```

```
     1  2  3
  1 27  2  0
  2  0 35  1
  3  0  0 24
```

The misclassification rate has fallen slightly, with only 3 observations misclassified. The overall accuracy is now 96.6%.

We conclude this section by observing that the value of `k`, and the selected distance metric can have a significant impact on overall performance of the KNN algorithm.

Limitations of k-Nearest Neighbors

For very large datasets real time predictive performance can be relatively slow. Furthermore, KNN is sensitive to irrelevant or redundant features because all features contribute to the distance calculation and therefore the classification prediction.

At the outset of a study it is unclear which distance metric will deliver the best performance. Nor is it obvious what value to set for the parameter k. In both cases, as with many machine learning techniques, experimentation is required.

Summary

In this chapter, we gained an in depth understanding of KNN. We discovered it is an intuitive, easy to understand and explain, machine learning algorithm. It is a particularly useful tool for classifying unknown observations, and can deliver very good prediction accuracy.

We explored how KNN models have been used to classify both animal behavior, and plant species. We also examined, via R, the impact of different distances metrics on model performance. The number of neighbors also directly impacted performance.

In the next chapter, we discuss a very powerful probabilistic based machine learning algorithm - the naive Bayes classifier.

Suggested Reading

- **Classifying Human Activity:** Alessio Martinelli, Simone Morosi, and Enrico Del Re, "Daily Living Movement Recognition for Pedestrian Dead Reckoning Applications," Mobile Information Systems, vol. 2016, Article ID 7128201, 13 pages, 2016. doi:10.1155/2016/7128201

- **Identifying Animal Behavior:** Bidder OR, Campbell HA, Gómez-Laich A, Urgé P, Walker J, Cai Y, et al. (2014) Love Thy Neighbour: Automatic Animal Behavioural Classification of Acceleration Data Using the K-Nearest Neighbour Algorithm. PLoS ONE 9(2): e88609. https://doi.org/10.1371/journal.pone.0088609

- **Leaf Classification:** Jin T, Hou X, Li P, Zhou F (2015) A Novel Method of Automatic Plant Species Identification Using Sparse Representation of Leaf Tooth Features. PLoS ONE 10(10): e0139482. https://doi.org/10.1371/journal.pone.0139482

Other

Source of Leaf Variation: Andres, Ryan J., et al. "Modifications to a LATE MERISTEM IDENTITY1 gene are responsible for the major leaf shapes of Upland cotton (Gossypium hirsutum L.)." Proceedings of the National Academy of Sciences 114.1 (2017): E57-E66.

Chapter 4

Naive Bayes Classifier

The Naive Bayes Classifier (NBC) is a simple probabilistic classifier based on Bayes theorem. It is especially well suited for situations where there are a large number of attributes or features.

In this chapter, we delve into the Naive Bayes Classifier. You will:

- Learn about the two main types of supervised classification.
- Refresh your memory of Bayes rule, and explore the notion of a Bayes classifier.
- Discover the difference between a generative and a discriminant classifier.
- Work through a step by step example to build your Intuition.
- Review three real world applications of NBC use.
- Use R to investigate NBC on two very different data sets

NBC has been applied since the 1950's, with roots stretching back to the 18th century work of Thomas Bayes. It continues to be a simple, powerful and often, highly effective tool for classification.

Understanding the Naive Bayes Classifier

The Naive Bayes Classifier is a supervised classifier. This simply means that the class labels are available for use in building the model. We walk through a detailed step by step example to help clarify this further shortly. However, before we get there, you should know that there are two broad types of supervised classification algorithms:

1. Classification using generative algorithms via the use of the Bayes rule;

2. Classification via discriminant analysis where the classification boundaries are directly learnt from data.

We will cover both types throughout this text. The Naive Bayes Classifier, the subject of this chapter, is a generative classifier.

The Core Concept of Bayes Rule

The easiest way to get a feel for generative algorithms is to use a little probability, and Bayes rule. Bayes rule, named after English statistician, philosopher and Presbyterian minister Thomas Bayes, states that given two random variables A and B, the conditional probability of A given B can be calculated by:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

It is common to refer to $P(A|B)$ as the posterior probability, $P(A)$ and $P(B)$ as prior probabilities, and $P(B|A)$ as the likelihood.

An example

Suppose event A is the probability that your route to work has a traffic major delay. The probability associated with that is, say, 50%. So, $P(A) = 0.5$.

Further, suppose that sometimes you drive, and at other times you take public transport. The benefit of driving is speed and a shorter journey time. The benefit of public transport is it gives you time to get a head-start on the workday by using your tablet computer. Let event B represent the probability that you drive your car, $P(B) = 0.75$.

The probability that you drive given that there is a major traffic delay is $P(B|A) = 0.25$.

Of course, when you are about to leave home you are more interested in the probability that there is a major traffic delay given that you drive i.e. $P(A|B)$.

Using Bayes theorem, we have:

$$P(A|B) = \frac{0.25 \times 0.5}{0.75} = 0.167$$

In other words, there is around a 1 in 6 chance that you will face a major delay.

The generative algorithm

Given a set of class labels y, and a set of explanatory attributes, x, generative algorithms explicitly specify the joint probability distribution of y and x. We denote this by $P(y, x)$. In order to do this, they require an estimate of the conditional distribution $P(x|y)$. Once this is obtained, a predictive distribution for the target y conditional on the attribute x can be calculated by applying Bayes rule:

$$P(y|x) = \frac{P(x|y)P(y)}{\int_y P(x|y)P(y)dy}$$

The entity $P(x|y)P(y)$ defines the joint distribution of y and x.

Discriminative classifiers

In discriminative algorithm's the conditional distribution $P(y|x)$ is modeled directly. It turns out that this is all that is needed for classification. Because discriminative algorithms only model the conditional distribution they can have a much simpler structure than generative based models. Some discriminative classifiers make things even simpler by restricting their focus to specific values. For example, the support vector machine which we discuss in chapter 8, focuses on whether $P(x|y)$ is greater or less than 0.5.

The Bayes Classifier

A Bayes classifier is a generative classifier that can be used to solve classification tasks. To begin let us denote $x = [x_1, ...x_n]$ as the input vector of features/ attributes, and the K classes by $C = \{c_1, c_2, ..., c_K\}$.

A Bayes classifier is a decision rule to estimate the most probably class c_i for an observation given the set of input features or attributes x. It makes good use of Bayes rule.

Conceptually, Bayes rule provides a mechanism to go from the conditional probability of the evidence provided by the input features given the classes $P(x|c_i)$, to the conditional probability of the class given the evidence provided by the attributes $P(c_i|x)$. This is important because in practice, you will often know how frequently some particular evidence occurs, given a known class or outcome.

For example, in medical research, the relative frequency of a disease, and the probability of a medical test being positive given the disease are often known. A patient with a positive test result is more interested in the probability of the disease, given that the medical test is positive. Bayes rule allows the computation of this probability.

A simple rule

Since by Bayes rule $P(c_i|x) \propto P(c_i)P(x|c_i)$, the Bayes classifier chooses a class by assigning x to c_i if:

$$P(c_i)P(x|c_i) > P(c_j)P(x|c_j)\ \forall j \in \{1, ..., K\};\ j \neq k$$

The intuition behind multiplying $P(x|c_i)$ by $P(c_i)$ is to give a higher weight to more frequently occurring classes, and lower weight to less likely classes.

A good classification procedure is one that minimizes the costs of misclassification. It turns out the Bayes classifier minimizes loss when risk is defined as the probability of misclassification. It gives the smallest possible misclassification error known as the Bayes error rate. The Bayes rule is the optimal classification rule if the underlying distribution of the data is known. The Naive Bayes Classifier is an approximation of the Bayes classifier.

NOTE... ✍

- The probability of observing x given class c_i, denoted by $P(x|c_i)$, are called the class conditional probabilities.

- The probabilities of occurrence of different classes, denoted by $P(c_i)$, are generally called the class prior probabilities or base rates.

Core Assumptions

The Naive Bayes Classifier (NBC) relies on two critical assumptions; First, there are no hidden or latent attributes. In other words, the set of features in x is complete; Second, all attributes are independent of each other given the class, so that:

$$P(x_1, ..., x_n|c_j) \approx \prod_{i=1}^{n} P(x_i|c_j)$$

This assumption reduces the number of parameters to be estimated.

Is this realistic?

The assumption of conditional independence may appear to be rather a strong claim to make about your data. I often think of it a bit like the requirement in linear regression that the explanatory variables be independent. Despite this unrealistic assumption, linear regression has found many useful real world applications.

Although independence is generally a poor assumption, in practice on real world data, NBC often competes well with much more sophisticated techniques.

NOTE... ✍

The assumption of mutually conditionally independence appears not to significantly compromise prediction accuracy. Indeed, University of Washington Professor Pedro Domingos and scholar Michael Pazzani have shown the optimality of the naive Bayes classifier even when the conditional independence assumption is significantly violated.

A Step by Step Example to Build Your Intuition

If you have ever been to London, you will have noticed that the buses tend to have two levels, and the London cabs are "*boxy*", black, and surprisingly roomy. Suppose you have been asked to develop a vehicle identification system using the following vehicle attributes:

1. Whether the vehicle is short;
2. whether the vehicle has four doors;
3. whether the vehicle is black.

Furthermore, a labeled sample of 1000 vehicles has been collected, containing a total of 300 black cabs, 350 Buses, and 350 regular cars, as shown in Table 13.

Type	Short	Not Short	Four Doors	Not Four Doors	Black	Not Black
Cab	200	100	150	150	225	75
Bus	50	300	25	325	5	345
Car	300	50	250	100	100	250

Table 13: Attributes and vehicle object type

Now, suppose you are presented with an unknown vehicle with the following attributes - "*Short*", "*Four Door*" and "*Black*". Is it a cab, bus or regular car?

Using the empirical sample, we can easily calculate the prior probabilities. We have:

1. $P(Cab) = \frac{300}{1000} = 0.3,$
2. $P(Bus) = \frac{350}{1000} = 0.35,$
3. $P(Car) = \frac{350}{1000} = 0.35.$

So, our prior probabilities for vehicle type lean towards "*Car*" or "*Bus*". Our intuition tells us "*Car*" or "*Cab*".

Let's calculate the prior probabilities for the evidence given by the attributes:

1. $P(Short) = \frac{200+50+300}{1000} = 0.55$,
2. $P(Four\ Door) = \frac{150+25+250}{1000} = 0.425$,
3. $P(Black) = \frac{225+5+100}{1000} = 0.33$.

Next, we need to calculate the likelihoods for each group of attributes. First, for "*Short*":

1. $P(Short|Cab) = \frac{200}{300} = 0.667$,
2. $P(Short|Bus) = \frac{50}{350} = 0.143$,
3. $P(Short|Car) = \frac{300}{350} = 0.857$.

Next, for "*Four Door*":

1. $P(Four\ Door|Cab) = \frac{150}{300} = 0.5$,
2. $P(Four\ Door|Bus) = \frac{25}{350} = 0.071$,
3. $P(Four\ Door|Car) = \frac{250}{350} = 0.714$.

Finally, for "*Black*":

1. $P(Black|Cab) = \frac{225}{300} = 0.75$,
2. $P(Black|Bus) = \frac{5}{350} = 0.014$,
3. $P(Black|Car) = \frac{100}{350} = 0.286$.

Now, all we need to do is calculate $P(c_j)\prod_{i=1}^{N} P(x_i|c_j)$ for each of the three vehicle classes, and pick the largest values as the most likely. For example:

$P(Cab|Short, Four\ Door, Black) = P(Cab) \times P(Short|Cab) \times P(Four\ Door|Cab) \times P(Four\ Door|Cab)$
$= 0.3 \times 0.667 \times 0.5 \times 0.75$
$= 0.075$.

Using a similar method, we find that:

- $P(Bus|Short, Four\ Door, Black) = 0.01$
- $P(Car|Short, Four\ Door, Black) = 0.061$

Finally, since $P(Cab|Short, Four\ \ Door, Black)$ is the largest value, we assign the unknown vehicle to that class.

Advantages of the Naive Bayes Classifier

NBC is relatively simple to understand and build. Numerically, it involves a bunch of simple counts and ordinary division. This makes it extremely fast to train, even with large samples. It is also super-fast to classify new observations.

Provided the features are independent, it will converge much faster on a solution than discriminative models like logistic regression. This means you need less training data to build a highly accurate prediction model.

It can be used where probabilistic predictions are required, and is not sensitive to irrelevant features. The larger the sample size, the less relevant the irrelevant features become. It can also handle real and discrete data, and can be easily deployed in real time online systems.

Practical Application of the Naive Bayes Classifier

NBC has been successfully deployed in numerous areas, and for several decades. It continues to deliver outstanding performance on many practical data science based projects. In this section, we discuss three areas of ongoing success. The first is human gesture recognition. The second, medical research; and the third area is in the field of traffic congestion.

Gesture Recognition

In the 1968 movie, *2001: A Space Odyssey*, HAL 9000, the ship computer, begins to behave irrationally. When HAL states that it is "*foolproof and incapable of error*", the pilots convene a meeting out of earshot of the machine. However, HAL with its surveillance camera, lip reads the conversation; The humans are planning to disconnect it. This would disrupt HAL's mission. Carefully, the machine begins to kill the humans.

As hinted at in the half century old movie, gesture recognition by computers has the potential to totally transform human-computer interaction. Scholars Escalante et al. deploy the NBC algorithm for this task. They used three different data sets of gesture recognition and human action. The largest data set had 10,304 train set examples, 21 classes and 2,000 features. The NBC delivered outstanding results for all three datasets. The researchers observe that NBC:

> "*...is extremely simple and very fast, yet it compares favorably with more elaborated state of the art methodologies...*"

Medical Research

Al-Aidaroos et al compare NBC performance on 15 medical classification problems. These include studies dealing with breast cancer, primary tumors, Dermatology, Echocardiogram, diabetes, liver disorders, lung cancer, and Hepatitis.

Whilst most of the studies involved binary classification; the Dermatology study had 6 classes, and the primary tumor study had 21 classes.

NBC performance was compared against a number of more sophisticated techniques including decision trees, feed-forward neural networks, and logistic regression. Performance was assessed using prediction accuracy, and area under the Receiver Operator Characteristic Curve (ROC). In 8 out of the 15 studies, NBC delivered the best performance. This included the 6 class Dermatology study, and the 21 class primary tumor study.

Predicting Traffic Congestion

Traffic congestion on major roadways is a major economic cost. Commuters, business, and even emergency services, can suffer from major delays due to the volume of traffic. Large metropolitan areas, such as London, New York and Los Angeles are notorious for highways where traffic moves at a slow crawl.

Managers of traffic systems need tools to predict when and where congestion will occur. Transportation researchers Guangxing Wang and Jiwon Kim develop NBC models to determine whether congestion will occur in the greater Brisbane region, Australia.

Congestion was dichotomized into a binary variable. The goal was to predict whether traffic congestion will occur in a given area of roadway within the next few minutes. Separate NBC target variable were created for 15, 30, 45, and 60 minutes.

Traffic incidents considered by the researchers were crash, hazard, and stationary vehicle. Separate NBC models were developed to predict each of these target variables. Each model predicted whether the specific type of traffic incident would occur in the next hour. Attributes included time, day, day of week, weather and speed.

The sample consisted of 35,040 data points or examples. In total, 245 NBC models were created. Precision, was one of a number of metrics used to evaluate the models. It is the ratio of true positive predictions to all positive prediction. Another metric used was the Recall rate. On average, for the congestion models it was 0.56; For the traffic incident models, it averaged 0.02. Clearly, these are more difficult to predict than congestion. Nevertheless, the researchers conclude:

> "*The validation results show that the proposed* [NBC] *models can successfully predict congestion*

and incident occurrence with a desired level of accuracy."

Example - Classifying Simulation Data

Simulation is a vital tool, and the analysis of simulated outputs is often a necessary part of an empirical study. Let's use the naive Bayes model to predict the labels shown in Figure 4.1. It contains observations on two simulated attributes, `x1` and `x2`, alongside their respective class labels (red circle, and open square).

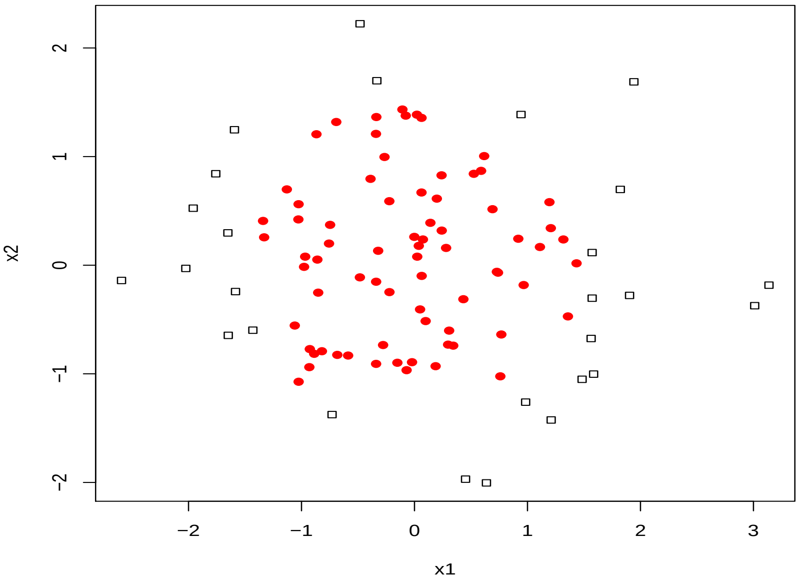


Figure 4.1: Simulated values

Step 1 – Collecting and Exploring the Data

The simulation data was generated for 100 cases, using two attributes x1 and x2, from the Gaussian distribution. Here is how it was created:

```
num_attrib <- 2
N <- 100
set.seed(2017)
x <- matrix(rnorm(N*num_attrib),
ncol=num_attrib)
colnames(x) <- c("x1","x2")
y <- as.numeric((x[,1]^2+x[,2]^2)> 2.3)
```

Let's go through this line by line:

- The first line specifies the number of attributes, in this case 2.
- The second line uses `N` to set the sample size equal to 100.
- Next, the `set.seed` method is used to ensure you can reproduce the example.
- The matrix `x` contains the values of the Gaussian attributes. These are generated using the `rnorm` function;
- and the `colnames` method is used to name the attributes `x2` and `x2`.

Inspecting the class labels and features

The R object `y` contains the class labels in the form of "`0`" or "`1`"; and the object x contains the attributes. To see the first few observations of each, use the head function:

```
head(y)
[1] 0 0 0 1 0 1

head(x)
```

```
                x1          x2
[1,]  1.43420148  0.01745491
[2,] -0.07729196  1.37688667
[3,]  0.73913723 -0.06869535
[4,] -1.75860473  0.84190898
[5,] -0.06982523 -0.96624056
[6,]  0.45190553 -1.96971566
```

Figure 4.2 shows the density plot for x1 and x2 (top), and a bar plot for the class labels in y. As expected, x1 and x2 are approximately normally distributed; and class "`0`" appears more frequently than class "`1`".

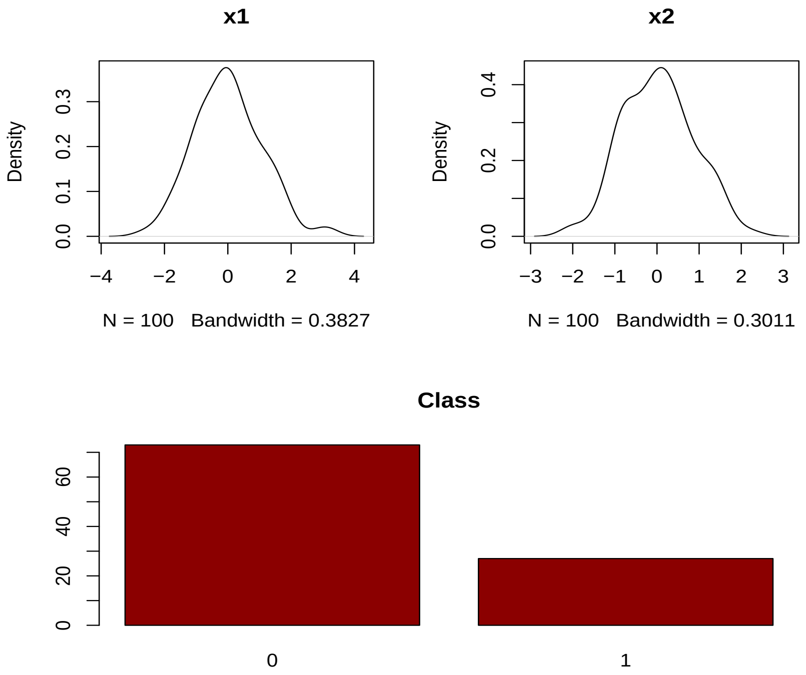


Figure 4.2: Density plot of `x1` and `x2` (top) and bar-plot of class (bottom)

Step 2 – Preparing the Data

Take a look at the class of `y`:

```
class(y)
[1] "numeric"
```

It is numeric. We will transform it into a factor. Here is how to do that:

```
y<-as.factor(y)
```

Next, a little housekeeping, we combine the attributes and class labels into the object `data`. The function `as.data.frame` is used to transform `data` from a matrix to a dataframe. This just makes it a little easier to use with the `naiveBayes` function:

```
data<-cbind(y,x)
data<-as.data.frame(data)
```

Next, we select 70 observations for our training sample, at random without replacement, via the `sample` function:

```
set.seed(2016)
train<-sample(1:N,70,FALSE)
```

Step 3 - Train Model using Train Set

The naive Bayes model can be trained on data using the `e1071` package. First, load the package. Then call the `naiveBayes` function. It is very easy to use, simply pass it the attribute data and the class data:

```
library (e1071)
fit <- naiveBayes(x[train,],
y[train])
```

The fitted model is stored in the R object `fit`.

Step 4 - Evaluate Model Performance

The `predict` function is used to make class predictions. It takes the fitted model (`fit`) and the sample data as arguments. It can report both the actual probabilities, and the class labels. To see the probabilities of the fitted model on the training set add the argument `type ="raw"` as follows:

```
pred_probs <- predict(fit,
data[train,-1],
type ="raw")
```

Here are the first few class probabilities:

```
head(pred_probs)
             0         1
[1,] 0.7922365 0.2077635
[2,] 0.6853967 0.3146033
[3,] 0.7519456 0.2480544
[4,] 0.8872060 0.1127940
[5,] 0.7553854 0.2446146
[6,] 0.4795222 0.5204778
```

The first observation has a posterior probability of 0.79 for class 0, and 0.21 for class 1. The predicted label would be class 0. The sixth observation has a probability of 0.48 for class 0 and 0.52 for class 1; Class 1 is the predicted class label.

Viewing the predicted class labels

It is often helpful to view the predicted class labels. To do this set `type="class"`:

```
pred<-predict(fit,
data[train,-1],
type="class")

head(pred)
[1] 0 0 0 0 0 1
Levels: 0 1
```

As expected, the first observation is assigned to class 0; and the sixth observation is assigned to class 1.

Train set performance

How well did our NBC perform on the training sample? Figure 4.3 shows the actual and predicted class labels. It appears the model predicts the class 0 labels (red dot) with a high degree of accuracy. It is less precise for class 1 (square).

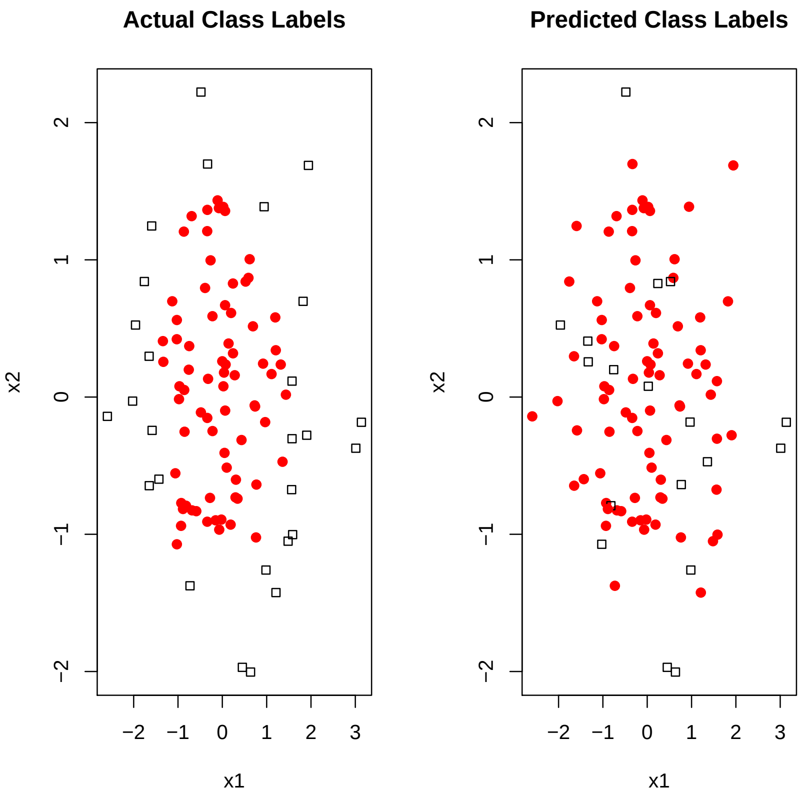


Figure 4.3: Actual and predicted class labels.

Confusion matrix

One way to assess performance quantitatively, is via the confusion matrix. It can be built easily via the `table` function. To make things a little easier to interpret, we pass the train set labels to the R object `y_train`, and then call the `table` function:

```
y_train<-y[train]
table( y_train,pred)
       pred
y_train  0  1
      0 52  0
      1  4 14
```

The model correctly classifies all 52 cases for class 0. It also correctly classifies 14 out of 18 cases for class 1. The overall accuracy on the training set is around 94%.

Step 5 - Assess Test Set Performance

The train set performance is encouraging, how did the model do on the test set? We can follow the above procedure. This time storing the test set labels in the R object `y_test`:

```
pred_test<-predict(fit,
data[-train,-1],
type="class")

y_test<-y[-train]
table( y_test,pred_test)
      pred_test
y_test  0  1
     0 21  0
     1  5  4
```

The model correctly classifies all cases for class 0. However, it only correctly classifies 4 out of 9 cases for class 1. Overall, the accuracy of the model on the test set is 83%, and somewhat lower than the performance delivered by the training sample.

Example - Identifying Radar Signals

The ionosphere is part of Earth's upper atmosphere. It is a very active part of the atmosphere, ionized by solar radiation as a result of the Sun's activity. Gases in the ionosphere are excited by solar radiation to form "ions," which have an electrical charge.

In ionospheric research, radar returns from the ionosphere are classified as either "good" or "bad". Good returns show evidence of some type of structure in the ionosphere, and are suitable for further analysis. This is not the case for bad returns. We build a NBC to identify good and bad radar returns.

Step 1 – Collecting and Exploring the Data

The radar sample was collected by a radar system, located in Goose Bay, Labrador. It contains 351 instances and 34 numeric attributes. The sample data is contained in the `evclass` package:

```
data("ionosphere",package="evclass")
```

We can use the `str` function to give us an overview of the `ionosphere` object:

```
str(ionosphere)
List of 2
 $ x: num [1:351, 1:34] 1 1 1 1 1 1 1 1 1 1
 $ y: num [1:351] 1 1 1 1 1 1 1 1 1 1 ...
```

The object is a composed of 2 list objects. The first list object contains the attributes. There are 34 in total. The second list object, contains the target variable. The target variable is binary. It can be viewed by appending `$y` to the `ionosphere` object.

The target variable

Figure 4.4 presents a bar-chart of the target variable with the labeled classes. We observe that good radar returns are far more frequent, in this sample, than bad radar returns.

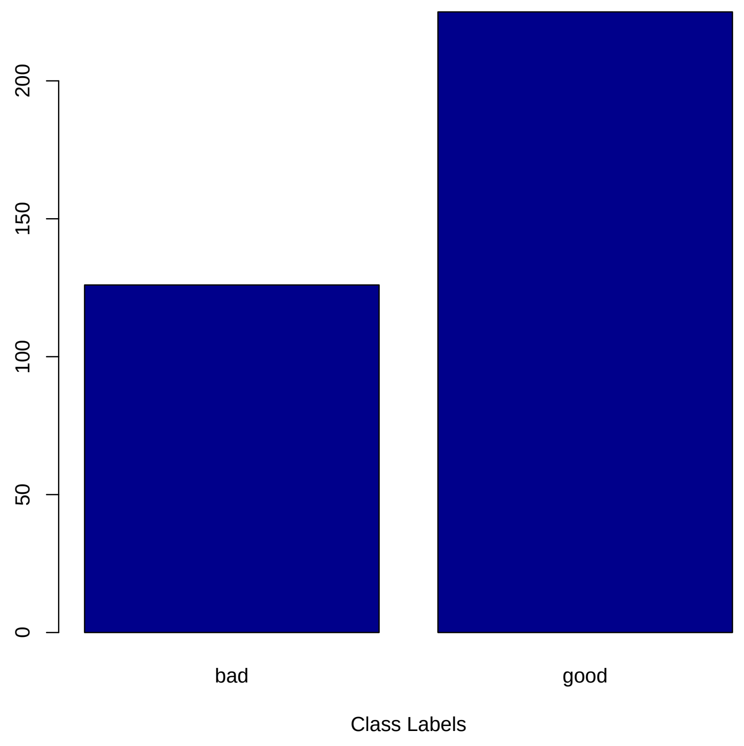


Figure 4.4: Bar-plot of ionosphere target variable classes

Attribute distribution

Ideally, the attributes would be generated from a Normal distribution. Figure 4.5 presents the density plots for nine of the attributes.

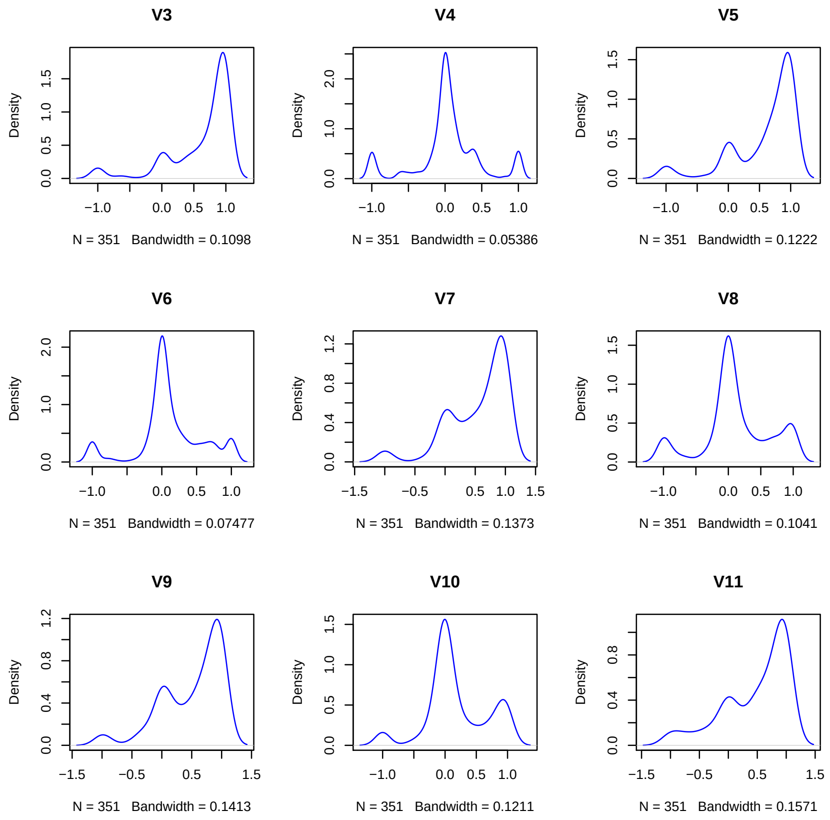


Figure 4.5: Density plots of some of the attributes

Many are highly skewed, and deviate quite considerably for the bell shaped Normal distribution. It seems, any hopes the data are Gaussian, are dashed!

Lack of Normality is common place with real world data. NBC often performs well despite this type of violation. So, lets continue with building our model. We use the R objects x, and y to store the attributes and target variable.

NOTE... ✍

The attributes can be viewed by appending `$x` to the `ionosphere` object. For example:

```
summary(ionosphere$x)
```

Step 2 – Preparing the Data

The first thing to notice is that the second attribute `ionosphere$x[,2]` (attribute `V2`) appears to be constant. To see this use the `summary` function:

```
summary(ionosphere$x[,2])
   Min.  1st  Qu.   Median     Mean  3rd  Qu.      Max.
      0         0        0        0         0         0
```

Since it has no variation, we remove it from the sample,and store the remaining attributes in the R object `x`:

```
x<-as.data.frame(ionosphere$x[,])
x$V2<- NULL
```

The R object x, now contains the attribute set.

Attribute `V1`

As you explored the attributes, you may have noticed that `V1`, is particularly interesting. Take a look using the `summary` function:

```
summary(ionosphere$x[,1])
   Min.  1st  Qu.   Median     Mean  3rd  Qu.      Max.
 0.0000   1.0000   1.0000   0.8917   1.0000   1.0000
```

It takes a minimum value of 0, and a maximum value of 1. Figure 4.6 shows the density plot; it has two peaks! Why?

In turns out that `V1` is a nominal variable that only takes the values 0 and 1. For our initial analysis, let's stick with the continuous variables only. Therefore, we remove it using the `NULL` argument:

x$V1<–NULL

Now, `x` contains the attributes we will use in our analysis.

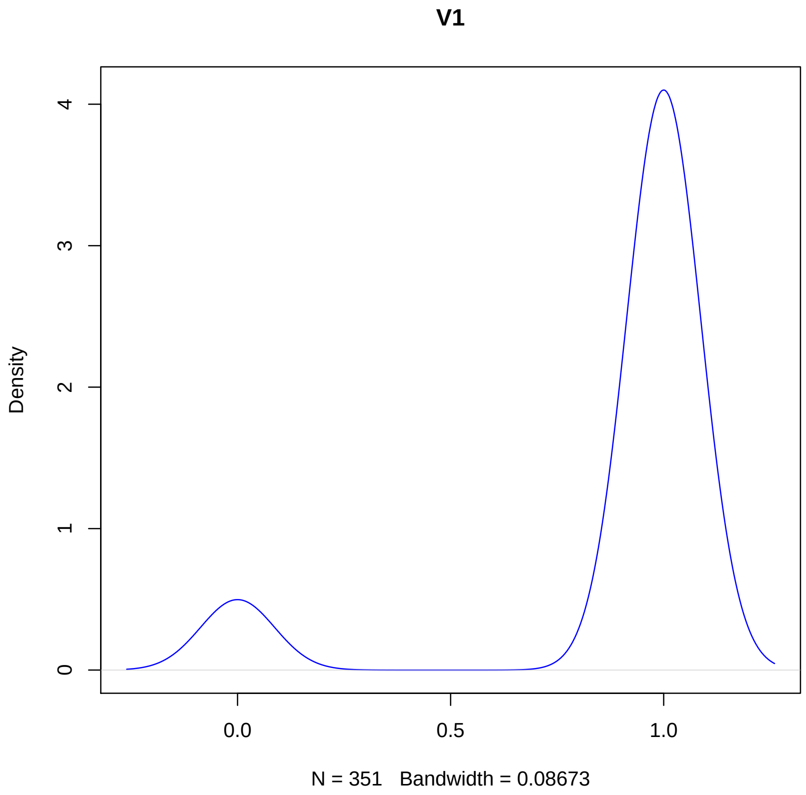


Figure 4.6: Density plot for `V1`

Target variable and random sample

The target variable is stored as a factor in the R object y:

```
y<-as.factor(ionosphere$y)
```

The training sample will contain 251 observations, selected at random without replacement via the `sample` function:

```
set.seed(2018)
N=nrow(ionosphere$x)
train<-sample(1:N,251,FALSE)
```

Step 3 - Train Model using Train Set

To build the NBC, we use the `naiveBayes` function in the `e1071` package. The function takes the target variable followed by the attributes as follows:

```
fit <- naiveBayes(y[train] ~ .,
data = x[train,])
```

You will notice that the model is fully trained on the data in a matter of moments. The fitted model's details are stored in the R object `fit`. Let's take a look at the attributes of `fit`:

```
attributes(fit)
$names
[1] "apriori" "tables"  "levels"  "call"

$class
[1] "naiveBayes"
```

The attributes can be accessed by appending the name to `fit`. For example, to see the apriori distribution of the target variable:

```
fit$apriori

Y
  1   2
160  91
```

There are 160 observations in class 1 ("`good`"), and 91 observations in class 2 ("`bad`").

NOTE... ✍

The argument `$tables` will report the mean and standard deviation of the attributes for each class. If the attributes are categorical, it returns the conditional probabilities given the target variable class.

Step 4 - Evaluate Model Performance

The fitted model probabilities can be viewed using the `predict` function with the type argument set to `raw`. For example, to see the probabilities associated with observations in the test set, you would specify:

```
pred_probs<-predict(fit, x[-train,],
type="raw")

head(round(pred_probs,3),4)

        1    2
[1,] 1.00 0.00
[2,] 0.35 0.65
[3,] 1.00 0.00
[4,] 1.00 0.00
```

The first observation has a posterior probability of 1 for class 1 (`good`), and 0 for class 2 (`bad`). The predicted label would be `"good"`. The second observation has a probability of 0.35 for class 1, and 0.65 for class 2, so class 2 (or `"bad"`) is the recommended class label.

It is often helpful to view the predicted class labels. To do this set `type="class"`:

```
pred<-predict(fit,
x[-train,],type="class")
```

```
head(pred,4)

[1] 1 2 1 1
Levels: 1 2
```

These values confirm our previous observations.

Test set performance

We will use the `confusionMatrix` function in the `caret` package to build the confusion matrix:

```
library(caret)
result <- confusionMatrix(pred,
y[-train])

result$table

          Reference
Prediction  1  2
         1 52  7
         2 13 28
```

For class 1, the model correctly classifies 53 observations, and for class 2 it correctly classifies 27 observations.

The accuracy metric can be viewed as follows:

```
result$overall[1]

Accuracy
     0.8
```

The NBC delivers an overall accuracy of 80%

Step 5 - Improving Model Performance

One way to improve performance, is to reexamine the optimal conditions for the NBC. One key assumption is the independence of the attributes. We can use the correlation coefficient as a crude proxy to assess how well this assumption is met. The idea is that if the features are independent they will have zero correlation.

Figure 4.7 shows a correlation plot of the attributes. The larger the square, the more highly correlated are any two attributes. Take a close look at the figure. It appears a rather large number of the features are correlated.

Correlation between supposedly "independent" attributes is a fact of real world modeling. Nevertheless, removing highly correlated items, might improve performance.

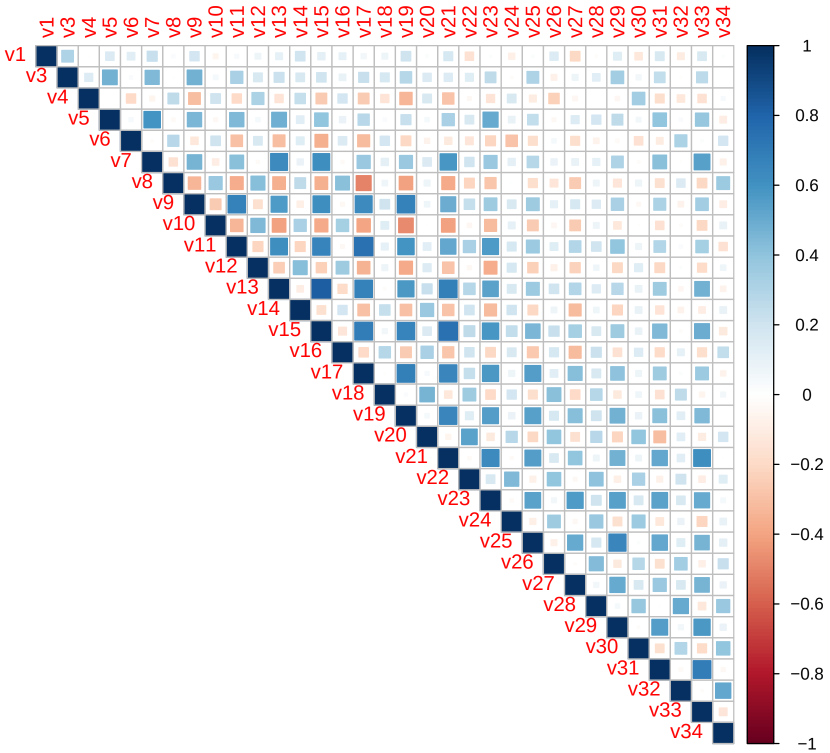


Figure 4.7: Correlation plot of attributes

Using the `findCorrelation` function

To illustrate, we will drop variables that have a correlation in excess of 0.6. The `findCorrelation` function from the `caret` package offers an easy way to achieve this. First, let's prepare the data:

```
x<-as.data.frame(ionosphere$x[,])
x$V2<- NULL
```

The first line reloads the original attribute data into the R object x. The second line deletes `V2` because it only takes the value 0. For this illustration, we keep `V1.` As a nominal binary variable, it may contain useful classification information.

The `findCorrelation` takes two primary arguments. The first is a correlation matrix, and the second is a correlation cutoff. We use 0.6 for the cutoff:

```
findCorrelation(cor(x),
cutoff = 0.6,
exact = TRUE,
names = TRUE)


[1] "V15" "V19" "V21" "V17" "V13" "V11" "
  V25" "V33"
```

The function reports the attributes that exceed the cutoff value. The first reported attribute is `V15`, the second `V19`, and so on. Let's drop these variables from the sample:

```
x$V15<-NULL
x$V19<-NULL
x$V21<-NULL
x$V17<-NULL
x$V13<-NULL
x$V11<-NULL
x$V25<-NULL
x$V33<-NULL
```

Now, re-run the NBC model, make the predictions and return the confusion matrix and accuracy score:

```
fit2 <- naiveBayes(y[train] ~ .,
data = x[train,])

pred2<-predict(fit2, x[-train,],
type="class")

result2 <- confusionMatrix(pred2,
y[-train])

result2$table

  Reference
Prediction  1  2
         1 61  8
         2  4 27

result2$overall[1]

Accuracy
    0.88
```

The overall accuracy increases to 0.88. This is a significant improvement over the first model, `fit`. Notice, the improvement in performance is driven by the model being better able to classify the most frequent class "`good`". No, improvement in the classification performance of class "`bad`" was observed.

NOTE... ✍

Experiment with other combinations of attributes. See if you achieve an accuracy score above 0.90.

Limitations of the Naive Bayes Classifier

The NBC is often criticized for assuming conditional independence of features. This assumption is frequently violated by real world data. In practice, results can be good even in the case of clear violation of this assumption.

However, since NBC estimates probabilities from the empirical data, it can sometimes run into difficulties if there are not enough observations to accurately estimate the probabilities. This can be resolved by collecting more observations, or imposing prior beliefs in place of the empirical estimates.

Like many machine learning algorithms, if the classes are severely imbalanced, classification performance may suffer. For example, in the simulated data analysis we did earlier with R, class 0 had almost 3 times as many observations as class 1. The model correctly classified all cases for class 0. However, it only correctly classified 4 out of 9 cases for class 1.

Summary

In this chapter, we learned about the NBC. It uses conditional probabilities and a simplified version of Bayes rule to perform classification. It is fast, and is especially useful where there are a large number of features and classes.

Despite the core assumption of independent features being frequently violated in real world data, NBC often performs very well. It is useful in areas ranging from medical research to investigating the ionosphere. It is a generative classifier, and a vital tool to add to your machine learning toolkit.

In the next chapter, we discuss an ancient classifier, a workhorse of the Statistician, that is so good, it was rediscovered by machine learning enthusiasts, and deployed with amazing results. It is linear discriminant analysis.

Suggested Reading

- **Gesture Recognition:** Escalante, Hugo Jair, Eduardo F. Morales, and L. Enrique Sucar. "A naive Bayes baseline for early gesture recognition." Pattern Recognition Letters 73 (2016): 91-99.
- **Medical Research:** Al-Aidaroos, K. M., Azuraliza Abu Bakar, and Zalinda Othman. "Medical data classification with Naive Bayes approach." Information Technology Journal 11.9 (2012): 1166.
- **Predicting Traffic Congestion:** Wang, Guangxing, and Jiwon Kim. "The Prediction of Traffic Congestion and Incident on Urban Road Networks Using Naive Bayes Classifier." Australasian Transport Research Forum (ATRF), 38th, 2016, Melbourne, Victoria, Australia. 2016.

Other

- **Assumptions of Naive Bayes:** Rennie, Jason D., et al. "Tackling the poor assumptions of naive Bayes text classifiers." ICML. Vol. 3. 2003.
- **Optimality of Naive Bayes Classifier:** Domingos, Pedro, and Michael Pazzani. "On the optimality of the simple Bayesian classifier under zero-one loss." Machine learning 29.2-3 (1997): 103-130.

Chapter 5

Linear Discriminant Analysis

LINEAR Discriminant Analysis (LDA) is a popular technique used by Statisticians. You may have come across it as a dimensionality reduction technique in the preprocessing step for machine learning models. It is also very useful for classification tasks.

In this chapter, we'll peek inside the mysterious LDA box where you'll:

- Gain an intuitive understanding of how LDA operates.
- Study several practical applications of LDA use for classification.
- Learn how to use LDA as powerful classification tool using R.

Perhaps because LDA has been around for almost a century, and is primarily taught by Statisticians, it does not often feature as a star machine learning tool in textbooks. However, it is a powerful classification technique in its own right, one you should add to your toolkit.

Understanding Linear Discriminant Analysis

The goal of LDA is to find the linear combinations of the feature variables that gives the best possible separation between the groups in a sample. Before we delve into the details, we begin with an intuitive illustration.

A Clarifying Illustration

Back in 1936 British statistician, Sir Ronald Fisher, noted that groups could be well separated using a linear function provided the mean values of each group were sufficiently different from each other. The idea was tested on data collected by American Botanist Edgar Anderson.

Anderson's sample is known as the "*Iris flower data*". It was collected in Canada's beautiful Gaspésie region. It contains 150 observations, 50 each from three species of Iris (Iris setosa, Iris virginica and Iris versicolor).

Four features were measured from each sample. The length and the width of the sepals; and the length and width of the petals in centimeters. Using a combination of these four attributes, Sir Fisher created a linear discriminant model to better distinguish the species from each other.

The group or class means are shown in Table 15. Recall, the idea was that these should be sufficiently different for each class (flower species) in order for linear functions of the features to separate them well. This appears to be the case, for example the average petal width of setosa is 0.246, and 2.026 for virginica.

Discriminant functions

LDA uses linear combinations of features to separate two or more classes. These linear equations are called discriminant functions, which we denote by $g(x)$. They are computed by

	Sepal Length	Sepal Width	Petal Length	Petal Width
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

Table 14: Group means for the Iris sample

searching for a linear combination of features that best separates the classes. In practice, two functions, say $g_1(x)$ and $g_2(x)$, are often sufficient to account for the majority of class differences.

The first discriminant for the Iris data is:

$$g_1(x) = (0.829 \times Sepal\ Length) + (1.534 \times Sepal\ Width)$$

$$+ (-2.201 \times PetalLength) + (-2.810 \times PetalWidth)$$

This discriminant explains more than 99% of the between-group variance in the iris dataset.

The second discriminant is:

$$g_2(x) = (0.024 \times Sepal\ Length) + (2.165 \times Sepal\ Width)$$

$$+ (-0.932 \times PetalLength) + (2.839 \times PetalWidth)$$

It accounts for 0.009% of the between-group variance in the sample. Together, $g_1(x)$ and $g_2(x)$ therefore account for all the between-group variance.

NOTE... ✍

Once the discriminants have been computed, they can be used for classification.

Linear Discriminant Analysis in a Nutshell

In LDA, we are interested in the axes that maximize the separation between multiple classes. Linear discriminant functions are used to define these axes.

The discriminant functions $g_1(x), g_2(x), ..., g_v(x)$ are linear combinations of the original features chosen in such a way that $g_1(x)$ captures the largest proportion of class differences; $g_2(x)$ captures as much as possible the class differences not captured by $g_1(x)$; $g_3(x)$ captures as much as possible the class differences not captured by $g_1(x)$ and $g_2(x)$, and so on.

In general, the first two discriminant functions are usually sufficient to capture most of the between group variance.

Key Assumptions

LDA assumes the within-group covariance matrix is the same for all groups. It also requires the data in each group to have been generated from a multivariate normal distribution. In other words, each feature is assumed to have been generated by a univariate normal distribution. Similar to linear regression, the features are assumed to be uncorrelated.

Advantages of Linear Discriminant Analysis

LDA has widespread application where you want to identify the group to which an object belongs. It can be used to classify two or more groups. Potential applications include predicting credit risk, classifying the quality of an industrial product, and determining the stage of a medical disease. It assumes normally distributed feature variables; and should give solid results in the case when this assumption is fulfilled.

LDA produces linear equations which simplify a multivariate data set. The discriminant functions can be inspected, and provide some insight into the decision-making process of the algorithm.

Practical Application of Linear Discriminant Analysis

Linear discriminant analysis assumes independent continuous attributes from a normal distribution, with all classes sharing the same covariance matrix. This assumption is frequently violated in real world data. Nevertheless, the technique has delivered solid performance in a wide range of real world classification challenges. In this section, we look at three examples, including food authentication, grading of chicken Fillets, and the classification of Autism Spectrum.

Food Authentication

Savory crusty meat pies are a British tradition. They can be eaten hot or cold, are sold by every single grocery store, consumed in cafes, and appear wrapped in elegant paper at upmarket bistros. If you ever travel to the United Kingdom, be sure to try one.

Back in 1846, the story *the String of Pearls*, took Victorian England by storm. In it, a Barber, by the name of Sweeney Todd, murders his customers by casting them down a hole and slitting their throat with his barber razor.

As if this was not terrifying enough, in dark and foggy Victorian London, their remains are processed into savory crusty meat pies, sold in the pie shop of his partner in crime, Mrs. Lovett! The String of Pearls was an instant hit, and derivative works remain popular even today.

In Victorian England, and the United States, you ate out at your own risk. The contents of a meal could be anything, no matter what the label (or food seller) stated. As Eric Schlosser, writing in a 2010 op-ed essay in the New York Times reported:

> "*...In those days, many companies had no qualms about selling children's candy colored with lethal*

heavy metals and rancid food laced with toxic chemicals to disguise the stench; such abuses were widespread."

Today, consumers pay higher prices for free range and organic products. How can they be sure they are receiving the genuine article? Products such as olive oil, milk, honey, coffee, orange juice, saffron, and yes - even meat pies, have been adulterated. This is because, as Eric Schlosser, noted:

> "*...a perverse economic incentive guides the marketplace. Adulterated food is cheaper to produce than safe food.*"

Food authentication is a rapidly growing industry. The primary goal is validation of label information about the food origin, content and production process.

Italian chemical engineers Angelo D'Archivio and Maria Magg, use LDA for the geographical identification of Saffron. The data was derived from Saffron grown and harvested in various regions in Italy. Features were extracted using UV–visible spectroscopy. Eight wavelengths (274, 275, 276, 279, 280, 305, 306, and 328 nm) were selected as representative input features to the LDA model.

Classes represented five distinct geographic locations where the saffron was grown. The underlying assumption was that the data was described by multivariate normal distributions having the same covariance but different location centroids associated with geographic region.

The LDA model accuracy by class was around 90% or better, leading the chemical engineers to conclude:

> "*Because of simple sample preparation and cheap and easy-to-use instrumentation, this method can be preferred to most sophisticated analytical techniques for the geographical traceability of saffron.*"

Grading Chicken Fillets

Chicken is a popular food item in many countries across the world. In the United Kingdom, it accounts for almost half meat consumption; And Americans eat around 60 pounds of chicken per person every year.

Wooden breast is a muscle syndrome that makes chicken breast meat hard, chewy, extremely tough, with a wood like texture. Figure 5.1 illustrates the differences in the morphological structure in chicken breast muscle of normal (left image), and wooden breasted chicken flesh (right image).

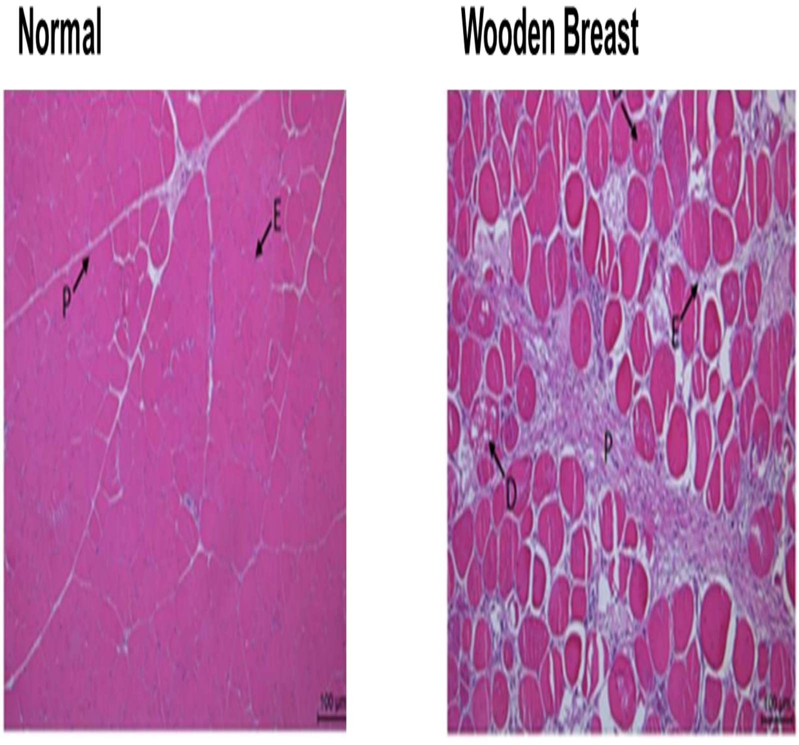


Figure 5.1: Morphological structure in chicken breast muscle. Adapted from Wold et al. See **Chicken Fillets** in suggested reading section for full citation.

Although it poses no threat to human health, the appearance of the meat is unpleasant, and the texture unsatisfactory. Wooden breast can cause significant economic losses for growers, who may see the disease in upwards of 50% of their flocks. The meat is generally processed into lower quality products.

Norwegian food researcher Jens Petter Wold develop a method for rapid grading of wooden breast in chicken breast fillets using LDA.

The sample was composed of 197 skinless chicken breast fillets acquired from a commercial chicken processing facility. A total of 28 of these fillets suffered from wooden breast. Features were extracted using near-infrared (NIR) spectroscopy.

The LDA model achieved 99.5% correct classification. All 28 wooden breast fillets were correctly identified, and one normal fillet was misclassified as wooden breast.

NOTE... ✍

The feature variables were derived from the absorption values at different wavelengths. These were orthogonalized using an partial least squares regression.

Autism Spectrum Classification

Autism spectrum disorder (ASD) is the name for a group of neurodevelopmental disorders. In the United States, upwards of one in sixty-eight children, has a diagnosis of ASD. Early diagnosis and treatment leads to better outcomes.

Malaysian engineer, Che Hasan, use LDA to identify individuals with the condition. The approach is somewhat novel, in that they use gait patterns in young children to determine which class an individual belongs.

The sample was composed of 24 children with ASD, and 24 healthy children ranging from 4 to 12 years old. The kinematic and kinetic gait characteristics were captured by a system composed of a motion capture device and foot force plates. Features were extracted from a single left limb gait cycle for each participant.

For the kinematic features, the LDA had a classification accuracy of 70%. This is not terribly high, and certainly could be improved upon. For the kinetic features, the LDA had an accuracy of 82.5%. Che Hasan concludes:

> "*Overall the results of this study suggest that LDA classifier with kinetic gait features as input predictors is more effective for categorising gait patterns of children with ASD. These potential findings would be beneficial for future applications in identifying gait abnormalities in individuals with ASD or other pathological gait patterns.*"

Example - Molecular Classification via Gene Expression

No too long ago, the medical profession had no general approach for assigning tumors to known classes. The traditional approach to cancer classification was based on the shape and size of the tumor. The challenge was that tumors with similar size and shape can have very different responses to intervention therapy.

In this section, we illustrate how to use LDA for cancer classification based on gene expression applied to human acute leukemia.

Step 1 – Collecting and Exploring the Data

The sample we use is contained in the `leukemia` object in the `ReorderCluster` package. It contains 100 genes expressed from bone marrow samples obtained from 72 acute leukemia patients. First, load the data:

```
data("leukemia",package="ReorderCluster")
```

The class labels are stored as a three-level factor in column 102 of the `leukemia` object. Figure 5.2 shows the distribution of the class labels. Class `B` is the most frequent occurring tumor class, followed by Class `M`, with `T` occurring in the minority of cases.

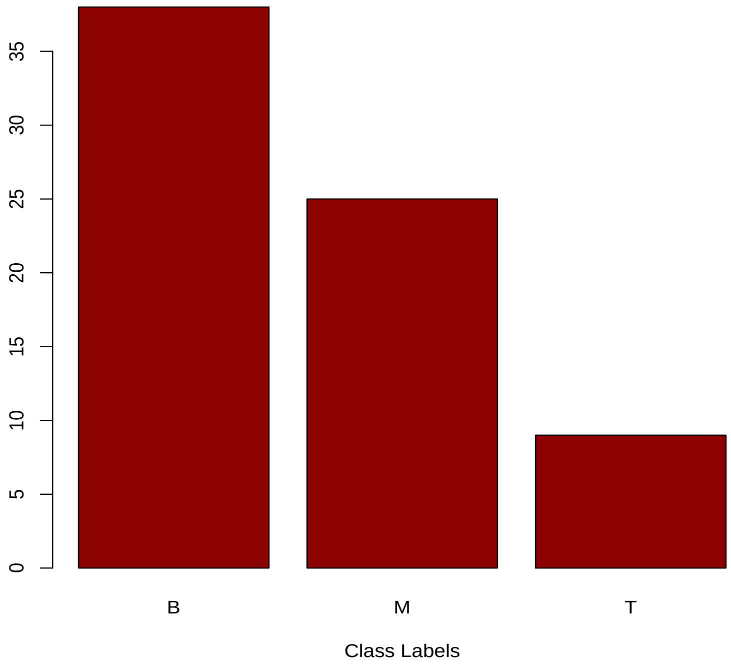


Figure 5.2: Bar-plot of classes for `leukemia`

The features are stored in columns 2 to 101 of the `leukemia` object. Figure 5.3 plots the characteristics for the first nine features. Two things are immediately evident from this figure:

- First, some of the genes have high correlation with each other;

- Second, features do not appear to be symmetric, as one might expect if they were from the Normal distribution.

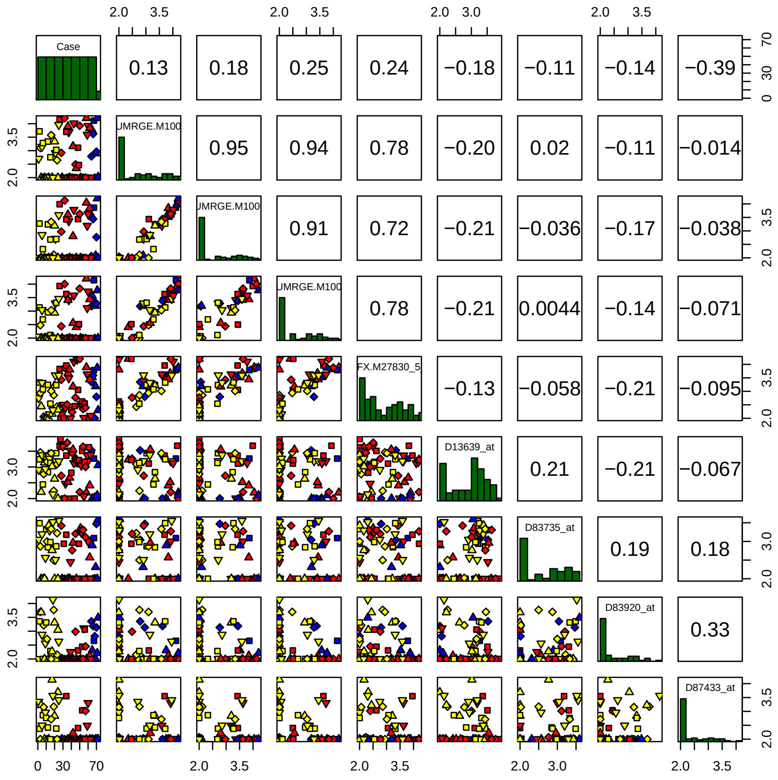


Figure 5.3: Pairs plot of a sample of the genes (features) for `leukemia`

Non-normality of the features

To dig a little deeper into the Normality of the data, Figure 5.4 shows the density plots for several of the features. There is considerable skew, and humps in the distribution of the features. In practice, away from the theoretical textbooks, real-world data often exhibits these characteristics.

Figure 5.5, confirms the lack of normality of the features. This is particularly evident in the left tail of the distributions.

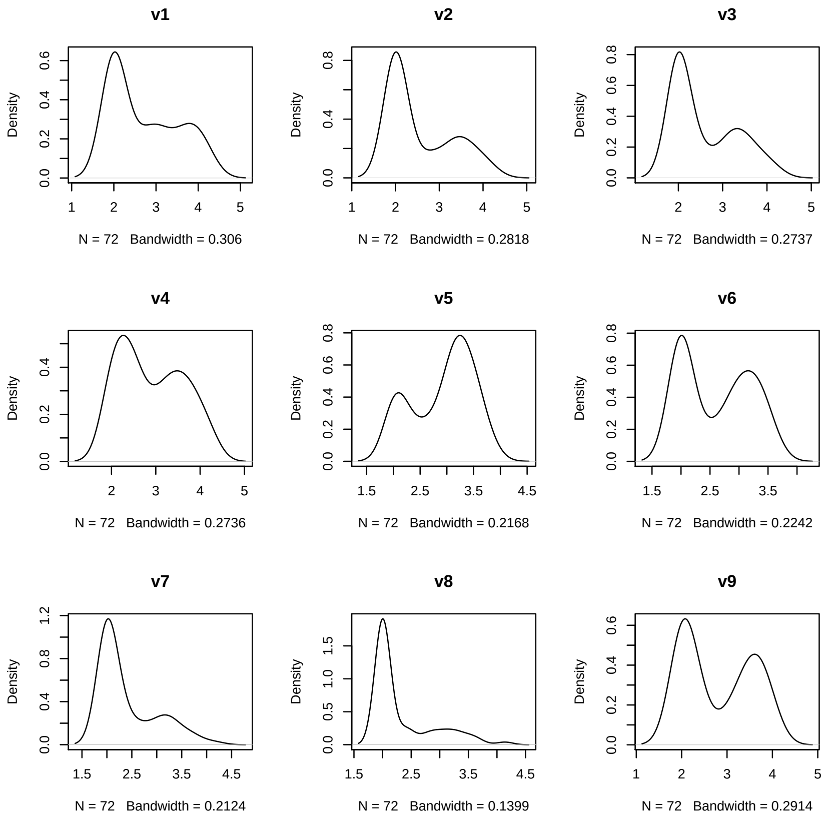


Figure 5.4: Density plots for several features of `leukemia`

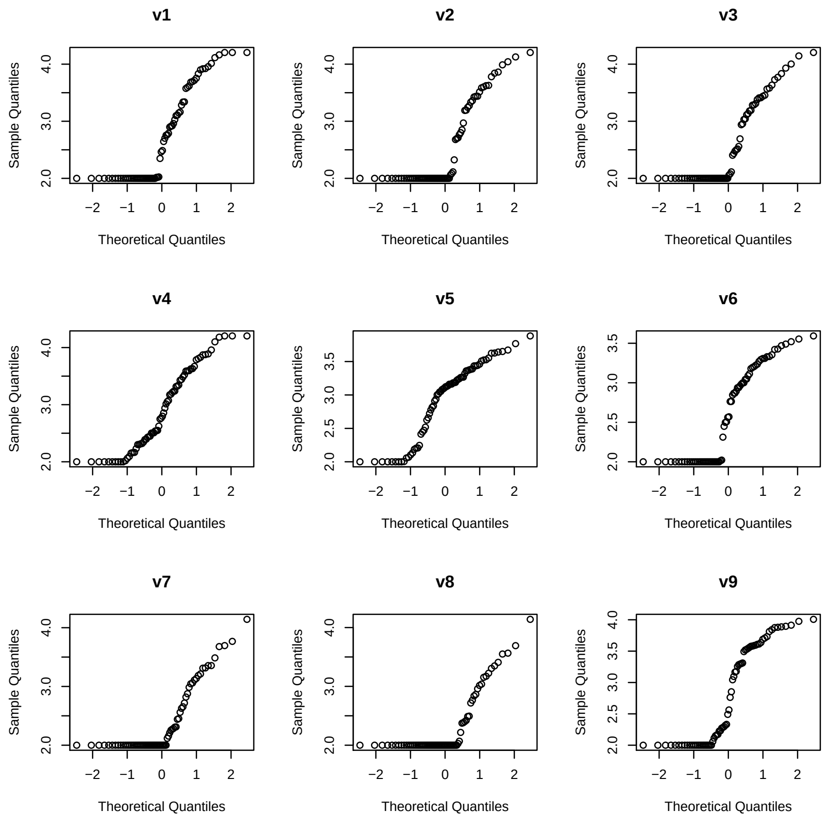


Figure 5.5: qnorm plots for several features of `leukemia`

Step 2 – Preparing the Data

From Figure 5.3, we saw that several of the features are highly correlated. We would like to identify and remove these as they violate the independence assumption. To identify candidates for removal, we calculate the Variance Inflation Factor (VIF) for each attributed.

VIF can be used to detect collinearity (strong correlation) between two or more features. If a feature has a strong linear relationship with at least one other feature, the correlation coefficient would be close to ± 1, and VIF for that variable will be large. One rule of thumb is to drop features whose VIF is greater than 8.

The `vifstep` function

The `vifstep` function in the `usdm` package calculates the VIF. It takes the features and VIF threshold as arguments:

```
x<-data.matrix(leukemia[,2:101])
require(usdm)
vif<-vifstep(x,th=8)
```

The first line transfers the features into the R object `x`. The VIF threshold is set to 8 via the `th` argument. The function may take several minutes to run.

The R object `vif` contains features with VIF scores above the threshold value of 8. It also returns a list of the remaining features:

```
vif

59 variables from the 100 input variables
   have collinearity problem:

AFFX.HUMRGE.M10098_5_at AFFX.HUMRGE.
   M10098_M_at AFFX.HUMRGE.M10098_3_at AFFX
   .M27830_5_at D13639_at D83735_at
   D83920_at D87433_at D88270_at D88422_at
   HG987.HT987_at J03909_at J04102_at
   J04164_at J04456_at J04615_at K0191 1_at
    L19686_rna1_at L20971_at M11717_rna1_at
    M11722_at M12759_at M12886_at M16279_at
    M19507_at M21005_at M21624_at M21904_at
    M27891_at M57731_s_at HG3576.
   HT3779_f_at Z83821_cds2_at Y00787_s_at
   U05255_s_at L33930_s_at X05908_at
   X64072_s_at U02020_at M74719_at
   X03934_at X77737_at M84526_at X59871_at
   M28826_at M15395_at M89957_at
   U23852_s_at X62744_at X76223_s_at
   U05259_rna1_at U57341_at U89922_s_at
   M13560_s_at M28130_rna1_s_at M25079_s_at
    U46499_at M87789_s_at M23178_s_at
   M26311_s_at

After excluding the collinear variables,
   the linear correlation coefficients
   ranges between:
min correlation ( Z84721_cds2_at ~
   M96326_rna1_at ):  2.63173e-06
max correlation ( X82240_rna1_at ~
   X58529_at ):  0.7593826

---------- VIFs of the remained variables
   --------
        Variables      VIF
1       M33882_at 3.541528
2       M38591_at 5.353593
3       M38690_at 4.831325
4       M57710_at 7.180962
5       M58459_at 2.518161
6       M63573_at 2.820346
7  M91036_rna1_at 3.766281
8       M91438_at 4.107234
9       M92934_at 3.761027
10      M94345_at 3.937819
11 M96326_rna1_at 6.338710
12      U02687_at 6.048121
13      U09770_at 4.143385
14      U10485_at 5.621536
15      U50743_at 6.542252
16      U60644_at 5.468369
17      X04500_at 5.169753
18      X17042_at 5.642437
```

```
19       X58529_at 7.684512
20 X82240_rna1_at 6.540773
21       X95735_at 6.577419
22       Z22548_at 4.098832
23       Z23090_at 2.318219
24       Z69881_at 4.823357
25 Z84721_cds2_at 2.384101
26       L08895_at 6.405885
27     J03077_s_at 6.217640
28     M21119_s_at 4.255803
29     M34996_s_at 4.304890
30     M16336_s_at 4.131298
31     M27783_s_at 5.662565
32     U20734_s_at 3.095324
33     M30703_s_at 4.184299
34     M57466_s_at 5.682944
35     M63438_s_at 4.834773
36     X00437_s_at 4.127446
37     X65965_s_at 4.990349
38   AF000424_s_at 5.108634
39       M21305_at 3.081309
40 U01317_cds4_at 5.906404
41       M34516_at 5.350216
```

A total of 59 features are identified as having a collinearity problem. Therefore 41 features remain for use in our LDA model.

Removing correlated variables

The `usdm` package has a great function called `exclude`, which allows you to easily drop all 59 highly correlated features using one line of R code:

```
x<-exclude(x, vif)
```

To ensure it worked as expected, check the new number of columns of `x`:

```
ncol(x)
[1] 41
```

Great, just as we expected.

Train and test sets

The R object `x`, contains the features. Let's store the class labels in the R object `y`, and then select 45 observations at random for the train set. The remainder to be used for the test set:

```
y<-leukemia[,102]
data<-data.frame(x,y)
set.seed(2016)
N=nrow(data)
ncol(x)
train<-sample(1:N,45,FALSE)
data_train<-data[train,]
data_test<-data[-train,]
```

The combined sample of attributes and classes is stored in the R object `data`. The train and test samples are now stored in `data_train` and `data_test` respectively.

Step 3 - Train Model using Train Set

We use the `lda` function from the `MASS` package to fit the LDA model:

```
require(MASS)
fit1<-lda(y ~.,data=data_train)
```

The function `lda` function fits a linear discriminant model to the train set data stored in `data_train`. The model is specified using a formula where the response variable `y` is on the left-hand side separated by a `~` from the features. In this example `~.` tells R to use all of the features in `data_train`. The fitted model is stored in the object `fit1`.

NOTE... ✍

The "`data=`" argument tells R where to look for the features used in the formula. In this case, they are stored in `data_train`.

Prior probabilities

Now that the model is saved as the R object `fit1`, we extract information about the fitted model and train set data. The prior probabilities can be viewed by appending `$prior` to `fit1`:

```
round(fit1$prior,3)
    B     M     T
0.511 0.333 0.156
```

The prior probabilities are derived from the actual observations. To see this, notice the training set has 45 observations, and the count of variables in each class is given by:

```
table(y[train])

 B  M  T
23 15  7
```

Class means

A key assumption of LDA is that the classes can be well separated using a linear function provided the mean values of each class are sufficiently different from each other. The class specific means for each feature can be accessed by appending `$means` to the fitted model. For example, to view the class means for column 36, 37 and 38 of the features in `data_train`:

```
round(fit1$means[,36:38],3)
  X00437_s_at X65965_s_at AF000424_s_at
B       2.171       2.790         2.646
```

```
M          2.159          2.941             3.066
T          4.073          2.657             2.983
```

The 36th feature is `X00437`. It has a class mean of 2.17 for `B`, 2.15 for `M`, and 4.07 for `T`. As expected, the class means differ by feature; and they also vary across class labels within a feature.

NOTE... ✍

We have only looked at three features. Investigate other features using `$mean`. What do you observe?

Between class variance

Between class variance measures the variance that is explained by successive discriminant functions. We can use the singular values (`svd`) reported by `lda` to calculate it:

```
prop_var<-round(prop.table(fit1$svd^2),4)
names(prop_var)<-c("LD1","LD2")

prop_var
   LD1    LD2
0.9188 0.0812
```

So, in this case around 92% of the between class variance is explained by the first discriminant function `LD1`; and the remaining 8% explained by the second discriminant function `LD2`.

NOTE... ✍

If you type:

```
summary(fit1)
```

You will see that the between class variance is labeled "`Proportion of trace`"

Feature importance

The `scaling` argument of the fitted model shows the linear combinations of the original features by each discriminant function. They can be used to yield some insight into the most influential features for classification. Features with large absolute values on `LD1` and `LD2` are more likely to influence the classification decision:

```
round(fit1$scaling,2)
                   LD1   LD2
M33882_at       -10.49  1.40
M38591_at        -0.69  2.56
M38690_at         0.90  0.74
M57710_at       -11.73 -0.28
M58459_at        -0.14 -0.02
M63573_at         7.36 -3.26
M91036_rna1_at   -0.70  0.88
M91438_at         1.71 -1.34
M92934_at        -5.39 -4.86
M94345_at        -0.54  2.33
M96326_rna1_at    8.11  0.23
U02687_at        -1.25 -4.71
U09770_at        -2.17 -2.17
U10485_at        -8.58  1.62
U50743_at         5.48  4.51
U60644_at         7.87 -3.20
X04500_at         5.86 -1.33
X17042_at        -9.84 -0.23
X58529_at        10.28  0.98
X82240_rna1_at   -3.75 -1.15
X95735_at        18.35  2.70
Z22548_at         9.55  0.43
Z23090_at        -6.43  1.28
Z69881_at        -8.39  5.54
Z84721_cds2_at   -5.62  0.75
L08895_at        -1.66 -0.87
```

```
J03077_s_at       9.06  2.31
M21119_s_at      -1.28 -0.46
M34996_s_at      -5.48 -1.49
M16336_s_at     -10.84 -4.73
M27783_s_at      -0.95 -0.37
U20734_s_at       3.44  0.34
M30703_s_at      -3.97 -2.59
M57466_s_at      -0.69  3.36
M63438_s_at      -7.82 -1.19
X00437_s_at      -0.17  2.17
X65965_s_at      -3.74  1.90
AF000424_s_at     0.58 -0.95
M21305_at        -0.26 -3.38
U01317_cds4_at    3.67 -2.75
M34516_at         1.48  1.62
```

It is not always convenient to list all of the features, and eyeball the most influential. We can use R to identify the top 3 features for `LD1` and `LD2`. We begin with `LD1`:

```
scale1<-data.frame(fit1$scaling)
ord1<- order(abs(scale1[,1]),
decreasing = TRUE)
names1<-rownames(scale1) [ord1]
```

The `scale1` object stores the LDA coefficients. These are sorted in descending order via the `order` function, with the argument `decreasing = TRUE`. The object `ord1` contains the row numbers of the sorted feature coefficients for `LDA1`. The row-names of the sorted values are stored in `names1`.

Now, we can look at the top three influential features, and their coefficient scores:

```
names1[1:3]
[1] "X95735_at"   "M57710_at"
"M16336_s_at"

scale1[names1[1:3],1]
[1]  18.34747 -11.72637 -10.83697
```

The most influential feature is X95735 with a coefficient of 18.34, followed by M57710 with a coefficient of -11.72. The third most influential feature for LD1 is M16336 with a score of -10.83.

We can use a similar procedure for LD2:

```
ord2<- order(abs(scale1[,2]),
decreasing = TRUE)
names2<-rownames(scale1) [ord2]

names2[1:3]
[1] "Z69881_at"   "M92934_at"   "
   M16336_s_at"

scale1[names2[1:3],1]
[1]  -8.394323  -5.391976 -10.836969
```

We see that Z69881, M92934 and M16336, are the three most influential features for LD2.

Step 4 - Evaluate Model Performance

Now that we have fitted the model, identified influential features, and calculated the between class variance, let's look at classification performance. To begin, we calculate the train set performance of the model. This can be carried out via the predict function. It takes two key arguments. The first is the fitted model, i.e. fit1; and the second argument is the sample, in this case the train set contained in data_train.

Here is how to use the predict function to predict the class labels:

```
pred1_train<-predict(fit1,
data_train)$class
```

The object pred1_train contains the class predictions. You can view the first 9 predictions via the head function:

```
head(pred1_train,9)
```

```
[1] B T M T M T B B B
Levels: B M T
```

The first prediction is for class B, the second for class T, the third for class M, and so on.

How well did the model do with the actual class values? We can use the `table` function to visualize the performance:

```
tab1_train<-table(data$y[train],
pred1_train)
tab1_train
   pred1_train
     B  M  T
  B 23  0  0
  M  0 15  0
  T  0  0  7
```

The model perfectly predicts each class! This is great, but remember this performance is on the training sample. Good performance is quite common. To get a better idea of performance on unseen observations, let's examine the test sample:

```
pred1<-predict(fit1,data_test)$class
tab1<-table(y[-train],
pred1)
tab1
   pred1
    B M T
  B 9 2 4
  M 0 8 2
  T 1 0 1
```

The model misclassified 9 observations. In other words, the model classification accuracy is around 66%. This is a little disappointing given the perfect performance on the training sample.

Step 5 - Improving Model Performance

Ideally, we would like classification performance to be at least 85%. The question is how to achieve that. One solution, that often works well, is to take another look at the assumptions of the LDA model and see if we can better manipulate the data to match these assumptions.

The key thing, that jumps out of the feature data, is the high pair-wise correlation. This violates the independence of attributes assumption. Maybe, if we choose a lower cut off for the VIF, our data will better conform to this assumption. Let's try a cut-off VIF of 5:

```
x<-data.matrix(leukemia[,2:101])
vif<-vifstep(x,th=5)
x<-exclude(x, vif)
```

Now, check the number of features retained in `x`:

```
ncol(x)
[1] 34
```

So, in this illustration, 34 attributes are retained for inclusion in our LDA model.

The next step is to regenerate the train and test samples:

```
data<-data.frame(x,y)
set.seed(2016)
N=nrow(data)
ncol(x)
train<-sample(1:N,45,FALSE)
data_train<-data[train,]
data_test<-data[-train,]
```

Next, fit the model via the `lda` function:

```
fit<-lda(y ~.,data=data_train)
```

The plot function can be used to visualize how well the model separates the classes:

```
plot(fit)
```

Figure 5.6 shows the plot. All three classes are well separated. The first discriminant function (`LD1`) does a great job of separating all three classes. Class `M` clusters in the right side of the image, class `B` clusters on the left side, and class `T` clusters at the top middle. The second discriminant function clearly separates `T` from `B` and `M`.

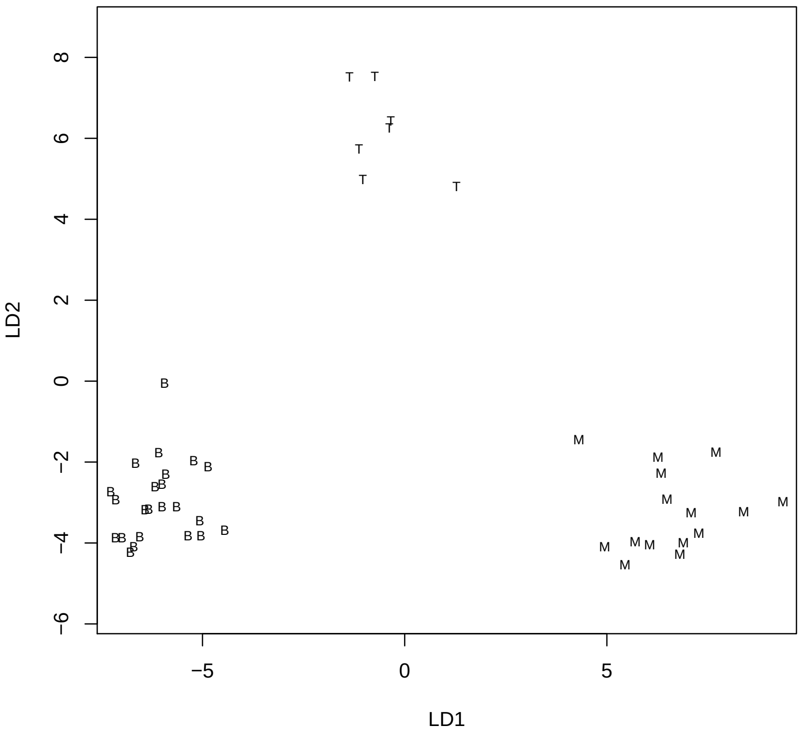


Figure 5.6: plot of lda model `fit`

Test set performance

How did it do with the test data? First, grab the predictions, second, create the confusion matrix:

```
pred<-predict(fit,data_test)$class
tab<-table(data$y[-train],
pred)
tab
   pred
      B  M  T
  B 14  1  0
  M  0  9  1
  T  1  0  1
```

Only 3 observations are misclassified, yielding a classification accuracy of 88.88%. This is significantly higher than for `fit1`.

Limitations of Linear Discriminant Analysis

At the core of LDA is the assumption of multivariate normality. This is often violated in practice. Solutions include engineering the data to better fit the normality assumption, or simply ignoring violations. We can expect the more severe the violation, the weaker classification performance. The same holds true for violations of the assumption that the covariance matrices are equal for groups.

Finally, LDA makes the implicit assumption that all relationships are linear. Complex nonlinear relationships may be missed by assuming linearity.

Summary

In this chapter, we studied LDA as a tool for classification. It involves fitting linear discriminant functions to data. These functions separate the data into classes.

We saw, that in practice features are often highly correlated, and non-normally distributed. Despite this, LDA frequently delivers outstanding classification performance, and continues to be useful in a wide range of applications.

We used it to for molecular classification of gene expression data. Along the way, we learned several methods for examining model fit, reducing the number of highly correlated features, and assessing performance.

In the next chapter, we investigate one of the classical tools of empirical analysis, linear regression.

Suggested Reading

- **Chicken Fillets:** Wold JP, Veiseth-Kent E, Høst V, Løvland A (2017) Rapid on-line detection and grading of wooden breast myopathy in chicken fillets by near-infrared spectroscopy. PLoS ONE 12(3): e0173384. doi:10.1371/journal.pone.0173384
- **Autism Spectrum:** Hasan, Che Zawiyah Che, et al. "Automated Classification of Autism Spectrum Disorders Gait Patterns Using Discriminant Analysis Based on Kinematic and Kinetic Gait Features." J. Appl. Environ. Biol. Sci 7.1 (2017): 150-156.
- **Food Authentication and Traceability:** D'Archivio, Angelo Antonio, and Maria Anna Maggi. "Geographical identification of saffron (Crocus sativus L.) by linear discriminant analysis applied to the UV–visible spectra of aqueous extracts." Food Chemistry 219 (2017): 408-413.

Other

- **Development of Linear Discriminant Analysis:** Fisher, Ronald A. "The use of multiple measurements in taxonomic problems." Annals of eugenics 7.2 (1936): 179-188.

- **Food Adulteration:** Eric Schlosser. Unsafe at Any Meal. July 24, 2010. OP-ED. New York Times.

Chapter 6

Linear Regression

Sir Francis Galton's sweet pea experiment back in 1875 propelled linear regression as a popular tool for modeling relationships in data. It subsequently became the "*first choice*" technique of analysis in a wide variety of disciplines. Partly, due to the ease of calculation, theoretical underpinnings and useful insights, it remains a useful tool to this day.

In this chapter, you will:

- Learn the fundamentals of simple and multiple regression.
- Identify the core underlying assumptions.
- Clarify how parameters are estimated.
- Gain insight on a key performance metric.
- Study some interesting modern applications.
- Build linear regression models to predict the length of Paleolithic hand-axes.

Linear regression is taught to students in engineering, social science, medical sciences, physical science, business, and in any discipline that uses empirical data. Our interest lies in its capability as a machine learning algorithm; and that is the primary focus of this chapter.

Understanding Linear Regression

Linear regression was developed to both describe the relationship between a target variable and a set of explanatory features; and to use this relationship to predict the value of the target variable. We begin by postulating a regression function and then make estimates of model parameters given values of the target and feature variables.

Simple Linear Regression

Simple linear regression is most frequently used to investigate whether there is a linear relationship between two quantitative variables. The variable we want to predict (y) is often called the response or target variable. The variable we use for this prediction (x) is called the explanatory or feature variable. Given a sample of n observations on a target and feature variable, the simple linear regression model assumes:

$$y_i = \alpha + \beta x_i \tag{6.1}$$

This is a straight line with intercept α, and slope equal to β. When $\beta > 0$ the line has positive slope. It has a negative slope when for $\beta < 0$. The parameter β is interpreted as the change in y for every unit change in x.

Adding an error term

Since we cannot expect this relationship to hold exactly for all of our paired observations $\{(x_1, y_1), ...(x_n, y_n)\}$, we include the error term ε_i, and write the simple regression equation as:

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

The character ε_i is known as the residual or error term. It measures the error between the actual value y_i and the value implied by equation 6.1. For example, if the linear approximation is:

$$y_i = 1 + 2x_i$$

and we observe $y_1 = 3$ and $x_1 = 1$, then $\varepsilon_1 = y_1 - (\alpha + \beta x_1) = 0$. However, if for the second pair of observations we have $y_1 = 2$ and $x_2 = 3$ then $\varepsilon_2 = 2 - (1 + [2 \times 3]) = -5$.

Clarifying Expected Value

Let's take a slight detour, and talk a little about expected value. We will use the notion in a short while to give us an alternative perspective on linear regression.

The feature x is assumed to be a random variable. A random variable is an observation generated by an underlying probability distribution. There are many probability distributions, the bell shaped normal distribution being one of many.

Figure 6.1 shows the standard normal distribution. In statistical analysis, it is important to have a rough idea of the distribution of the observations. We saw earlier the normality assumption played a key role in linear discriminant analysis.

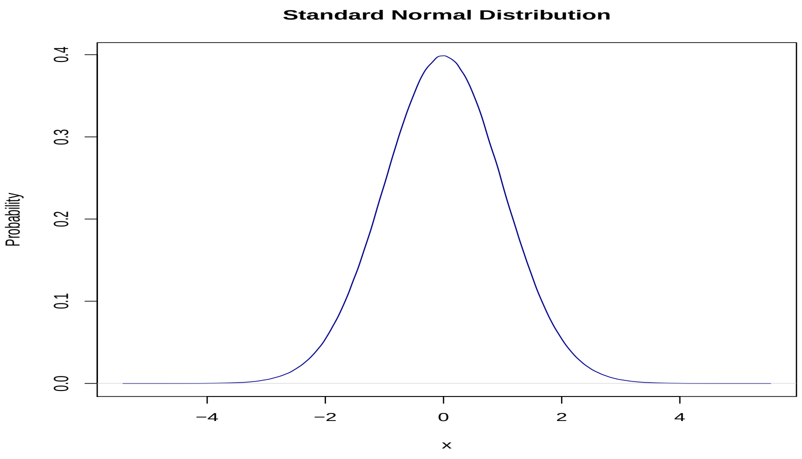


Figure 6.1: Standard Normal Distribution

Expected Value

The expected value of a random variable is a measure of it's average or typical value. It is often denoted by $E[X]$. Given the probability of x, denoted by $p(x)$, the expected value of a discrete random variable X is calculated as:

$$E[X] = \sum_x x \times p(x) \tag{6.2}$$

An Illustration

Suppose, after surprisingly beating your "*know it all*" Boss at their favorite game - tennis, you feel *"Lady Luck"* is on your side. As a one-off event, you decide to buy a lottery ticket. The outcome is a random variable. In fact, there are only two possible outcomes, `win lottery`, or `lose lottery`.

If you have success the random variable $X = 1$, and you will retire from your current job, and tell your Boss where to stick it. However, for failure $X = 0$, and you remain "*friends*".

This type of random variable, with a binary outcome, is known as a Bernoulli random variable. Let the probability of winning the lottery be $Probability(X = 1) = p$. What is the expectation of this random variable?

$$E[X] = \sum_x x \times p(x) = 0 \times (1 - p) + (1 \times p) = p$$

Unfortunately, for most lotteries p is very, very small. For example, in the United Kingdom's National Lottery $p = \frac{1}{13{,}983{,}816}$. Your chance of winning is approximately 1 in 14 million! The chances are *"Lady Luck"* disappears, leaving you and your annoying Boss "*friends*"; And remember to lose next (and every other) time!

An example with R

Let's try to make this concrete with a hands-on example in R. Consider the number of security personal available to protect a highly prestigious exhibit, on a particular day, at the British Museum in London. The manager of security has been asked to report on the expected level of staffing. He knows that on any day there is always at least one individual available for duty, and at most nine. Therefore, X is a discrete random variable that can take on values between 1 to 9. Suppose the probabilities are as follows:

```
probs=c("1"=0.301,"2"=0.176,"3"=0.125,
"4"=0.097,"5"=0.079,"6"=0.067,
"7"=0.058,"8"=0.051,"9"=0.046)


probs
    1     2     3     4     5     6     7     8     9
0.301 0.176 0.125 0.097 0.079 0.067 0.058 0.051 0.046
```

We see the probability that one person is on duty ($X = 1$) is 0.301, and the probability that nine guards are on duty is ($X = 9$) is 0.046.

Since X is discrete, equation 6.2 can be used to calculate the expected value. First, create the R object `Staff,` and then multiply it by `probs`. The values from this calculation are stored in the R object `result`:

```
Staff =1:9
Result = Staff*probs
Result
    1     2     3     4     5     6     7     8     9
0.301 0.352 0.375 0.388 0.395 0.402 0.406 0.408 0.414
```

The sum of `result` gives the expected value:

```
ExpectedStaff=sum(Result)
ExpectedStaff
[1] 3.441
```

Therefore, $E[X] = 3.44$. This informs the manager of security that the average or center of mass of the distribution lies between 3 and 4. However, as the actual number of staff on duty can only take integer values, in reporting the findings, the manager of security is likely to say that the number of security personal available on a particular day are expected to be 3 or

4. The key thing to remember is that an expected value tells you what happens on average.

NOTE... ✍

If X is continuous, the expected value is calculated by:

$$E\left[X\right] = \int_{-\infty}^{+\infty} x f(x) dx$$

where $f(x)$is the probability density function for X.

Linear regression and conditional expectation

The conditional expectation of Y given X is denoted by $E(Y|X)$. We can interpret linear regression as a conditional expectation where:

$$E(Y|X) = \alpha + \beta X$$

The intercept tells us the value of Y that is expected when $X = 0$. The slope parameter β, measures the relationship between X and Y. It is interpreted as the expected (or average change) in Y for a 1-unit change in X.

For example, if we estimate a regression and find:

$$E(Y|X) = 1 + 2X$$

Then a 1 unit change in X is expected to lead to a 2 unit change in Y.

Explaining Ordinary Least Squares

Since we don't know the value of the parameters, α and β, they are estimated from the data. A technique known as ordinary least squares is used for this task. It selects α and β, to minimize the error between the observed value of the target variable (y), and the value predicted by the linear regression equation $(\hat{y})$. Recall, that the residual or error is given by:

$$Residual = Observed\ value - predicted\ value$$

$$\Rightarrow \varepsilon_i = (y_i - \hat{y})$$

In general, the smaller the residual, the better the model fits the observed values of y. However, as we saw on page 130, some residuals are positive and some are negative. Therefore, we square the residuals, and then sum them together. The sum of these squared residuals is called the residual sum of squares (RSS):

$$RSS = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \hat{y})^2 \tag{6.3}$$

The "*best*" regression line chooses α and β to minimize RSS. It turns out this value, for the intercept, can be obtained using the formula:

$$\hat{\alpha} = \overline{Y} - \beta\overline{X}$$

where $\overline{Y}$ and $\overline{X}$ are the sample mean of the target and feature respectively. The slope parameter is calculated by:

$$\hat{\beta} = \frac{\sum_{i=1}^{n} \left(x_i - \overline{X}\right)\left(y_i - \overline{Y}\right)}{\sum_{i=1}^{n} \left(x_i - \overline{X}\right)^2}$$

Coefficient of Determination

The coefficient of determination, also known as R-squared (R^2), is often used to measure how well the simple linear regression model fits the data. It is essentially the square of the correlation coefficient. If there is no linear relationship between X and Y the correlation coefficient is equal to zero and so therefore is the coefficient of determination. Moreover, since the coefficient of determination is the square of the correlation coefficient it lies between zero (no linear relationship) and one (a perfect linearly relationship).

Interpretation

We can interpret the coefficient of determination as the proportion of the variation in the target variable y explained by the linear regression. It is a measure of the percentage of the sample's variance accounted for by the regression model. To see this, note the following quantities:

- $\sum_{i=1}^{n}\left(y_i - \overline{Y}\right)^2$ is the Total Sum of Squares (TSS)
- $\sum_{i=1}^{n}\left(\hat{y}_i - \overline{Y}\right)^2$ is the Explained Sum of Squares (ESS).

The difference between TSS and RSS represents the improvement obtained by using the feature x to explain y. This difference is ESS. In other words, ESS measures the variation accounted by the feature x. In fact, in linear regression:

$$TSS = ESS + RSS$$

So we see that TSS is essentially a measure of the total variation in y explained by both the regression model and the residual ε.

The coefficient of determination can be calculated by taking the ratio of the explained variance to the total variance:

$$R^2 = \frac{variation\ accounted\ for\ by\ x}{total\ variation} = \frac{ESS}{TSS}$$

$$= 1 - \frac{variation\ not\ accounted\ for\ by\ x}{total\ variation} = 1 - \frac{RSS}{TSS}$$

Therefore, provided an intercept term is included in the regression, R^2 is simply the proportion of total variation in the target variable explained by the regression model.

For a poor fitting model, RSS is large and ESS is small, consequently R^2 will be small (close to 0). For a well-fitting model ESS is large and RSS small, and therefore R^2 will be large (close to 1). It is therefore, a measure of the overall quality of the regression.

Multiple Regression

In practice, more than one independent variable or feature will influence the target variable. Multiple regression allows more than one x variable:

$$y_i = \alpha + \beta_1 x_i^1 + \beta_2 x_i^2 + ... + \beta_k x_i^k + \varepsilon_i$$

In this case, we have k features (or independent variables).

Adjusted coefficient of determination

The adequacy of the fit of the multiple regression model is often assessed using the adjusted R^2 statistic. This is because R^2 can be inflated towards its maximum value of 1 simply by adding more independent variables to the regression equation. The adjusted R^2 statistic takes into account the number of explanatory variables in the model:

$$Adjusted\ R^2 = \left[\frac{\frac{RSS}{(n-k)}}{\frac{TSS}{n-1}} \right]$$

If we only have one independent variable in our model (so that we have a simple linear regression) then k =1, and we see that:

$$Adjusted\ R^2 = \left[\frac{\frac{RSS}{(n-k)}}{\frac{TSS}{n-1}} \right] = 1 - \frac{RSS}{TSS} = R^2$$

NOTE... ✍

A simple rule of thumb to avoid over specification is to set the sample size so that $n \geq 5(k+2)$, where n is the sample size, and k the number of features.

Assumptions of the Linear Regression Model

Whilst a high R^2 is desirable, it is not the whole story. It is important that the regression model is valid. The statistical validity of linear regression analysis rests on the following assumptions:

1. **Linearity:** The relationship between the target and features is linear. It can be described by a straight line.

2. **Normality**: The residual ϵ are independently, identically distributed from the standard normal distribution. We write this as $\epsilon \overset{iid}{\sim} N(\mu, \sigma^2)$, where σ is the standard deviation. The standard normal distribution has a mean of zero ($\mu = 0$) and standard deviation of 1 ($\sigma = 1$). Independence implies $\epsilon_i$and ϵ_k are uncorrelated for $i \neq j$. Identical, simply means they are generated from the same underlying fixed probability distribution.

3. **Constant variance:** The variance of the residual term is assumed to be constant. This is known as homoscedasticity in variance. A violation of this assumption is called heteroscedasticity. This assumption implies the variability of y is constant for all values of x.

4. **Independent variables uncorrelated**: The independent variables should be uncorrelated. A violation of this assumption can result in the least squares estimator giving the wrong estimates of the parameters. It also causes

the estimates to be inefficient. Inefficiency implies estimates with highly correlated independent variables will have a larger standard error than the same estimate if the variables were uncorrelated. This hampers statistical hypothesis testing about the size and direction of model parameters. High correlation between the independent variables is sometimes termed as multicollinearity.

5. **No autocorrelation**: The dependent variable and error terms are independently, identically distributed. This ensures errors associated with different observations are independent of one another. It also implies the residuals are not correlated with the dependent variable. A violation of this assumption is called auto or serial correlation.

If the above assumptions are valid, then the ordinary least squares estimators are known as the Best Linear Unbiased Estimators (BLUE). A BLUE estimator has the smallest variance in the class estimators that are linear in the dependent variable. Why is this useful? Because apart from wanting an estimator to be unbiased, we would also like an estimator that is always close to the population parameter we are trying to estimate. One way to measure this closeness is through the standard error of the estimator. If we have two unbiased estimators, one with a large standard error, and the other with a smaller standard error, we would always select the latter.

Advantages of Linear Regression

Linear regression is ubiquitous, and remains the most popular curve fitting technique. It is used in the Sciences, Social Sciences, business, government, and everywhere in-between. It has delivered outstanding results across numerous disciplines.

It is popular because it allows you to examine the relationship between one or more features and a response variable. The linear relationship helps make insights more intuitive. It is therefore useful when you need to describe the relationship between a target variable and a set of explanatory features. Of course, it can also be used to predict the value of the target variable using a set of features, and often does this very well.

Practical Application of Linear Regression

In this section, we look at some interesting and useful applications for predicting soil density, assessing attitudes in age friendly cities, and measuring the relationship between foreign direct investment and economic growth.

Predicting Soil Density

In his wonderfully revealing book *Don't Look Behind You!* Safari guide, Peter Allison, describes his many escapades in the African bush. In one incident, he is cornered by a very angry snake:

> "*I tried mimicking the snake's movement to spare my tiring right side, but just flopped from side to side before losing balance and getting a mouthful of soil. My taste buds were sadly all functioning, and there was something in the dirt that I was sure must have been through an animal's digestive process before it made it into my mouth.*"

Peter's taste buds were correct. The composition of soil depends on the mix between organic and mineral components. In fact, if you look very closely, as Peter did, you will notice soil is composed of tiny particles. These particles are nothing more than very small jagged bits of rock. The weight of an individual soil particle per unit volume is called particle density.

Accurate measurements of soil particle density are required to better understand water, air, heat-flow and chemical transportation through soil. This is especially important for agricultural production. Usually, particle density is expressed in units of grams per cubic centimeter (g/cm^3); and for normal soils is in the range of 2.65g/cm^3 to 2.66g/cm^3. It is higher if soil contains a large amount of heavy minerals such as magnetite; limonite and hematite. It decreases as the amount of organic matter increases.

Agroecologist P. Schjønning built a simple liner regression model to predict solid particle density (Dp) from soil organic matter (SOM). Using a data set of 282 examples, the estimated regression equation was:

$$Dp = 2.646 - 2.8 \times SOM$$

Several things are interesting about this estimated liner regression equation:

- First, the intercept term, takes a value of 2.646 and therefore captures the average Dp of normal soils.
- Second, the slope coefficient, takes a negative sign. This agrees with the scientific observation that Dp decreases with soil organic matter.

Hence, both the intercept and sign of the slope coefficient give us some confidence in the validity of the model. The R^2 for the model was around 0.77.

A multiple regression model was also investigated, this time on another data set. It included clay content ($Clay$) and SOM as independent features. Clay content was expected to have a positive association with Dp. This was confirmed by the regression estimates:

$$Dp = 2.652 + 0.216 \times Clay - 2.237 \times SOM$$

The $R^2 = 0.919$, and as P. Schjønning explains:

> “*This means that nearly 92% of the variation in measured Dp data can be explained by the two independent variables in the model (i.e., Clay and SOM contents).* ”

Age Friendly Cities

Several years ago, the World Health Organization issued a report on age-friendly cities. These are towns, cities and urban areas that:

> “*...encourages active aging by optimizing opportunities for health, participation and security in order to enhance quality of life as people age.*”

Age friendly cities (AFC) are excellent in eight core areas

1. Outdoor spaces and buildings (O)
2. Transportation (T)
3. Housing (H)
4. Social participation (S)
5. Respect and social inclusion (R)
6. Civic participation and employment (CE)
7. Communication and information (CI)
8. Community support and health services (CSH)

Age friendly cities are of growing importance, because as the National Institute on Aging explains:

> “*The world’s population is growing—and aging. Very low birth rates in developed countries, coupled with birth rate declines in most developing countries, are projected to increase the population ages*

65 and over to the point in 2050 when it will be 2.5 times that of the population ages 0-4. This is an exact reversal of the situation in 1950."

Social Scientists Stephen Chan decided to use linear regression to assess the attitudes of senior adults to the components of the Age-friendly City core areas for Hong Kong.

The sample consisted of 682 adults. Two subgroups were separately assessed using linear regression. The first group were participants aged 65-75, a total of 351 individuals. This group were referred to as $young - old$.

The second group, aged 75^+ contained the remaining 331 participants. This group were referred to as $old - old$.

Each group were asked to complete a survey questionnaire consisting of metrics on the Age-friendly City Scale.

The estimated multiple regression, ignoring the intercept, for the first group was:

$$AFC_{young-old} = -0.030 \times O + 0.141 \times T + 0.088 \times H$$

$$+0.207 \times S - 0.136 \times R - 0.09 \times CE$$

$$+0.05 \times CI + 0.194 \times CH$$

The positive coefficients indicate areas where this group are more satisfied, and the negative coefficients reflect dissatisfaction.

The estimated multiple regression, ignoring the intercept, for the second group was:

$$AFC_{old-old} = -0.021 \times O + 0.136 \times T + 0.049 \times H$$

$$+0.147 \times S + 0.002 \times R + 0.117 \times CE$$

$$+0.65 \times CI + 0.06 \times CH$$

Take a close look at the estimated coefficients for $AFC_{young-old}$ relative to $AFC_{young-old}$. What do you observe?
Stephan Chan and his co-researchers comment:

"...young-old adults tended to be more satisfied in an environment with good transportation, social participation, and community and health services, while old-old adults were more satisfied with an environment with good transportation, social participation, and civic participation and employment."

Foreign Direct Investment in Nepal

Nepal, is a beautiful Himalayan country sandwiched between India and China. It is famous for it's Sherpa's, yaks, yetis, monasteries and international travelers. It is also an increasingly modern nation eager to establish a market economy.

Economists argue that Foreign Direct Investment (FDI) from countries such as the United Kingdom, United States, China and the European Union, will have a positive impact on economic growth in Nepal. Other Economists disagree and believe the direct opposite.

Researchers Xinfeng Yan and Majagaiya Kundan Pokhrel build a simple linear regression to investigate the issue. The target variable is Gross Domestic Product (GDP), a measure of overall economic activity in an economy. The sample, is rather small, and consists of 25 annual observations on the pair GDP and FDI.

Their initial simple linear regression is estimated as:

$$GDP = -4100.89 + 64.16 \times FDI$$

The model has an $\mathrm{R}^2 = 0.09$, indicating that 9% of the variation in GDP is explained by FDI. Although low, this is not of great concern as we would not expect all of GDP (or even the majority) to be explained by FDI.

The coefficient on FDI at 64.16 suggests that an increase of one unit of FDI is expected to increase GDP, on average, by 64.16 units. Hence the model appears to confirm a positive relationship between GDP and FDI. However, this interpretation has to be taken with caution because the researchers observe the coefficient on FDI is not statistically significant at the 10% level. They report a p-value = 0.128, meaning they have little confidence that the actual coefficient value on FDI differs much from zero. In addition, the researchers observe autocorrelation in the residual, a direct violation of the assumptions of linear regression.

In the end, Xinfeng and Majagaiya reject the model as being unreliable; and conclude they are unable to confirm a positive relationship between GDP and FDI:

> "*There was no direct way of identifying the linkage between FDI and GDP...*[even] *without the presence of auto-correlation FDI does not adequately describe the GDP.*"

Example - Predicting the Length of Paleolithic Hand-Axes

Labor saving tools have been with man from the very beginning. The very first tools may have been sharp-edged rocks or bone that could be usefully deployed. During the lower Paleolithic period, around 2,500,000 to 200,000 years ago, Acheulean handaxes, made from flint or chert became popular.

Figure 6.2 shows the illustration of a hand axe. It was drawn by John Frere, and found in the parish of Hoxne, in the English county of Suffolk in the year 1800. John conjectured:

> "*They are, I think, evidently weapons of war, fabricated and used by a people who had not the use of metals. They lay in great numbers at the depth of about twelve feet, in a stratified soil, which was dug into for the purpose of raising clay for bricks.*"

In this section, we build a linear regression model to predict the maximum length of a Lower Paleolithic hand axe.

Figure 6.2: The first published picture of a hand axe, drawn by John Frere in the year 1800.

Step 1 – Collecting and Exploring the Data

The Handaxes data-frame contains 600 measurements on Lower Paleolithic hand-axes from Furze Platt, Berkshire, England, see Table 15.

Name	Description
Catalog	Specimen catalog number.
L	Maximum Length.
L1	Distance butt.
B	Maximum breadth.
B1	Breadth from the tip.
B2	Breadth from the butt.
T	Maximum thickness.
T1	Thickness measured at B1.

Table 15: Handaxes data frame.

The data is contained in the archdata package. Load the data:

```
data("Handaxes",package="archdata")
```

We should check the data for unusual values caused by data-entry errors, outliers and so on. For this illustration, we use the summary function and investigate the minimum and maximum of each variable:

```
summary(Handaxes[,3:5])
       L1                 B                 B1
 Min.   : 15.00   Min.   : 34.00   Min.   : 18.00
 1st Qu.: 32.00   1st Qu.: 61.00   1st Qu.: 32.75
 Median : 39.00   Median : 70.00   Median : 40.00
 Mean   : 41.64   Mean   : 70.86   Mean   : 42.21
 3rd Qu.: 49.00   3rd Qu.: 80.00   3rd Qu.: 50.00
 Max.   :136.00   Max.   :123.00   Max.   :103.00
```

```
summary(Handaxes[,5:7])
         B1               B2               T
 Min.   : 18.00   Min.   : 21.00   Min.   :20.00
 1st Qu.: 32.75   1st Qu.: 52.00   1st Qu.:32.00
 Median : 40.00   Median : 62.00   Median :38.00
 Mean   : 42.21   Mean   : 61.97   Mean   :38.44
 3rd Qu.: 50.00   3rd Qu.: 70.00   3rd Qu.:44.00
 Max.   :103.00   Max.   :102.00   Max.   :69.00

summary(Handaxes[,2])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   69.0   103.0   118.0   121.9   136.0   242.0
```

Nope, nothing unusual. All looks in order.

NOTE... ✍

You should check for yourself to ensure that there are no missing values in this data set.

Visual relationships

Figure 6.3 shows the density-plots, correlation matrix, and scatter-plots for the target variable L, and the explanatory attributes. It is interesting to note that:

- The relationship between `L` and each attribute is positive, as indicated by the upward slope in the scatter plots; and the positive correlation coefficients. It appears `L` has a high positive correlation with the features with B (0.77), `B2`(0.70) and `T`(0.61).

- The attributes are all positively correlated with each other. For example, as the maximum breadth increases, so do all the other attributes. This is to be expected - larger handaxes have larger dimensions all round.

- Several of the attributes are highly correlated, for example, `B2` has a 0.85 correlation with `B`; and `B1` has a 0.75 correlation with `L1`.

- The density plots, appear to indicate the attributes are approximately bell shaped. Thus, they might be generated from the normal distribution.

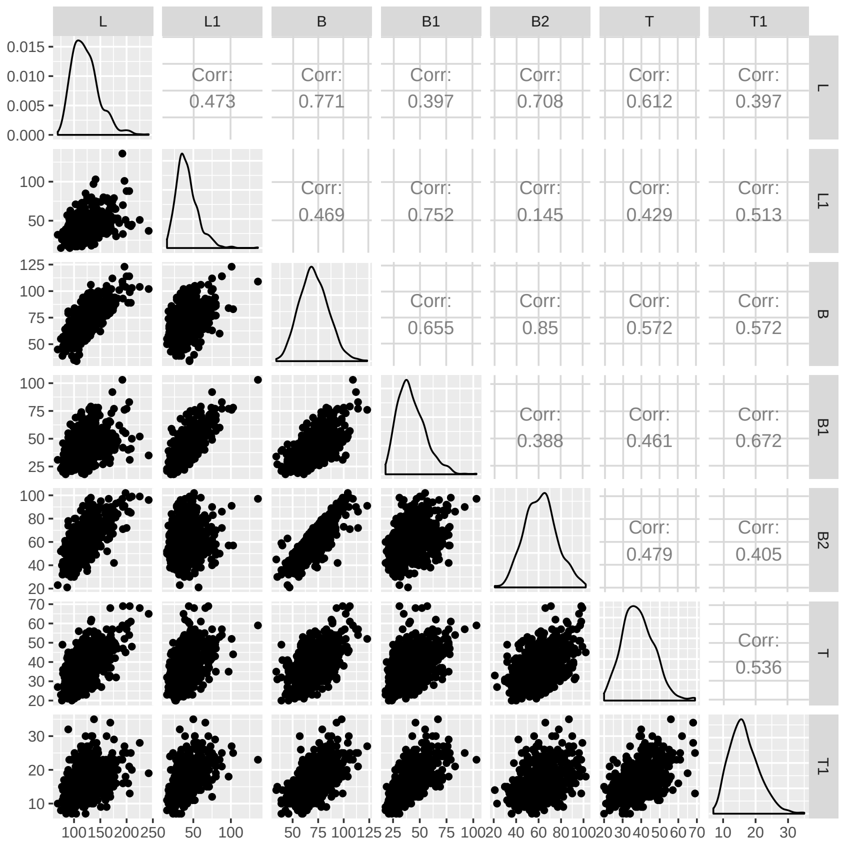


Figure 6.3: Scatter, density and correlation plot for `Handaxes`

Step 2 – Preparing the Data

In regression modeling, a standard approach is to apply a natural log transformation to both the target variable and the attributes before fitting the model. When a simple linear regression model is fitted to logged variables, the slope coefficient represents the predicted percent change in the target variable per percent change in the attribute variable.

We transform the attributes using the natural logarithm via the `log` function. The transformed target variable is stored in the R object `y`, and the attributes in the R object called `data_sample`:

```
y<-log(data.matrix(Handaxes[,2]))
data_sample<-log(data.frame(data.matrix(
   Handaxes[,3:8])))
```

Take a quick look at the features:

```
head(round(data_sample,3))
     L1     B    B1    B2     T    T1
1 4.234 4.369 4.174 4.043 3.714 3.091
2 3.932 4.644 3.951 4.595 4.220 3.332
3 3.932 4.644 4.007 4.625 3.807 2.890
4 3.584 4.357 3.497 4.357 3.611 2.565
5 4.317 4.718 4.522 4.500 4.043 3.219
6 3.555 4.220 3.332 4.159 3.332 2.303
```

and the target variable:

```
head(round(y,3))
      [,1]
[1,] 4.898
[2,] 5.416
[3,] 5.288
[4,] 5.069
[5,] 5.153
[6,] 4.615
```

The `findCorrelation` function

The high correlation between the attributes might mess with the independence assumption of linear regression. To reduce this risk, the `findCorrelation` function can be deployed to automatically identify highly correlated features.

The findCorrelation takes two primary arguments. The first is a correlation matrix, and the second is a correlation cutoff. We use a cutoff of 0.6:

```
cor_matrix<-cor(data.matrix(data_sample))
require(caret)
rid_col<-findCorrelation(cor_matrix,
cutoff = 0.6,exact=FALSE)
```

The object cor_matrix contains the correlation matrix for the features; and rid_col contains the attributes. Let's take a look to see what it found:

```
rid_col
[1] 3 2


colnames(data_sample)[rid_col]
[1] "B1" "B"
```

The findCorrelation function suggests we remove column 3 and 2 from data_sample. These correspond to feature B1 and B. We can drop both using the NULL argument:

```
data_sample$B1<-NULL
data_sample$B<-NULL
```

We had better check that things went as expected:

```
head(round(data_sample,3))
     L1    B2      T    T1
1 4.234 4.043 3.714 3.091
2 3.932 4.595 4.220 3.332
3 3.932 4.625 3.807 2.890
4 3.584 4.357 3.611 2.565
5 4.317 4.500 4.043 3.219
6 3.555 4.159 3.332 2.303
```

Yep, B1 and B have been successfully removed.

Train and test set

We use 550 examples for the test set, selected randomly without replacement, via the `sample` function:

```
set.seed(2016)
N=nrow(data_sample)
train<-sample(1:N,550,FALSE)
y_train<-y[train]
y_test<-y[-train]
data_train<-data_sample[train,]
data_test<-data_sample[-train,]
```

The train sample are reference via the argument "`_train`", and the test sample use "`_test`". Now, let's check we have the correct number of rows in both the train and test set feature sample:

```
nrow(data_train)
[1] 550

nrow(data_test)
[1] 50
```

The numbers are as expected, with 550 examples in the train set, and 50 in the test sample. You should check for yourself that the numbers in the target variable (R object `y`) also match these numbers.

Step 3 - Train Model using Train Set

The function `lm` can be used to perform multiple linear regression in R. The first argument it receives is the standard R formula, followed by the sample data:

```
fit= lm(y_train ~. , data = data_train)
```

The argument "~." tells R to build the regression using all of the variables in `data_train`. The fitted model is stored in the R object `fit`.

The estimated coefficients

The estimated coefficients can be viewed by appending `$coefficients` to `fit`:

```
round(fit$coefficients,3)

(Intercept)          L1          B2           T          T1
      1.376       0.235       0.510       0.216      -0.120
```

The estimated values indicates that the fitted line is given by:

$$\hat{L} = 1.376 + 0.235 \times L1 + 0.510 \times B2$$

$$+0.216 \times T - 0.12 \times T1$$

As expected the coefficients are positive, with the exception of `T1`. For example, a 1 unit increase in $L1$ leads to a 0.235 increase in L.

Exploring the coefficient on $T1$

The coefficient on $T1$ looks a little odd. Especially since the correlation between it and L is positive at 0.612. Let's run a simple linear regression between `L` and `T1`, to confirm our suspicion that something is not quite right here:

```
fitT1=lm(y_train ~T1 , data = data_train)
round(fitT1$coefficients,3)
(Intercept)          T1
     3.869        0.330
```

Yep, in this simple regression, the coefficient has the expected positive sign.

What to do? Well there are many solutions to this issue proposed in the literature. For now, we take the simplest route and drop `T1` from the model:

```
fit= lm(y_train ~L1+B2+T , data = data_train)
round(fit$coefficients,3)
```

```
(Intercept)           L1          B2          T
      1.457        0.197       0.476      0.179
```

The coefficients are all positive. This agrees with both our intuition, and the empirical correlations.

NOTE... ✍

The feature `B2` has the largest impact per unit change `L`, followed by `L1`, then `T`.

Viewing confidence intervals

The 95% confidence intervals can be displayed via the `confint` function:

```
round(confint(fit, level=0.95),3)
            2.5 % 97.5 %
(Intercept) 1.259  1.655
L1          0.165  0.230
B2          0.430  0.523
T           0.123  0.234
```

The lower bound on the confidence intervals are greater than zero for all of the estimates. This gives solid support to the idea that the relationship between the attributes and `L` is positive.

Step 4 - Evaluate Model Performance

The complete statistical summary of the `fit` can be viewed using the `summary` function:

```
summary(fit)
```

```
Call:
lm(formula = y_train ~ L1 + B2 + T, data = data_train
    )

Residuals:
     Min         1Q    Median        3Q       Max
-0.38706  -0.07776  -0.00497   0.08466   0.39887

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.45696    0.10076  14.459  < 2e-16 ***
L1           0.19740    0.01647  11.985  < 2e-16 ***
B2           0.47625    0.02376  20.044  < 2e-16 ***
T            0.17884    0.02819   6.345 4.68e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
    0.1
' 1

Residual standard error: 0.1183 on 546 degrees of
    freedom
Multiple R-squared:  0.6717,    Adjusted R-squared:
    0.6699
F-statistic: 372.4 on 3 and 546 DF,  p-value: < 2.2e
    -16
```

The top part of the output reminds you of the formula used to build the model. The output also reports several quality and performance metrics.

t-test of regression coefficients

In the above regression output, estimates of the model coefficients are provided along with their standard errors. Standard errors are useful for statistically assessing the value of the coefficients. This is done using a t-statistic. The t-statistic is calculated as:

$$t_{statistic} = \frac{estimated\ coefficient}{standard\ error\ of\ estimate}$$

For example, for $L1$ we see that:

$$t_{L1} = \frac{0.19740}{0.01647} = 11.985$$

We will take a close look at this value in a moment. But first, what precisely is the standard error of the estimate?

Standard error of the estimate

Recall that the data from which we derive the parameter estimates is a random sample. If we took measurements on a different sample of Lower Paleolithic hand-axes, say from the central English town of Bedworth, the recorded values would be different. The implies the actual estimates themselves, which are a function of the data, are also random variables. Their value will change from sample to sample. We estimate the variation in the parameter estimates using the standard deviation of the estimate, more commonly called the standard error of the estimate.

The t value

R automatically translates the t value into a probability (see the column `Pr(>|t|)`), we reject the null hypothesis that the estimated coefficient is zero if this probability is very small. The coefficients on all three features and the intercept, have a p-value less than 0.01. In other words, there is no evidence any of these estimated values are different from zero (and we can therefore reject the null hypothesis that they are zero). This agrees with our earlier finding using the `confint` function.

NOTE... ✍

The probability distribution of the above t statistic is the Student t distribution with n-2 degrees of freedom. Where n is the number of examples in the sample.

The Adjusted R-Squared

At the bottom of the output, the R^2 and adjusted R^2 are reported. Since, we have more than one independent feature, the adjusted R^2 is the relevant measure of overall fit. It indicates that around 67% of the variation in `L` is explained by the features.

The F test of regression coefficients

The output also shows the F statistic. It is a joint test of the null hypothesis that none of the explanatory variables have any effect on the dependent variable i.e. the features `L1`, `B2` and `T` have no effect on `L`. Rejection of the null hypothesis implies at least one of the coefficients on the explanatory variables is not equal to zero.

For our example, the F statistic is equal to 372.4, and R reports that the associated probability is less than 2.2e-16 (`p < 2.2e-16`). Thus, we can reject the null hypothesis that all of the coefficients are zero.

NOTE... ✍

This test statistic has an F distribution with k and n-k-1 degrees of freedom.

Check for Non-linear patterns in the residuals

Before we use our model for prediction, we need to ensure it meets the assumptions required by linear regression. Fortunately, R produces several visual plots that help us with this task via the `plot` function:

```
plot(fit ,which=1)
```

You should see Figure 6.4, it shows the residual versus fitted values of `fit`. It is used to detect non-linear patters in the relationship between the target variable and the features. You visually check to see if the residuals are approximately equally spread around the red line (which should be approximately horizontal) without any distinct patterns.

There are no distinctive patterns evident in the chart, and the residuals appear evenly spread around the red line (solid line). Overall, we can draw some confidence that the data do not contain any non-linear relationships.

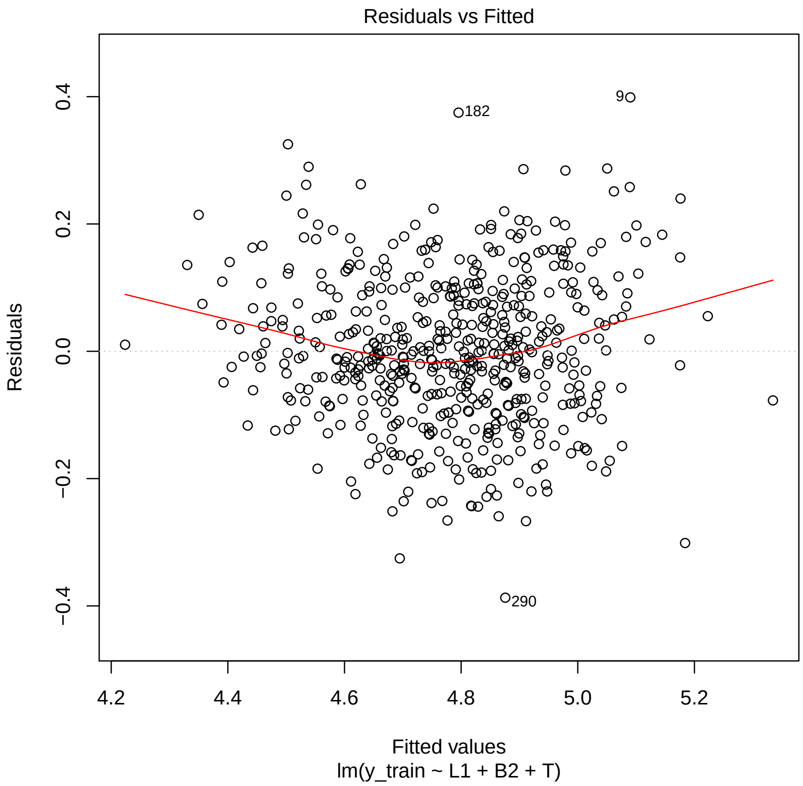


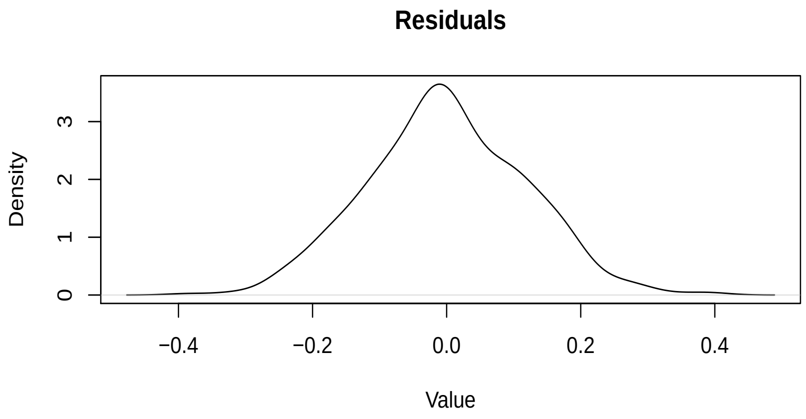
Figure 6.4: Residual v fitted values

Check the distribution of the residuals

We can visually assess the normality of the residuals using a density plot of the residuals, and a normal Q-Q plot:

```
par(mfrow=c(2,1))
plot(density(fit$residuals),
main="Residuals",
xlab="Value")
plot(fit,which=2)
```

Figure 6.5 shows the resultant plots. The density plot is approximately bell shaped. This is a good sign.



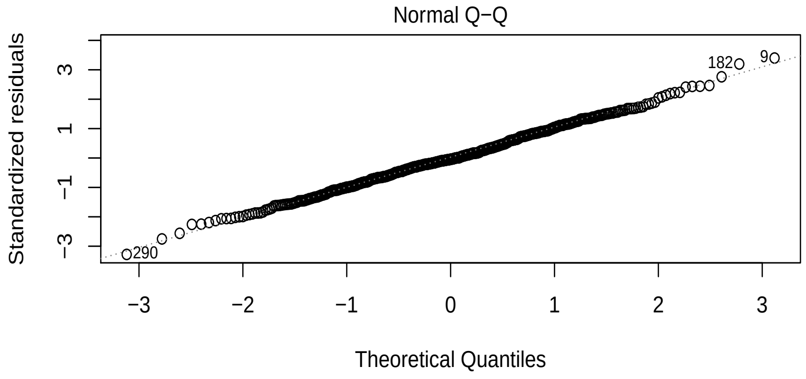


Figure 6.5: Density plot (top) and Normal Q-Q plot (Bottom)

In a Normal Q-Q plot the ordered residuals (y-axis) are plotted against the expected quantiles from a standard normal distribution function (x-axis). The plotted points should lie on an upward sloping (45 degree) straight line.

Figure 6.5 (bottom), shows the residual points fall approximately along a straight upward sloping line. However, several points are annotated on the chart. Points 182, 9 and 290 look a little off, although not by a large amount.

Should we accept that the residuals are from the normal distribution? It appears so, but we can also use a formal statistical check for an added layer of comfort. The Shapiro-Wilks normality test is quick and easy to execute:

```
shapiro.test(fit$residuals)

        Shapiro-Wilk normality test

data:  fit$residuals
W = 0.99811, p-value = 0.8106
```

The null hypothesis is that the residuals are from a normal distribution. The p-value of the test statistic, at 0.81, is rather large (maximum value =1), and we cannot reject the null hypothesis.

NOTE... ✍

Remember, we reject the null hypothesis if the probability (`p-value`) is very small.

Check for equal variance

Heteroskedasticity (non-constant variance) causes the estimated standard error of the estimates to be wrong. This means the confidence intervals and hypotheses tests may not be reliable. The scale-location plot is useful for assessing the constant variance assumption. You can view the plot by setting `which=3` in the `plot` function.

```
plot(fit,which=3)
```

As shown in Figure 6.6, this plot is similar to the residuals versus fitted values plot shown in Figure 6.4, but it uses the square root of the standardized residuals along the y axis. There should be no discernible pattern to the plot with the points spread evenly around a horizontal line.

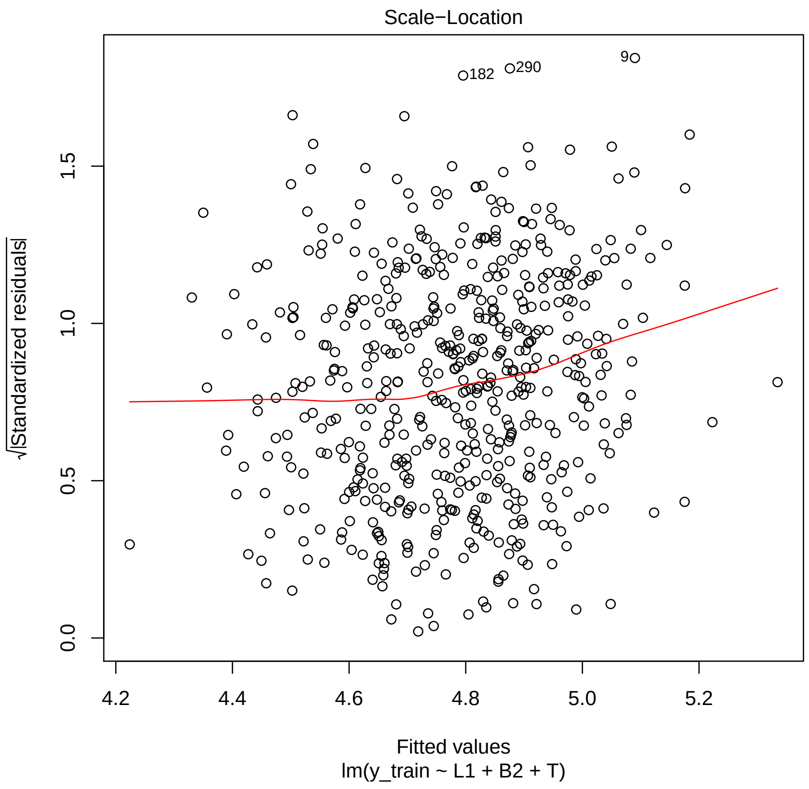


Figure 6.6: Scale Location plot

Looking closely at the figure, the points seem to form a homogeneous cloud, equally spread on both sides of the line. This is good. However, there is a slight upward slope to the line in Figure 6.4, this is a little worrying.

We better dig a little further via formal statistical tests. Two popular tests are the Breuch Pagan test from the `lmtest` library, and the NCV test from the package `car`. In both cases, the null hypothesis is that of homoscedasticity, and we reject the null hypothesis if the p-value is small.

Here is how to call the tests:

```
library(lmtest)
library(car)
bptest(fit)

         studentized Breusch-Pagan test
data:   fit
BP = 12.847, df = 3, p-value = 0.00498

ncvTest(fit)

Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 7.828589      Df = 1       p =
    0.005142623
```

For both statistical tests the p-value is less than 0.01, and we reject the null hypothesis of constant variance.

Heteroskedasticity impacts the standard errors, not the estimated values. Our real concern is to ensure the coefficients in the model are statistically significant (they probably are given their tiny original p-values, but we should double check). The question is what to do? Well, whenever any of the core assumptions of linear regression fail, there are four paths you can follow:

1. Transform the dependent variable;
2. Transform the independent variables;
3. Add or remove independent variables;
4. Look for another regression estimation technique.

Transformations are often used to correct for non-constant variance, non-linearity and non-normality. Popular transformations include the square root and natural logarithm. Since we have already used a log transformation on the dependent and independent variables, and deleted `T1` from the model, let's try the fourth option - "`Look for another regression estimation technique`".

A popular solution for heteroscedasticity, involves adjusting the standard errors to better reflect the actual underlying probabilities for hypothesis testing. The `car` package has the function **`hccm`** that does the job. It calculates the heteroscedasticity-corrected covariance matrix from which the standard errors are derived. Here, is how to use it:

```
coeftest(fit,
vcov=hccm(fit))

t test of coefficients:

             Estimate  Std. Error  t value   Pr(>|t|)
(Intercept)  1.456964    0.109762  13.2738 < 2.2e-16 ***
L1           0.197396    0.016749  11.7853 < 2.2e-16 ***
B2           0.476251    0.026172  18.1970 < 2.2e-16 ***
T            0.178841    0.031264   5.7204 1.753e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
   0.1 ' ' 1
```

The first thing to notice is that the coefficient estimates agree (to three decimal places) with the values, we already estimated using the `lm` function. Second, there is some change in the standard errors from those observed in `fit`, although the absolute change is not very large. Third, even with the adjustment to the standard errors for heteroscedasticity, all the coefficients, including the intercept, continue to have tiny p-values.

Check for influential observations

We want to check for influential observations. These are observations that greatly impact the slope of the regression line. One natural way to search for these is to identify the largest absolute residuals. We can use the `which.max` function to identify these values:

```
which.max(abs(fit$residuals))
  9
469
```

It appears two observations are identified. These are row 9 and row 469 in the original `Handaxes` sample. Let's take a look at the attribute values associated with these two observations:

```
rbind(round(data_train[c("9"),],3),
round(data_train[c("469"),],3))

       L1     B2     T      T1
9     3.611  4.564  4.174  2.944
469   2.996  4.127  3.784  2.708
```

Should we get rid of these observations? Probably not. Outlying observations are not always influential. A better way to identify problematic data points is via leverage.

The leverage of an observation measures how far away it is from the other observations. It takes values between 0 and 1. A point with zero leverage has no effect on the regression model. They can be calculated as follows:

```
lev = hat(model.matrix(fit))
summary(lev)

    Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.001843  0.003988  0.006024  0.007273  0.009219  0.043970
```

In this case, the largest value is relatively small, at 0.04. To identify this point type:

```
data_train[lev >0.04,]
          L1        B2        T         T1
333  4.043051  3.044522  3.496508  2.639057
```

It appears observation 333 from the original `Handaxes` sample has the largest leverage. However, at 0.04, it is not of major concern.

You can also visualize the information using the `influencePlot` function:

```
influencePlot(fit)
      StudRes         Hat       CookD
333  1.172131  0.04396762  0.01578540
9    3.433410  0.01736059  0.05105806
```

It reports the influential observations, along with various metrics. Figure 6.7 shows the resultant plot. It identifies observation 333, and also observation 9 as influential.

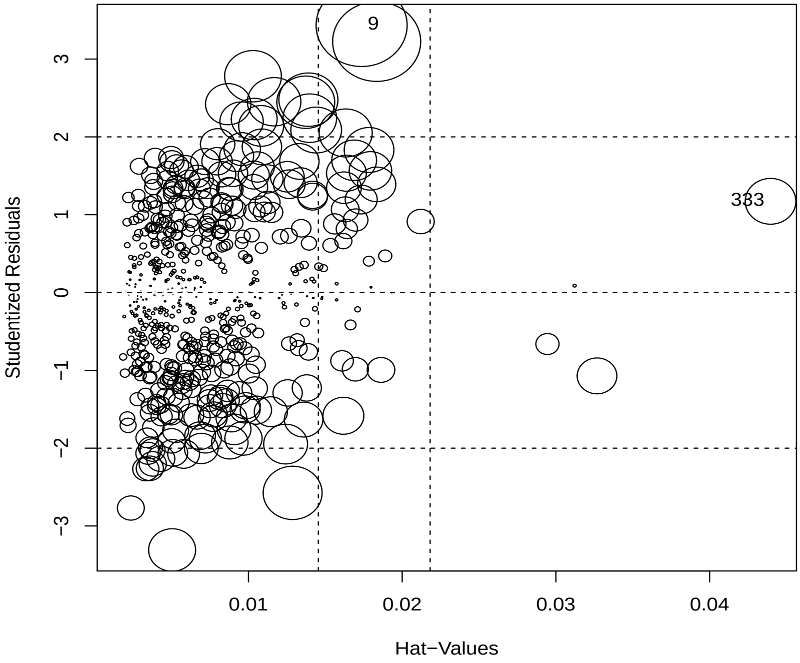


Figure 6.7: Influence plot

You may have noticed that the `influencePlot` function also reports Cook's distance, this metric identifies points which have more influence than other points. Influence is the amount that a data point is affecting the regression line, measured by how much the regression line would change if the point were excluded from the regression model. Points with a large Cook's distance have a high influence, and might require further investigation.

A visual representation of Cook's distance is obtained via:

```
plot(fit ,which=4)
```

Figure 6.7 shows the resultant plot. There are no hard and fast rules for interpreting Cook's distance. Larger values are labeled with their observation number. In this illustration points 232, 182 and 9 represent points which might require further investigation.

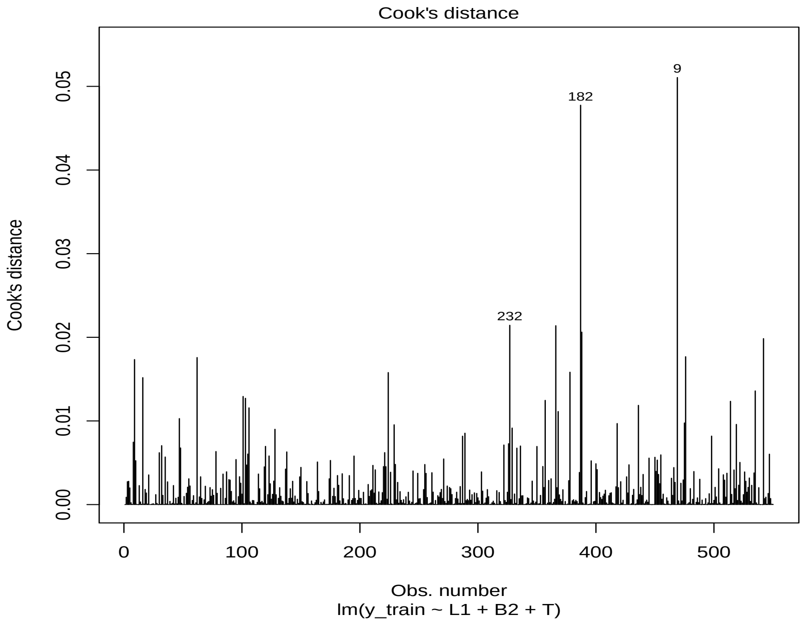


Figure 6.8: Cooks distance plot

The points identified in Figure 6.7, or Figure 6.8 are not necessarily definitive. The key is whether excluding these points will have a significant impact on the regression coefficients. In our case, we have identified a handful of potential observations, none of which appear particularly extreme.

One way to assess whether our intuition is correct, is to re-estimate the model using robust regression, and then check to see whether the coefficient estimates are similar. If they are, then we can leave keep the identified examples in our analysis. If not, you would re-run your regression model, dropping each potential influential observation one at a time, and then assess the impact on the coefficients. Observations that significantly impact the regression coefficients can be excluded from further analysis.

The `rlm` function in the `MASS` package estimates robust regression. We will also use the `fit.models` function to display both regular and robust regression results:

```
require(MASS)
require("fit.models")
fmclass.add.class("lmfm", "rlm")
fm1 <- fit.models(c("rlm", "lm"),
y_train ~L1+B2+T ,
data = data_train)
```

The `fmclass.add.class` function tells R to compare the coefficient estimates produced by `rlm` with `lm`. These values are stored in `fm1`:

```
fm1

Calls:
rlm: rlm(formula = y_train ~ L1 + B2 + T,
    data = data_train)
```

```
lm: lm(formula = y_train ~ L1 + B2 + T, data
     = data_train)

Coefficients:
      (Intercept)      L1      B2      T
rlm        1.4596  0.1982  0.4824  0.170
lm         1.4570  0.1974  0.4763  0.179
```

A quick visual inspection of the estimated coefficient does not real any significant differences between regular regression `(lm)` and robust regression `(rlm)`. We could test for a difference statistically if we saw a large difference, but won't bother in this case.

It appears the identified "outlying" observations are not sufficiently different from the underlying sample for us to remove them from our analysis.

Check for residuals for autocorrelation

A super simple and fast way to check for autocorrelation in the residuals, is to plot the autocorrelation function:

```
acf(fit$residual)
```

Figure 6.9 shows the result. It plots the correlation of the residual $\varepsilon_t$with ε_{t-j}. The first spike occurs because the correlation between ε_t and itself ε_t is always 1. Thereafter the correlation essentially falls to zero. There no evidence of autocorrelation in the residual.

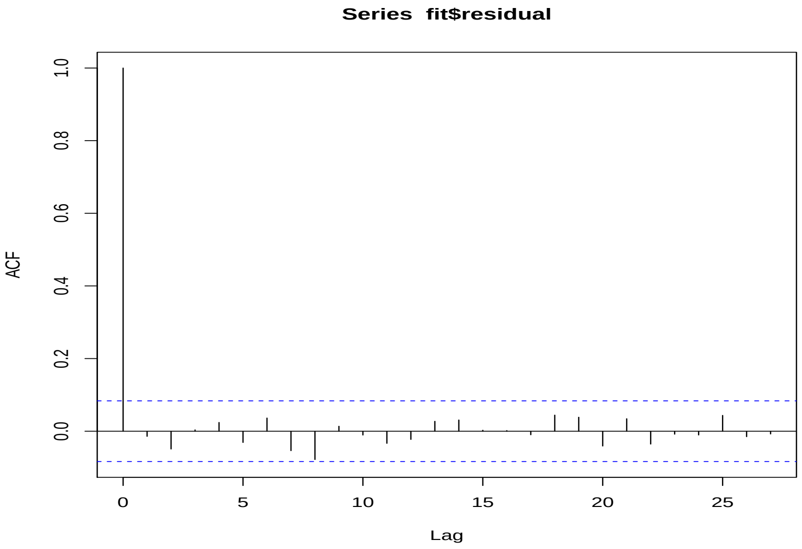


Figure 6.9: Autocorrelation function of residual

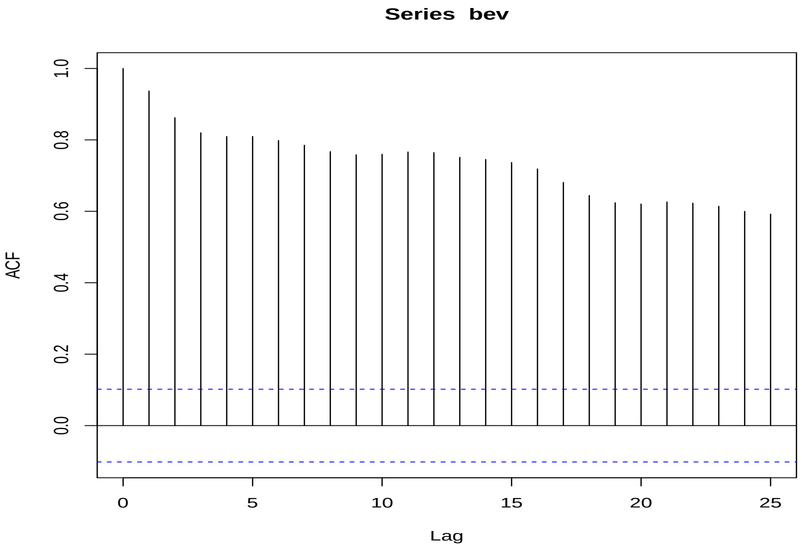


Figure 6.10: Example of autocorrelation in residuals

For illustration, Figure 6.10, shows what you might see if your residuals contained auto-correlation. In this case you might observe several lags with a high correlation with ε_0.

Of course, we could also use a statistical test. The Durbin-Watson test is a popular choice. It is contained in the `lmtest` packaged:

```
dwtest(y_train ~L1+B2+T, data = data_train )

        Durbin-Watson test

data:  y_train ~ L1 + B2 + T
DW = 2.0282, p-value = 0.6303
alternative hypothesis: true autocorrelation
   is greater than 0
```

The p-value, at 0.63, indicates there is no evidence of correlated residuals. It confirms our earlier observation.

NOTE... ✍

In general, if the errors appear to be correlated, you can use generalized least squares estimation, This is implemented by the function `gls()`from the `nlme` package.

Assessing train set performance

Figure 6.11 displays a scatter plot of the target variable and fitted values from `fit`. The points have a relatively narrow dispersion around a positive trend.

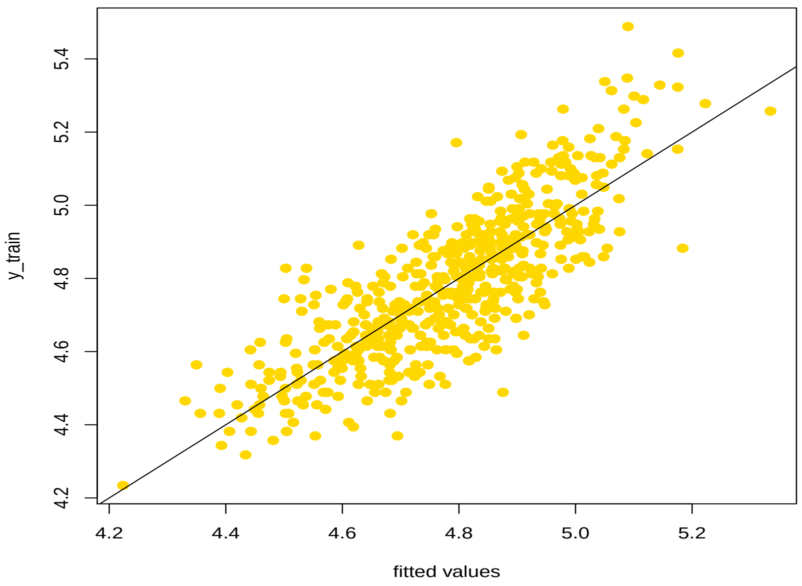


Figure 6.11: Predicted and actual values for training sample

We can calculate the correlation using:

```
round(cor(fit$fitted ,y_train),4)
[1] 0.8196
```

This indicates a fairly strong relationship between the actual and predicted values; The squared correlation coefficient yields the R^2 statistic of 0.67 we saw earlier.

Assessing test set performance

Our primary interest is how the model performs on new unseen data. Our proxy for this is the test set. The `predict` function allows us to assess this:

```
pred <- predict(fit, data_test)
round(head(pred,3),3)
   34    45    49
```

```
4.699  5.014  4.950
```

The first prediction takes a value of 4.699, and the second a value of 5.014 (remember we are predicting log y). From Figure 6.12 we see the predicted values align closely with the observed values. The correlation is 0.751, and R^2 is 0.563:

```
r2<-round(cor(y_test,pred)^2,3)
r2
 [1]  0.563

round(sqrt(r2),4)
 [1]  0.7503
```

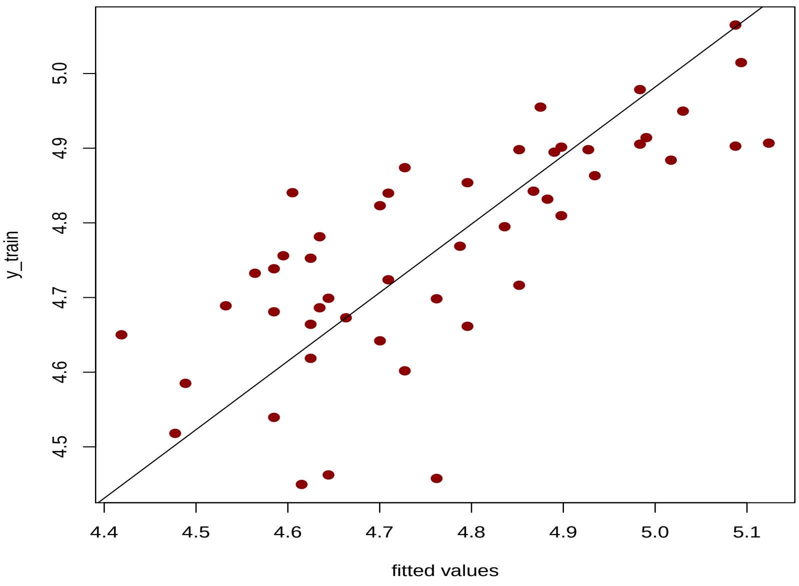


Figure 6.12: Predicted and actual values for test sample

Step 5 - Improving Model Performance

You may have noticed that we excluded T1 from our initial analysis. This allowed us to have coefficient estimates that concurred with our intuition and the empirical evidence. However, with such a small set of features, we really want to be able to use all of the relevant information in our model. One way to achieve that goal is via feature engineering.

First, load the features into the R object data_sample:

```
data_sample<-log(data.frame(data.matrix(
    Handaxes[,3:8])))
```

Feature engineering

Next, we engineer two new relative size features:

```
BB<-data_sample$B/data_sample$B1
TT<-data_sample$T/data_sample$T1
```

These are the ratio of size of B (T) relative to B1 (T1); and as such have a nice archaeological/ geometric interpretation.

Next, we combine the new features with data_sample, and then remove B,B1, T and T1:

```
data_sample<-cbind(data_sample,TT,BB)
data_sample$B<-NULL
data_sample$B1<-NULL
data_sample$T<-NULL
data_sample$T1<-NULL
```

Take a look at the first few observations:

```
round(head(data_sample),3)
     L1    B2    TT    BB
1 4.234 4.043 1.201 1.047
2 3.932 4.595 1.266 1.175
3 3.932 4.625 1.317 1.159
4 3.584 4.357 1.408 1.246
5 4.317 4.500 1.256 1.044
```

```
6  3.555  4.159  1.447  1.266
```

We now have four features, two engineered and two original.

Train, test set and fitting the model

Now, we are ready to set the train and test sets:

```
y_train<-y[train]
y_test<-y[-train]
data_train<-data_sample[train ,]
data_test<-data_sample[-train ,]
```

The model is fitted, as we saw earlier, using the `lm` function:

```
fit1= lm(y_train ~.  , data = data_train)
summary(fit1)

Call:
lm(formula = y_train ~ ., data = data_train)

Residuals:
     Min       1Q   Median       3Q      Max
-0.28583 -0.07197 -0.00593  0.06754  0.42859

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.14311    0.15351   0.932 0.351601
L1           0.37139    0.01758  21.127  < 2e-16 ***
B2           0.50833    0.02039  24.936  < 2e-16 ***
TT           0.15748    0.04636   3.396 0.000732 ***
BB           0.85478    0.08783   9.732  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
   0.1 ' ' 1

Residual standard error: 0.1089 on 545 degrees of
   freedom
Multiple R-squared:  0.7223,    Adjusted R-squared:
   0.7202
F-statistic: 354.3 on 4 and 545 DF,  p-value: < 2.2e
   -16
```

The estimated coefficients are all positive, and highly significant (tiny p-values), except the intercept. The joint F statistic is equal to 372.4 ($p < 2.2e\text{-}16$). This indicates that we can reject the null hypothesis that all of the coefficients are zero. Finally, we observe an adjusted R^2, higher than for our first model, at 0.72.

Figure 6.13 shows the diagnostic plots for `fit1`; and Figure 6.14 plots the estimates and confidence intervals of `fit (Model 1)` and `fit1 (Model 2)` for comparison.

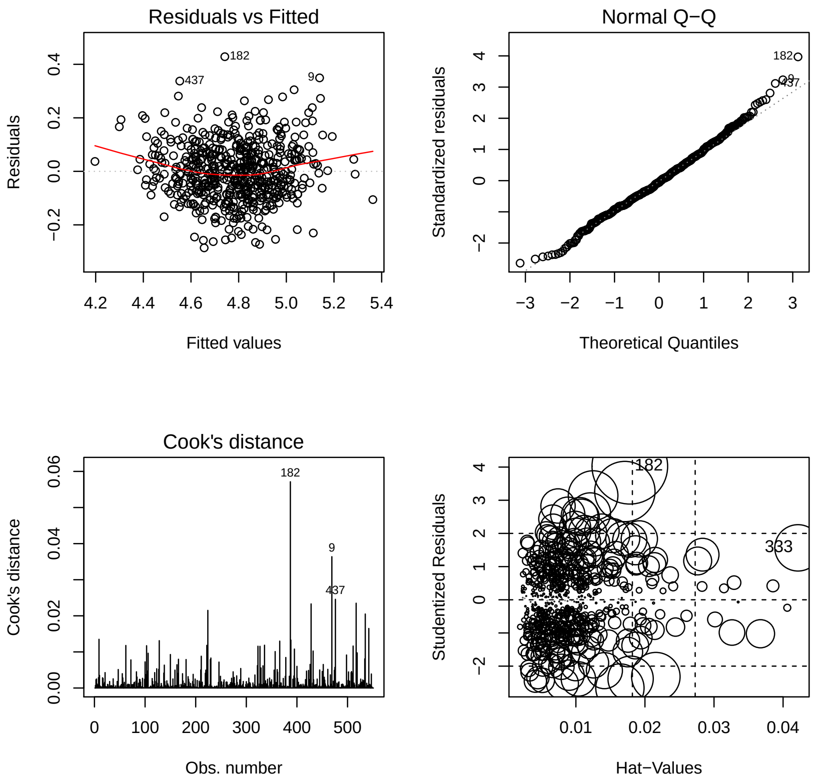


Figure 6.13: Diagnostic plots for `fit1`

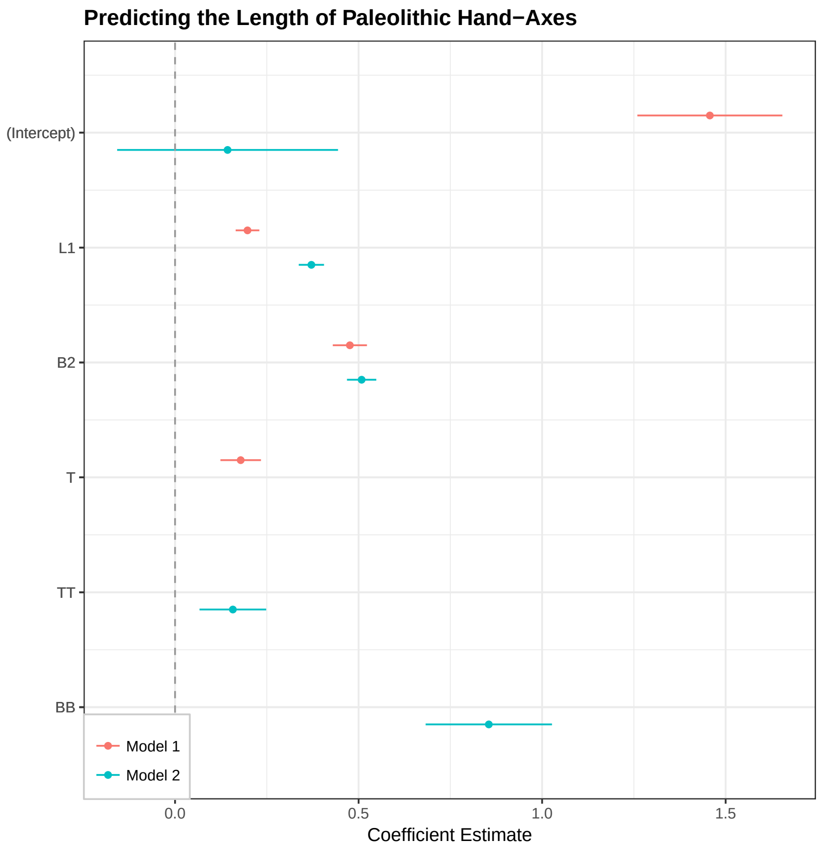


Figure 6.14: Estimates and confidence intervals of `fit (Model 1)` and `fit1 (Model 2)`

Test set performance

The training set correlation and R^2 statistics indicate an improved fit over our first model. How well does the feature engineered model perform on the test set?

```
pred1 <- predict(fit1, data_test)
round(cor(y_test,pred1),3)
[1] 0.817

round(cor(y_test,pred1)^2,3)
```

```
[1] 0.667
```

Quite a nice improvement in predictive performance. Figure 6.15 shows the predicted and actual values plot.

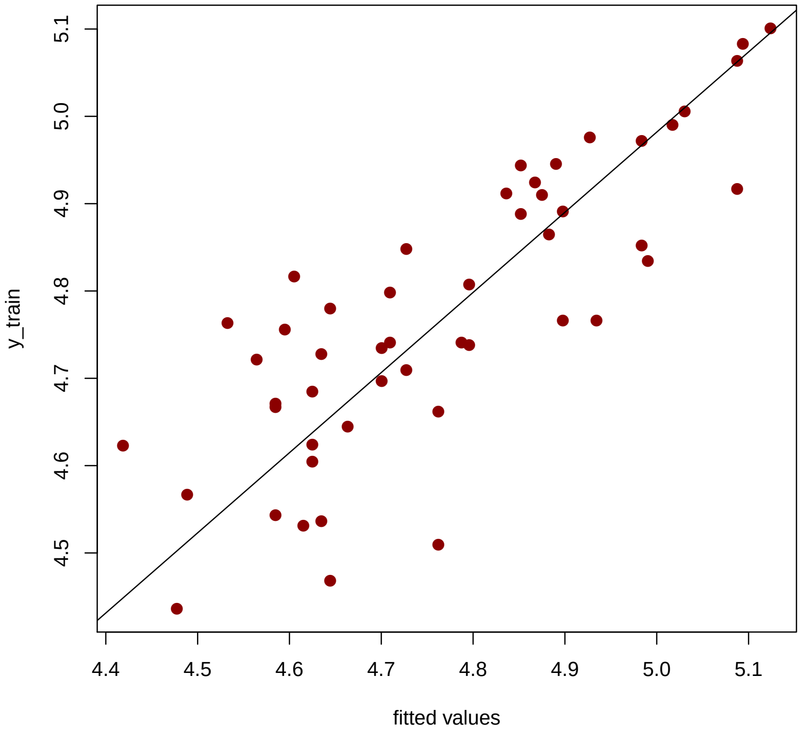


Figure 6.15: Test set predicted and actual values for `fit1`

Although the creation of a couple of engineered features seems like a small change, it improved the overall fit of the model. The great thing was that the features are interpretable. Often you need to be careful to design features with a useful interpretation. However, in pure performance driven domains like predicting the stock market or forecasting the number of visitors to a website, you can often be more creative.

Limitations of Linear Regression

Linear regression is the workhorse of empirical analysis. For almost all of it's limitations, there is an alternative type of regression model or estimation procedure. It forecasts the mean response (expected value) of the target variable. If you are interested in the median response (or other quantiles), you should use quantile regression. This available in the `quantreg` package.

Linear regression coefficient estimates are sensitive to outliers. If they are identified in you sample, they can be removed, or an alternative regression estimation technique used (i.e. robust regression).

Similar to discriminant analysis, the features are assumed to be independent. Highly correlated features should be removed from the sample, as they contain redundant information.

Summary

In this chapter, we learned about prediction using linear regression. It is a very large topic, and we only scratched the surface. However, we covered much of the essential theory, discussed several real-world applications and built step by step several models to predict the length of Paleolithic hand-axes. Such models are of great value to archaeologists and inform the debate on their potential uses.

In the next chapter, we explore a related regression model, but for a binary target variable - logistic regression.

Suggested Reading

- **Age-Friendly Cities:** Alma M. L. Au, Stephen C. Y. Chan, H. M. Yip, et al., "Age-Friendliness and Life Satisfaction of Young-Old and Old-Old in Hong Kong," Current Gerontology and Geriatrics Research, vol. 2017, Article ID 6215917, 10 pages, 2017. doi:10.1155/2017/6215917.

- **Foreign Direct Investment:** Yan, Xinfeng, and Kundan Pokhrel Majagaiya. "Relationship between Foreign Direct Investment and Economic Growth Case Study of Nepal." International Journal of Business and Management 6.6 (2011): 242.

- **Soil Particle Density:** Schjønning et al. "Predicting soil particle density from clay and soil organic matter contents." Geoderma 286 (2017): 83-87.

Other

- **Flint Axes:** John Frere. Account of Flint Weapons discovered at Hoxne in Suffolk. Archaeologia, Volume 13 (1800).

- **Safari Guide Peter Allison:** Peter Allison. 2009. Don't Look Behind You! A Safari Guide's Encounters with Ravenous Lions, Stampeding Elephants, and Lovesick Rhinos. Rowman & Littlefield.

- **Sir Francis Galton's Sweet Pea Experiment:** See Stanton, Jeffrey M. "Galton, Pearson, and the peas: A brief history of linear regression for statistics instructors." Journal of Statistics Education 9.3 (2001).

- **WHO:** World Health Organization. Global age-friendly cities: A guide. World Health Organization, 2007.

Chapter 7

Logistic Regression

LOGISTIC regression is a special type of regression where the target variable is binary. The features can be discrete or continuous. As with linear regression, it can be used for prediction, descriptive studies, and testing theoretical hypothesis. Machine learning is primarily concerned with accurate prediction, and so the first use is focus of this chapter. In this chapter, you will:

- Gain an understanding of the mechanics of logistic regression.
- Explore the idea of maximum likelihood.
- Study examples of its use in business (customer churn), marketing (opinion mining) and health research (disease classification).
- Build and assess several logistic regression models using R.

Logistic regression is one of the most widely used types of regression model. It will be a valuable addition to your machine learning tool kit.

Understanding Logistic Regression

Logistic regression is similar to linear regression except that the target variable is a binary. For example, an Ethologist might assign a crab to one of two classes (say female or male). A male might be coded as $y = 1$, and a female coded by $y = 0$.

Recall, in linear regression the target variable is related to the features via the linear relationship:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon \tag{7.1}$$

Suppose $p(y)$ is the probability that a crab is male (we could write $p(y = 1)$ but stick to the shorter notation).

To relate $p(y)$ to the features you might consider writing:

$$p(y) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \tag{7.2}$$

Unfortunately, this specification, can generate values for $p(y)$ from -∞ to ∞. We need a model that generates probabilities in the 0 to 1 range. This is not guaranteed to be the case if we use equation 7.2. Furthermore, linear regression assumes the values of y are normally distributed. In logistic regression y takes the values 0 or 1, so this assumption is clearly violated.

We need a more appropriate transformation. This can be achieved using the logistic regression model:

$$\ln\left(\frac{p(y)}{1 - p(y)}\right) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \tag{7.3}$$

It is a non-linear transformation of the linear regression model where the target variable is binary.

Odds Ratio

When we have a binary classification problem, we are typically interested in the probability that an observation belongs to a specific class. Statisticians often think of this in terms of odds.

For example, in a sample of crabs, the odds of being a male can be calculated as:

$$Odds = \frac{Number\,of\,Male\,Crabs}{Total\,Number\,of\,Crabs}$$

or, equivalently:

$$Odds = \frac{p(y)}{1 - p(y)}$$

The ratio $\left(\frac{p(y)}{1-p(y)}\right)$ is called the odds ratio.

Log odds ratio

The natural logarithm of the odds ratio is called the log odds ratio or logit. Let's investigate what an odds ratio of 1 implies:

$$\ln\left(\frac{p(y)}{1 - p(y)}\right) = 1$$

$$\Rightarrow p(y) = [1 - p(y)]$$

$$\Rightarrow p(y) + p(y) = 1$$

$$\Rightarrow p(y) = 0.5$$

In other words, an odds ratio equal to 1 implies the probability that $y = 1$ is 0.5.

A key point, is that logistic regression models the log odds ratio as a linear function of the features. Why is this so important? Well, in equation 7.2, we attempted to model $p(y)$ as a linear regression. However, this resulted in probabilities outside the [0, 1] range. The odds function takes values between 0 and ∞. When we take the natural log of the odds function, we get a range of values from -∞ to ∞. We can see this using the following R code:

```
p=seq(from =0,to=1, by=.1)
odd.ratio= p/(1-p)
odds.ratio=p/(1-p)
log.odds=log(odds.ratio)
tab=cbind(p,odds.ratio,log.odds)
round(tab,3)
        p odds.ratio log.odds
 [1,] 0.0      0.000     -Inf
 [2,] 0.1      0.111   -2.197
 [3,] 0.2      0.250   -1.386
 [4,] 0.3      0.429   -0.847
 [5,] 0.4      0.667   -0.405
 [6,] 0.5      1.000    0.000
 [7,] 0.6      1.500    0.405
 [8,] 0.7      2.333    0.847
 [9,] 0.8      4.000    1.386
[10,] 0.9      9.000    2.197
[11,] 1.0        Inf      Inf
```

We see from the above output that:

- If $p > 0.5$ the odds ratio > 1 , and the log odds ratio is positive.

- If $p < 0.5$, the odds ratio < 1, and the log odds ratio is negative.

Since p is a probability, we can see that the logistic regression model is constructed so that $0 \leq p \leq 1$. Indeed, from equation 7.3 as:

$$\alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ...\beta_k x_k$$

becomes very large, p approaches 1; and as:

$$\alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ...\beta_k x_k$$

becomes very small, p approaches 0; Furthermore, if:

$$\alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ...\beta_k x_k = 0$$

then $p = 0.5$.

Interpreting coefficients

We interpret $exp(\beta_i)$ as the effect of the independent variables or features on the odds ratio. For example, if we postulate the logistic regression:

$$\ln\left(\frac{p(y)}{1-p(y)}\right) = \alpha + \beta_1 x_1$$

and on estimation find that $\hat{\beta} = 0.963$, so that $exp(\hat{\beta}) = 1.999$. This implies a 1 unit change in x_1 would make the event $y = 1$ about twice as likely.

NOTE... ✍

In statistics textbooks you will often see $\ln\left(\frac{p(y)}{1-p(y)}\right)$ called the logit transform or simply logit.

The Logistic Curve

The logistic curve or sigmoid function, captures the relationship between a binary target variable and features. It is calculated as:

$$p(y) = \frac{\exp(\alpha + \beta x)}{1 + \exp(\alpha + \beta x)} \tag{7.4}$$

Because the relationship between $p(y)$ and x is nonlinear, the parameters α and β do not have a straightforward interpretation as they do in linear regression.

Figure 7.1 shows the logistic curve. It is bounded by 0 and 1, and therefore can be interpreted in terms of probabilities. The curve is symmetric about the point where $x = -\frac{\alpha}{\beta}$. In fact, the value of $p(y)$ is 0.5 for this value of x.

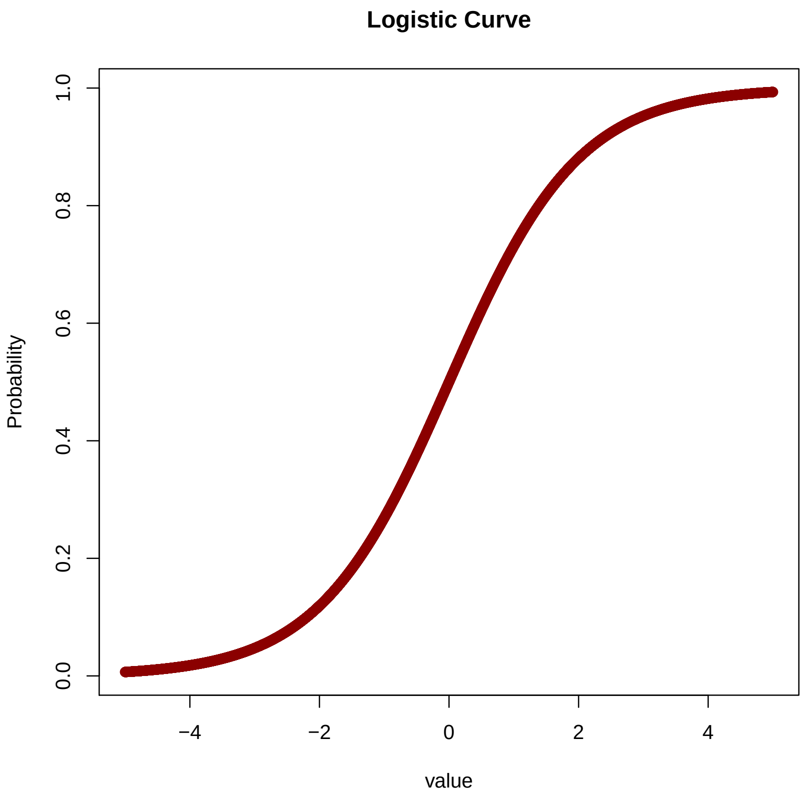


Figure 7.1: Logistic Curve

NOTE... ✍

The values of α and β, determine the location and spread of the logistic curve.

Relationship to logistic regression

If we were to run a logistic regression on the feature x we would specify:

$$\log\left(\frac{p(y)}{1-p(y)}\right) = \alpha + \beta x$$

In terms of the odds we can rewrite the above as:

$$\frac{p(y)}{1-p(y)} = \exp\left(\alpha + \beta x\right)$$

Of course, our interest is in $p(y)$; it is given by:

$$p(y) = \frac{\exp(\alpha + \beta x)}{1 + \exp(\alpha + \beta x)},$$

which is equation 7.4 we saw earlier.
The key points to note are that:

1. The log odds are linearly related to the features.

2. The relation between the features and $p(y)$ is nonlinear taking the S-shaped curve of Figure 7.1.

NOTE... ✍

Similar to linear regression, logistic regression assumes the features are independent. However, it does not make any assumptions about the probability distribution of the features.

Maximum Likelihood Estimation

In linear regression, the method of ordinary least squares can be used to estimate the regression coefficients. In logistic regression, we use a different approach called maximum likelihood estimation (MLE). MLE is a very general approach to obtain estimates of the parameters of probability models. Similar to ordinary least squares, the goal is to find the smallest possible deviance between the observed (y) and predicted values ($\hat{y}$).

The idea behind MLE is to choose the most likely values of the parameters α and β given, the observed sample, say $\{x_1, ...x_n\}$. Intuitively, the actual values these parameters take should depend in some way on the values observed in the sample data. This link is established via a probability model, which we denote by $f(x)$. The probability model is used to form the likelihood equation.

Probability model & likelihood equation

In logistic regression, the probability model is based on the binomial distribution, where:

$$f(x,p) = \begin{cases} \theta & if\ y_i = 1 \\ 1-\theta & if\ y_i = 0 \end{cases}$$

In other words, the probability of x_i being a male crab ($y_i = 1$) occurs with probability θ. And therefore:

$$p(y_i = 1) = \theta = \frac{\exp(\alpha + \beta x)}{1 + \exp(\alpha + \beta x)}$$

.

The likelihood equation is given by:

$$L = \prod_{i=1}^{n} \theta_i^{y_1} \left(1 - \theta_i\right)^{1-y_1}$$

Statisticians figured out a while back that is easier to work in natural logarithms. Taking the natural log of both sides and simplifying we get the log likelihood equation:

$$LL = \ln\ L = \sum_{i=1}^{n} \left[y_i \ln \theta_i + (1 - y_i) \ln(1 - \theta_i)\right]$$

Whilst ordinary least squares, in linear regression, minimizes the residual sum of squares; MLE in logistic regression minimizes model deviance. The MLE estimate of α and β, that minimize model deviance, are retrieved as the values $\hat{\alpha}$ and $\hat{\beta}$. These values maximize the probability of the observed data given the specified probability model.

NOTE... ✍

The maximum likelihood estimate of the parameters α and β are the values that maximizes the probability of the sample data $\{x_1, ...x_n\}$.

As a Bayes Classifier

A direct link to the Bayes classifier can be observed if we specify the logistic regression formula as a probability distribution of the class posterior probabilities. For a two class classification problem we have:

$$P(c_i = 1|x) = \frac{1}{[1 + \exp{[-\alpha - \sum_{i=1}^{n} \beta_i x_i]}]}$$

$$P(c_i = 0|x) = \frac{\exp{[-\alpha - \sum_{i=1}^{n} \beta_i x_i]}}{[1 + \exp{[\alpha + \sum_{i=1}^{n} \beta_i x_i]}]}$$

In this case we predict $c_i = 1$ if:

$$P(c_i = 1|x) > P(c_i = 0|x),$$

which is essentially the decision criteria used for the Bayes classifier.

Advantages of Logistic Regression

Since the log odds takes the values between -∞ to ∞, it allows the properties of linear regression to be exploited via a non-linear relationship between the target variable and features. In addition, probabilities (or odds ratios) can be directly calculated from the model parameters.

Unlike linear discriminant analysis, the features are not assumed to be normally distributed, or have equal variance in each class. This makes it more robust.

Practical Application of Logistic Regression

In this section, we discuss three examples of logistic regression use. The first is in business and involves measuring customer churn. The second is in marketing, where we discuss its use in opinion mining; The final illustration is in public health, where it is used for disease classification.

Customer Churn

If you work for a company that offers a service or product that requires a subscription, you no doubt have come across the problem of customer churn. Churn happens when a customer leaves their current product or service supplier, and moves to another competing firm that offers a similar product or service. Churn is a problem because customers are the principle asset of any company. Customer attrition results in direct loss of income. The business goal is to retain a greater proportion of current customers, and minimize the overall churn rate.

Teemu Mutanen built several logistic regression models to predict the customer churn of a retail banking company. The initial sample consisted of 151,000 customers and 75 features. After initial analysis, 12 features were selected for use in the logistic regression models. These included, customer age, length of time at the bank, volume of phone payments, number of ATM transactions, number of debit and credit card transactions, and customer salary.

A total of eight logistic regression models were assessed. The best performing model predicted customer churn with a 72% accuracy. Teemu concludes:

> "*The different models predicted the actual churners relatively well...The findings of this study indicate that, in case of logistic regression model,* [in order to maintain predictive performance] *the user should* [frequently] *update the model to be able to produce predictions with high accuracy.*"

NOTE... ✍

Customer retention is an important element of a subscription based business model. When a customer leaves, you lose a recurring source of revenue, and the marketing dollars you spent to acquire the customer.

Opinion Mining

Opinion mining evaluates the sentiment of individuals to assess their attitudes, emotions and opinions about a specific thing or set of events. It attempts to capture information on subjective impressions rather than measure hard core facts.

Opinion mining often gives a much clearer picture of public opinion than surveys or focus groups. For this reason, it is used by marketers to evaluate the success of an advertising campaign, politicians to measure voter sentiment on campaign issues, and educational establishments to understand student views.

A group of researchers, including Matthew England of Coventry University, built a logistic regression model to understand Arabic opinions on health services. The sample was collected over a six-month period from Twitter. Matthew comments that:

"*Twitter has a huge number of Arabic users who mostly post and write their tweets using the Arabic language. While there has been a lot of research on sentiment analysis in English, the amount of researches and datasets in Arabic language is limited.*"

A total of 2,026 tweets were collected. They were broken down into four topics as follows:

- Closing Hospital, 1009 tweets
- Resolving Health, 492 tweets
- Opinions about Health, 285 tweets
- Improving Health, 240 tweets

The tweets were manually assessed by three judges as either positive or negative. There were approximately 2.5 negative tweets for every positive tweet. This made the sample classes somewhat unbalanced.

The train and test sample split was adjusted several times. At each split the classification accuracy was assessed. For a training set that consisted of 60% of the data, the logistic regression model achieved 86.92% accuracy. For the training sample that consisted of 30% of the sample, the model achieved an accuracy of 88.32%.

NOTE... ✍

Opinion mining is also referred to as "*sentiment analysis*". In the United States, the Obama administration used sentiment analysis to gauge public opinion to policy announcements and campaign messages ahead of 2012 presidential election. President Obama was re-elected.

Zoonotic Disease Classification

A zoonotic disease is a disease spread between animals and people. Zoonotic diseases can be caused by viruses, bacteria, parasites, and fungi. Because some of the sicknesses in humans can be life threatening, health authorities track and report their occurrence.

Lyme disease and Rocky Mountain spotted fever, are two zoonotic diseases you can get from a tick bite. Predictive analytics expert, and trained biologist Stephen Jones built logistic regression models to classify both zoonotic diseases.

> "*Our objective was to determine if nonlinear modeling could improve explanatory power in describing the occurrence of 2 tick-borne diseases (Lyme disease (LD) and Rocky Mountain spotted fever (RMSF)) known to occur in Tennessee.*"

Medically diagnosed cases in the state of Tennessee formed the basis of the sample. The target variable was the post code (zip code). It took the value 1 if a disease was reported from that location, and 0 otherwise.

The logistic regression model contained six features. These were derived from demographic, geographic, landscape habitat characteristics, and clinical variables.

The Lyme disease logistic regression model achieved a classification accuracy of 72.8%; and the Rock mountain spotted fever model obtained 68.8% accuracy. These are great results, and Stephen wraps up the study by commenting that:

> "*Little work exists using more advanced nonlinear modeling techniques like those used in this study, and it is recommended to explore these options as they may provide better results than traditional* [linear] *regression-based approaches.*"

Example - Classifying Purple Rock Crabs

In Guy Smith's *Night of The Crabs*, Ian Wright and his fiancée Julie Coles, head to the beautiful Welsh seaside for a few days of rest and recreation. Close to the small village of Llandedr, they decide to take night a swim. Their destination, the aptly named Morchras (shell) island. When Julie disappears to the sound of frantic screams, Ian enters to water to rescue her. It is his last swim!

What happened? Ian and his fiancée, are the first of a ghastly sequence of innocent victims chased, trapped, brutally dispatched and devoured, by giant flesh eating crabs!

Whilst man eating crabs only inhabit a fictional Welsh coastline in the imagination of Guy Smith, the purple rock crab (leptograpsus variegatus) inhabits the actual shoreline of the North Island, Kermadec Islands, and many other islands dotted throughout the South Pacific. It is an athlete, being both fast and strong with a ferocious nip (try picking one up!). This spiny critter comes in two distinct colors - gorgeous purple-blue and radiant orange. In this section, we develop a logistic regression model to classify this spunky critter into male and female.

Step 1 – Collecting and Exploring the Data

The `crabs` data frame, in the `MASS` package contains observations on 200 purple rock crabs:

```
data("crabs",package="MASS")
```

For each crab, several features were measured. These include color, sex, and several morphological measurements detailed in Table 16.

Take look at the first few observations:

```
head(crabs)
```

Name	Description
`sp`	colour "B" or "O"
`sex`	Class (Male/Female)
`FL`	frontal lobe size (mm)
`RW`	rear width (mm)
`CL`	carapace length (mm)
`CW`	carapace width (mm)
`BD`	body depth (mm)

Table 16: Class and independent variables in `crabs` data frame

```
  sp sex index   FL  RW   CL   CW  BD
1  B   M     1  8.1 6.7 16.1 19.0 7.0
2  B   M     2  8.8 7.7 18.1 20.8 7.4
3  B   M     3  9.2 7.8 19.0 22.4 7.7
4  B   M     4  9.6 7.9 20.1 23.1 8.2
5  B   M     5  9.8 8.0 20.3 23.0 8.2
6  B   M     6 10.8 9.0 23.0 26.5 9.8
```

Notice, the column `index` is simply a count of the number of examples and therefore won't feature in our analysis.

Figure 7.2 shows the scatter plots, distribution and correlation between the attributes. Three things immediately pop out:

- The first is the linear nature of the relationship shown in the scatter plots between the attributes.

- Second, and related, is the high inter-attribute correlation. This is not surprising because the features are all essentially body size measurements.

- Third, is the apparent bell shaped symmetry of the measurements.

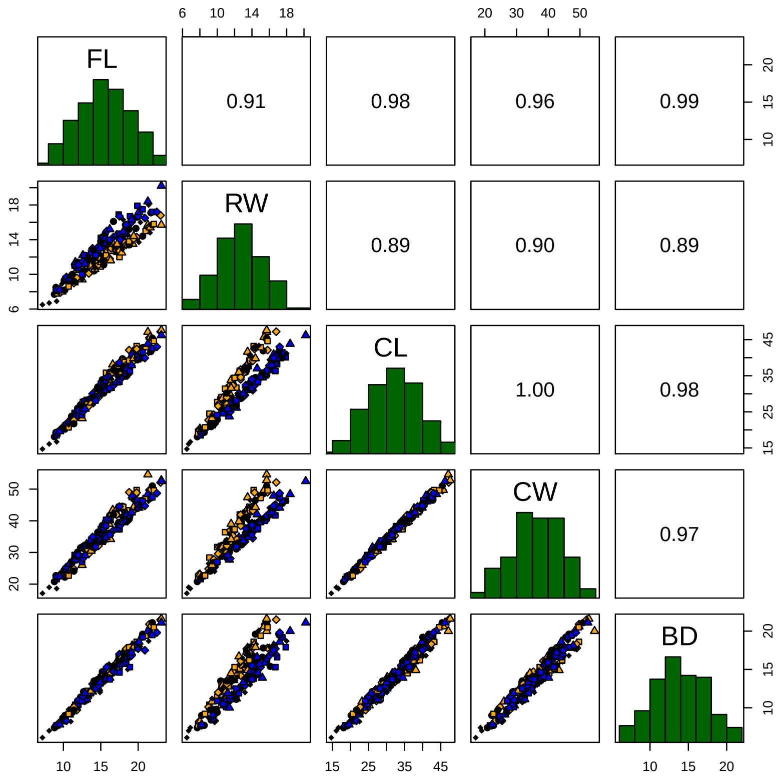


Figure 7.2: Pairwise scatter plot of `crabs` attributes.

The feature `sp` is a categorical variable. We should check to make sure there are no cells with zero values. We also would like this feature to be balanced relative to our target variable. This simply means we want approximately equal numbers of males and females in both color categories. The `xtabs` function can be used to investigate this. It creates a simple contingency table:

```
xtabs(~sex + sp, data = crabs)
   sp
sex  B  O
  F 50 50
  M 50 50
```

Wow! The observations are perfectly balanced. This is not normally the case with empirical data, but we'll take it.

Step 2 – Preparing the Data

First, we specify the R object `y` as our target variable. It contains the class variable `sex`. Take a look at the first few observations:

```
y<-crabs$sex
head(y)
[1] M M M M M M
Levels: F M
```

Next, we select, at random, 150 observations without replacement using the `sample` function. This will form the training set:

```
set.seed(2018)
N=nrow(crabs)
train<-sample(1:N,150,FALSE)
```

Take a look at the R object `train`, it contains the row numbers of the randomly selected observations:

```
head(train)
[1]  68  93  12  39 199  59
```

The first randomly selected observation is from row 68 in the crabs data set. The second, is from row 93, and so on.

Step 3 - Train Model using Train Set

Similar to linear regression, logistic regression assumes the features are independent. We saw earlier that there is high correlation between the features. Given this, we will fit our initial model using `FL`, `RW`, and the categorical variable `sp`.

The logistic regression can be fitted to the sample data using the `glm` function, with the `family` argument set to `binomial`:

```
fit1 <- glm(y[train] ~sp+FL+RW,
family=binomial(link='logit'),
data=crabs[train,])
```

The function takes the standard R formula, followed by the type of model (`logit`), and the sample data. The fitted model is stored in `fit1`.

Using the `summary` function

The `summary` function displays the result:

```
summary(fit1)
```

```
Call:
glm(formula = y[train] ~ sp + FL + RW, family =
    binomial(link = "logit"),
     data = crabs[train, ])

Deviance Residuals:
      Min          1Q      Median          3Q         Max
-2.17068    -0.16571    -0.00255     0.20631     2.14300

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)    8.777      2.059   4.263 2.02e-05 ***
sp            -3.919      1.104  -3.551 0.000384 ***
FL             3.725      0.722   5.160 2.47e-07 ***
RW            -5.111      0.967  -5.286 1.25e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
    0.1 ' ' 1

(Dispersion parameter for binomial family taken to be
     1)

     Null deviance: 207.917  on 149  degrees of
         freedom
Residual deviance:  58.703  on 146  degrees of
    freedom
AIC: 66.703

Number of Fisher Scoring iterations: 7
```

The output contains a lot of information. The first few lines provide details on the model. They serve to remind us what we did by providing details on the feature, sample and formula used to fit the model.

Deviance residuals

Next, the deviance residuals are reported. Like the residual in a linear regression model, they are a measure of model fit. Smaller absolute values indicate better fit. This part of the output shows minimum, quantiles, and the maximum of the deviance residuals for individual sample examples used to fit the model. The maximum deviance is 2.143, with a very small median value of -0.00255.

Estimated coefficients

The estimated coefficients are shown in the next part of the output. They indicate that `FL` influences crab sex positively, while `sp` and `RW` have a negative effect. The estimated values tell us the change in the log odds of the target variable for a one unit increase in a feature variable. As an example, for a one unit increase in `FL`, the log odds of being a male crab (versus female) increases by 3.725. However, for a one unit increase in `RW`, the log odds of being a male crab decreases by 5.11.

Statistical tests of individual features

The coefficient estimates are reported alongside their standard error (`Std. Error`), z-statistic (`z value`), and p-values (`Pr(>|z|)`). The statistical significance of estimated coefficients can be tested using the z-statistic or p-values. Our primary interest is in classification accuracy rather than descriptive analysis or hypothesis testing. However, it is interesting to notice the features are highly statistically significant. This is indicated by the symbol `***`. Of course, statistical significance tells us little about classification accuracy. But we can use it to confirm or shape our intuition. Be careful however, as it can also be very misleading.

Confidence intervals, using the standard errors, can be calculated via the `confint.default` function:

```
confint.default(fit1)
                   2.5 %     97.5 %
(Intercept)  4.741698 12.811630
sp          -6.082269 -1.756093
FL           2.310317  5.140640
RW          -7.006455 -3.216081
```

Since none of the confidence intervals straddle 0, we can have some empirically grounded certainty that the sign of the estimated coefficients captures the direction of the relationship of the features to the log odds of being a male crab.

Variable importance

The absolute value of the z-score is often used to measure variable importance. In this case, `RW` with an absolute value of 5.286, followed by `FL` with an absolute value of 5.160 are the most influential features. This is useful to know, and makes sense because both features are related to body size.

NOTE... ✍

In the text, we used the standard errors to calculate the confidence intervals. However, for logistic models, R reports the confidence intervals using the profiled log-likelihood function. To see these use:

```
confint(fit1)
```

Null and residual deviance

The residual deviance is analogous to the residual sum of squares of a linear regression model. Lower values indicate better fit. It takes a value of 58.703.

The null deviance reports how well the target variable is predicted by a model that includes only the intercept. We would expect our model to do better than this. In this case it does, as the null deviance = 207.917. This implies our model has reduced the deviance by just over 149 points.

AIC and Fisher Scoring

The following two items are also reported via the `summary` function:

- The Akaike Information Criterion (AIC) is a measure of the relative quality of statistical models. It is only useful for comparing models.
- The `"Number of Fisher Scoring"` iterations simply tells you how many iterations were needed to fit the model by maximum likelihood.

Step 4 - Evaluate Model Performance

Our initial read of the statistics of the model suggests it fits the data well. In this section, we cover several metrics we can use to help evaluate model performance.

Using the `anova` function

How well our model fits, depends on the difference between the model and the observed data. One approach to evaluate this is to use the `anova` function:

```
anova(fit1, test="Chisq")
```

```
Analysis of Deviance Table

Model: binomial, link: logit

Response: y[train]

Terms added sequentially (first to last)


     Df Deviance Resid. Df Resid. Dev Pr(>Chi)
NULL                   149    207.917
sp    1    0.433       148    207.485   0.5105
FL    1    0.533       147    206.951   0.4652
RW    1  148.248       146     58.703   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
   0.1 ' 1
```

The `anova` function adds the features in the order given in the model formula (left to right). Hence, `sp` appears first followed by `FL` and `RW`.

Analyzing the table, we observer a small drop in deviance when adding each `sp` and `FL`. For example, adding `sp`, reduces the deviance from the Null model's value of 207.917 to 207.485. This is a tiny drop. We see a similar pattern for adding in FL. In this case, the model deviance drops by 0.533 to 206.951.

The good news, is that at least the addition of these two variables moves the model deviation in the right directions (downward). However, as indicated by the large p-value (`Pr(>Chi)`) on both `sp` and `FL`, the change in not statistically significant. This indicates the model without these variables explains approximately the same amount of variation. Fortunately, adding the feature, `RW` leads to a significant reduction in deviance of over 148 points. A highly significant p-value here, supports the importance of this feature.

Pseudo R^2 statistic for logistic regression

The pseudo-R^2 is a useful goodness-of-fit metric for logistic regression. Similar to the traditional R^2 statistic, it takes a value between 0 and 1. It is calculated as:

$$pseudo\ R^2 = 1 - \frac{model\ deviance}{Null\ deviance} = 1 - \frac{58.703}{207.917} \approx 0.717$$

The closer to 1 is the metric, the more useful are the features in predicting the target variable. In statistical language, it is more a measure of effect size than overall fit. In any case, the value of 0.717 indicates that the model is useful for predicting crab sex.

Model discrimination

The discrimination of a model – that is, how well the model separates male from female crabs - can also be assessed using the area under the receiver operating characteristic curve (AUC). It uses two metrics, Specificity and Sensitivity.

Specificity is a measure of how often the model predicts "female" (y = 0) when the actual observation is "female crab". It is calculated as:

$$Specificity = \frac{True\ Negatives}{Total\ Negatives}$$

Sensitivity or true positive rate measures when it's actually "male", how often does the model predict "male". It is calculated as:

$$Sensitivity = \frac{True\ Positives}{Total\ Positives}$$

Specificity and Sensitivity are often combined via a Receiver Operating Characteristic Curve (ROC). The ROC visually measures how well the predictive model separates the data into positives and negatives.

The graph of Figure 7.3 shows the ROC curve for a perfect fit. In this case, the sensitivity (Specificity) take the value of 1 for all possible ranges of specificity (Sensitivity).

The opposite extreme is where the model is no better than random. In this case, as shown in Figure 7.4, the ROC lies along a diagonal line. The random predictor is commonly used as a baseline against which to assess a potential model. In practice, a model will generate a ROC curve somewhere in between Figure 7.3 and Figure 7.4. Figure 7.5 shows a typical result. Notice that as Sensitivity increases, Specificity decreases.

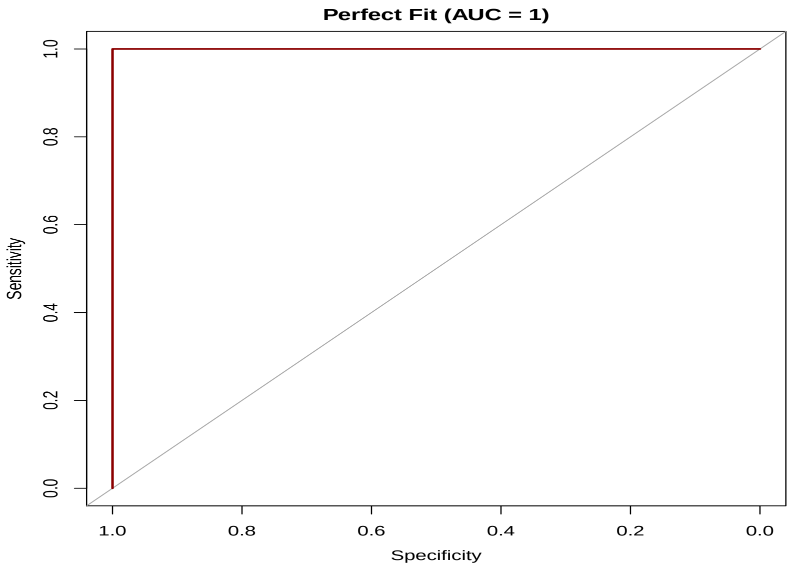


Figure 7.3: ROC for a perfect fit

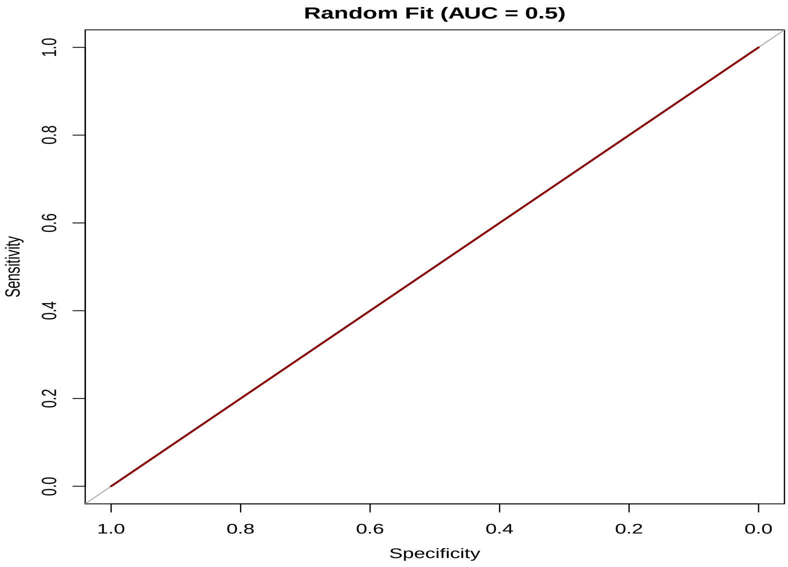


Figure 7.4: ROC for a random fit

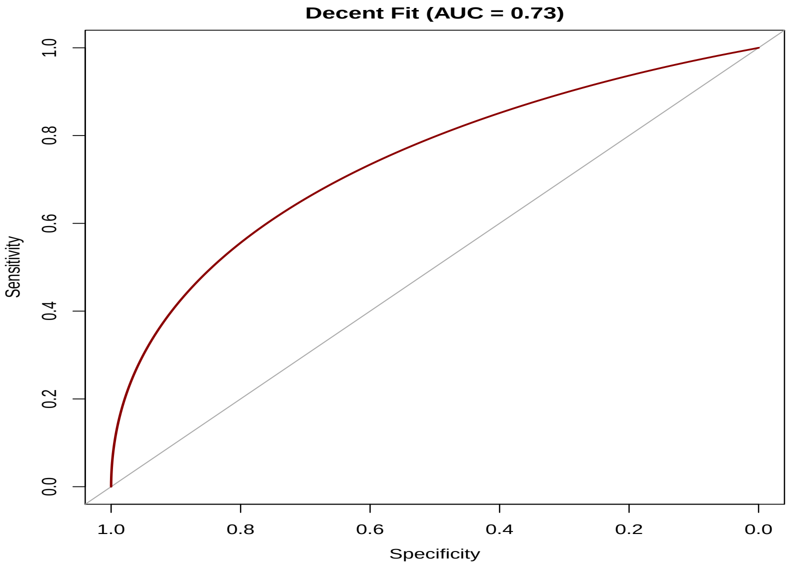


Figure 7.5: ROC for a typical fit

The overall accuracy of a model can be measured by the area under the ROC curve (AUC). An area of 1 represents a perfect fit (Figure 7.3); an area of 0.5 represents a random fit (Figure 7.4). AUC therefore measures discrimination, that is, the ability of a model to correctly classify positive and negative examples.

We can use the `pROC` package to calculate the AUC for `fit1`. First, we use the `predict` function to store the fitted probabilities in the R object `pred1`. The plot function is then called to visualize the contents of `pred1`:

```
pred1 <- predict(fit1, type="response")
plot(pred1)
```

You should see Figure 7.6, which confirms the values of `pred1` lie between 0 1n 1.

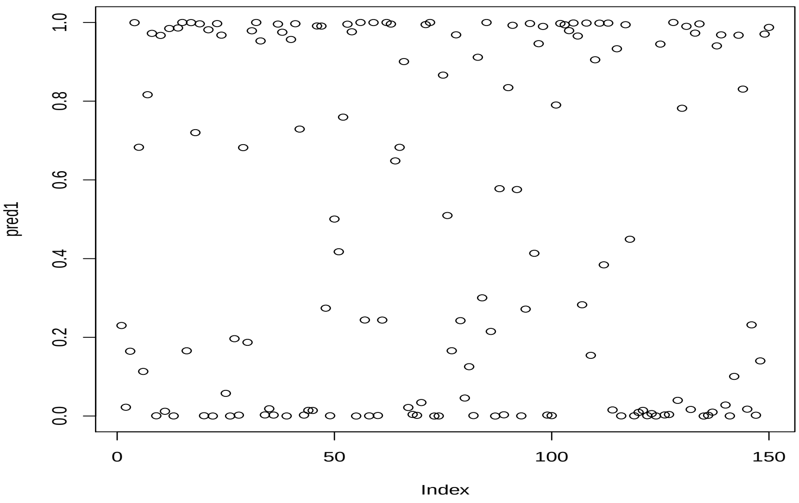


Figure 7.6: Plot of probabilities in `pred1`

Take a quick look at the probabilities in `pred1` using the head function:

```
round(head(pred1,5),2)
  68   93   12   39  199   59
0.23 0.02 0.16 1.00 0.68 0.11
```

The first observation has a probability of 0.23, the second observation an estimated probability of 0.02. It we use a cutoff probability of greater than 0.5 to indicate a male crab, then the first three observations would be classified as female, and the fourth and fifth observation male.

The next step is to use the probabilities in `pred1` indicate whether the prediction is a male or female. We use a cutoff probability greater than 0.5 to indicate a male crab:

```
pred1_train<- ifelse(pred1>0.5, 1, 0)
```

The labels in `y[train]` also need to be converted to binary:

```
y_train<- ifelse(y[train]=="M", 1, 0)
```

Now call the `roc` function from the package `pROC` to display the area under the curve statistic:

```
library(pROC)
(roc <- roc( y_train,pred1_train))

Call:
roc.default(response = y_train, predictor =
    pred1_train)

Data: pred1_train in 76 controls (y_train
  0) < 74 cases (y_train 1).
Area under the curve: 0.9067
```

The train set data gave a value of 0.9067, indicating that the model discriminates well. The confidence interval can be called using the `ci` function:

```
ci(roc)
95% CI: 0.8598-0.9535 (DeLong)
```

It is covers a narrow range. The lower value at 0.86 also indicates good discriminative performance of the model.

To plot the ROC curve, call the `plot` function:

```
plot(roc)
```

Figure 7.7 shows the resultant plot.

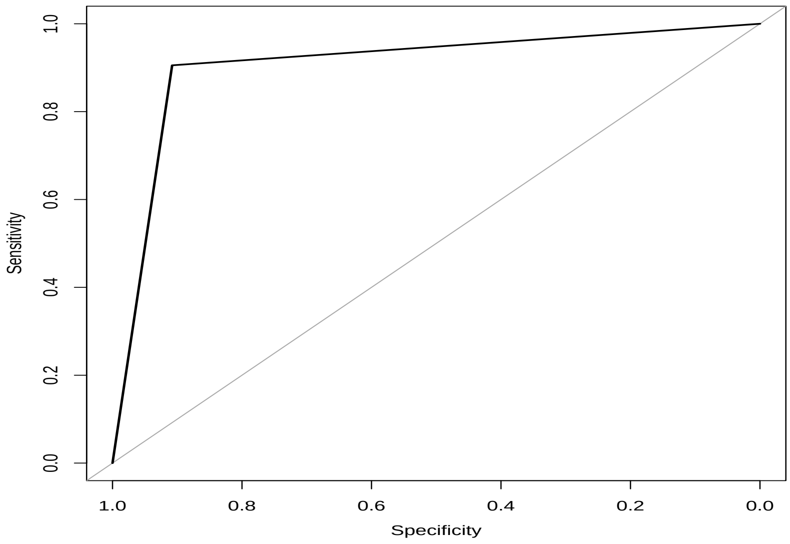


Figure 7.7: ROC for training sample of `fit1`

Classification accuracy

The previous metrics have given us a solid perception on the quality of the model. But how did it perform in terms of classification accuracy? Let's calculate the confusion metric using the `table` function. For the training sample we have:

```
predclass1 <- factor(ifelse(pred1>0.5,
"M", "F"))
table(y[train],predclass1)
   predclass1
     F  M
  F 69  7
  M  7 67
```

Of the 150 observations in the training sample, the model incorrectly classifies 7 male, and 7 female crabs. This leads to an overall accuracy of:

```
round(1-mean(predclass1 != y[train]),3)
[1] 0.907
```

Not bad, but remember this is the training set! We are interested in how the model performs on new observations. To see this we need to calculate the confusion matrix for the test set. But first, we transfer the data in `crabs` to the R object `x`:

```
x<-crabs
x$sex<-NULL
x$index <-NULL
```

Notice we remove `index`, and `sex`. The R object `x` only contains the features. Take a look using the `head` function:

```
head(x)
     FL  RW   CL   CW  BD
1   8.1 6.7 16.1 19.0 7.0
2   8.8 7.7 18.1 20.8 7.4
3   9.2 7.8 19.0 22.4 7.7
4   9.6 7.9 20.1 23.1 8.2
5   9.8 8.0 20.3 23.0 8.2
6  10.8 9.0 23.0 26.5 9.8
```

Yep, all is as expected.

Calculating test set accuracy

Following similar steps to those discussed earlier, we use the `predict` function, followed by a probability cutoff factor of 0.5:

```
pred1_test <- predict(fit1,
newdata=x[-train,],
type="response")
```

```
predclass1_test <- factor(ifelse(pred1_test
   >0.5,"M", "F"))
```

Now, calculate the confusion matrix, and overall class accuracy:

```
table(predclass1_test, y[-train])

predclass1_test  F  M
              F 22  2
              M  2 24
1-mean(predclass1_test != y[-train])
[1] 0.92
```

The model appears to generalize well, with similar classification performance as the training set. In this case, 2 male and 2 female crabs are misclassified, with an overall classification accuracy of 92%.

Step 5 - Improving Model Performance

The negative coefficient value of -6.08 on `sp` is a little troublesome. It is difficult to interpret in a meaningful way. The color of a crab is likely determined by the environment rather than gender. Even if the value were positive, we would have difficulty explaining it. For now, let's drop `sp` from our analysis:

```
x$sp <-NULL
```

Dealing with high correlation

Take another look at Figure 7.2. It shows the scatter plots, distribution and correlation between the attributes. What stands out is the very high correlation between the attributes. This is not surprising as they are all measurements related to body size. However, it violates the assumption that features are independent. What to do? One solution is to use the principal components.

I first came across PCA though the classic 1972 and 1973 papers of the University of Kent at Canterbury scholar I. T. Jolliffe; the technique has been a permanent addition to my data science toolkit ever since. Principal component analysis (PCA) is a statistical procedure that transforms correlated features into several uncorrelated variables called principal components. Each principal component is a linear combination of the original variables.

The goal of principal components analysis is to explain the maximum amount of variance with the fewest number of principal components. The first principal component accounts for as much of the variability in the data as possible. The second component accounts for as much of the remaining variability as possible, and so on.

The principal components are orthogonal, in other words they have zero correlation with each other. Because they contain the same information as the original features, we can use them in place of our features in the logistic regression model.

NOTE... ✍

In PCA, linear projections ordered by variance are computed from eigenvectors of the data covariance matrix.

The `prcomp` function

The `prcomp` function calculates the principal components:

```
pca <- prcomp(data.matrix(x),
center = TRUE,
scale. = TRUE)
```

PCA components are linear combinations of the original features. Let's look at the PCA coefficient weights:

```
round(pca$rotation,3)
```

```
     PC1    PC2    PC3    PC4    PC5
FL 0.452  0.138  0.531  0.697  0.096
RW 0.428 -0.898 -0.012 -0.084 -0.054
CL 0.453  0.268 -0.310 -0.001 -0.792
CW 0.451  0.181 -0.653  0.089  0.575
BD 0.451  0.264  0.443 -0.707  0.176
```

The first component (`PC1`), a linear combination of the attributes, is composed of an (almost) equally weighted combination of the features. Interpreting components can often be difficult. However, `PC1` appears to be a proxy for body size.

For crab i, the `PC1` score would be calculated as:
$\text{PC1}_i = 0.452 \times \text{FL} + 0.428 \times \text{RW} + 0.453 \times \text{CL} + 0.451 \times \text{CW} + 0.451 \times \text{BD}$

We can check to ensure the components are uncorrelated using the `cor` function:

```
round(cor(pca$x),2)
    PC1 PC2 PC3 PC4 PC5
PC1   1   0   0   0   0
PC2   0   1   0   0   0
PC3   0   0   1   0   0
PC4   0   0   0   1   0
PC5   0   0   0   0   1
```

As expected the components are uncorrelated. This allows us to use them as features in our logistic regression model.

Proportion of variation explained

The proportion of variation explained by each principal component can be viewed using the `summary` function:

```
summary(pca)
```

```
Importance of components:
                          PC1     PC2     PC3     PC4     PC5
Standard deviation     2.1883 0.38947 0.21595 0.10552 0.04137
Proportion of Variance 0.9578 0.03034 0.00933 0.00223 0.00034
Cumulative Proportion  0.9578 0.98810 0.99743 0.99966 1.00000
```

The first component explains 95.8% of the variation in the feature data, and the second component 3%. Since, these two components account for over 98% of the variation in the data, we will use them as our independent variables in the logistic regression model. To do this we create a new R object called `pca_data`:

```
pca1<-pca$x[,1]
pca2<-pca$x[,2]
pca_data<-cbind.data.frame(pca1,pca2)
```

Use of two components as features

The objects `pca1` and `pca2`, contain the PCA scores for the first and second component. They are combined into the object `pca_data`. Recall, PCA components are linear combinations of the original features, and therefore have the same number of rows as the original data:

```
nrow(x)
[1] 200

nrow(pca_data)
[1] 200
```

Fitting the model

We fit the model using the training data. For ease of illustration, we transfer this data into `pca_train`, and then fit the model:

```
pca_train<-pca_data[train,]

fit2 <- glm(y[train]~.,data=pca_train,
   family="binomial")
```

Now, take a look at estimated parameters:

```
round(fit2$coefficients,3)
(Intercept)          pca1         pca2
      0.572         0.328       21.217
```

It appears that both `pca1` and `pca2` influence crab sex positively. For a one unit increase in `pca1`, the log odds of being a male crab (versus female) increases by 0.328. However, notice for `pca2` the effect size at 21.217 is many multiple times larger than that of `pca1`. Such a large difference cannot be ignored. Let's take a look at the statistics, in terms of the confidence interval of the estimates:

```
confint.default(fit2)
                 2.5 %     97.5 %
(Intercept) -0.6523838  1.7958668
pca1        -0.1403309  0.7970819
pca2        10.9193396 31.5146555
```

The confidence interval of `pca1` straddles 0, indicating that it is not statistically significant. This means the sign of the estimated coefficient on `pca1` may not accurately capture the direction (and size) of the relationship to the log odds of being a male crab. For now, we keep both features.

Model deviance

The deviance of the null model and `fit2` are given by:

```
round(fit2$null.deviance,2)
[1] 207.92


round(fit2$deviance,2)
[1] 31.14
```

Let's put on our Statistician hat for a moment. A simple way to evaluate the overall performance of the model is to look at the null deviance and residual deviance. Null deviance indicates how well the class is predicted by a model with nothing but the intercept. We would expect such a model to be a poor classifier. Despite `pca1` not being statistically significant, the null deviance of `fit2` at 31.14 is considerably lower than for the null model. Adding in our predictors decreased the deviance by just over 176 points.

You may also have noticed that `fit2` deviance is lower than for `fit1`. This indicates `fit2` has a smaller prediction error. Let's see, if that is reflected in the confusion matrix for the training sample:

```
pred2 <- predict(fit2, type="response")
predclass2 <- factor(ifelse(pred2>0.5,
"M", "F"))

table(predclass2, y[train])

predclass2  F  M
          F 72  4
          M  4 70
1-mean(predclass2 != y[train])
[1] 0.9466667
```

The confusion matrix indicates the model misclassified a total of 8 examples, yielding an overall classification accuracy of 94.6%.

Test set accuracy

We pass the test set pca data to the R object `pca_test`, then follow the procedure we have already discussed:

```
pca_test<-pca_data[-train,]
pred2_test <- predict(fit2,
newdata=pca_test,
type="response")

predclass2_test <- factor(ifelse(pred2_test
   >0.5,
"M", "F"))
```

```
table(predclass2_test, y[-train])

predclass2_test   F   M
              F  22   0
              M   2  26
1-mean(predclass2_test != y[-train])
[1] 0.96
```

Yep, the model improves classification performance for both the train and test set. Figure 7.8 plots the logistic regression decision boundary which confirms the overall good fit.

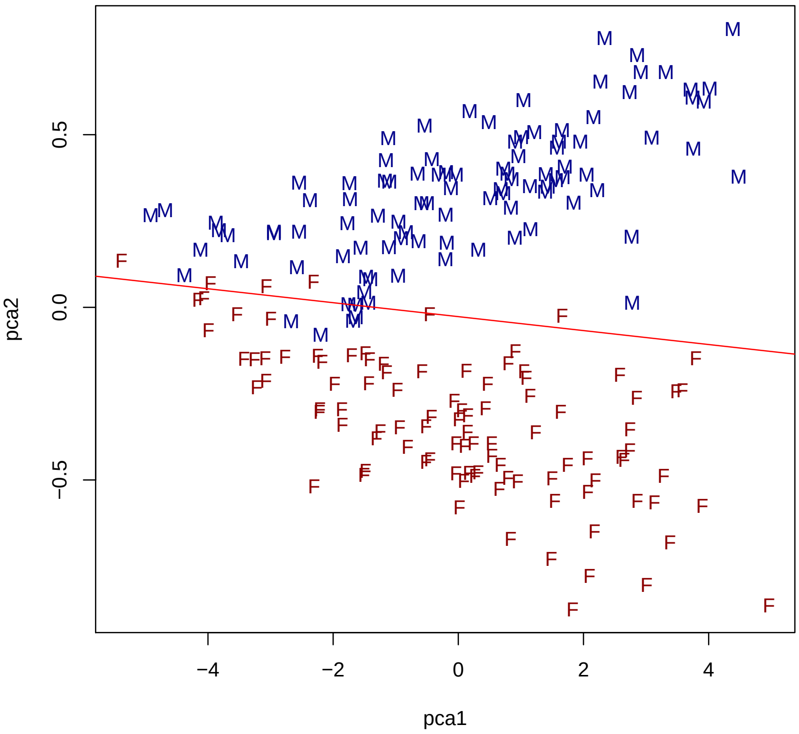


Figure 7.8: Logistic regression decision boundary for `crabs`.

Using statistics to enhance performance

Earlier we saw that `pca1` is statistically insignificant. This implies we may be able to remove it from the model. Let's drop it, re-estimate the model, and calculate the train set confusion matrix and classification accuracy:

```
fit3 <- glm(y[train]~ pca2,
data=pca_train,
family="binomial")
pred3 <- predict(fit3,
type="response")

predclass3 <- factor(ifelse(pred3>0.5,
"M", "F"))

table(predclass3, y[train])

predclass3   F  M
          F 71  4
          M  5 70

1-mean(predclass3 != y[train])
[1] 0.94
```

The train set classification accuracy is similar to that of `fit2`. How did it do in terms of test set accuracy?

```
pred3_test <- predict(fit3,
newdata=pca_test,
type="response")

predclass3_test <- factor(ifelse(pred3_test
    >0.5,
"M", "F"))

table(predclass3_test, y[-train])
```

```
predclass3_test   F  M
               F 23  0
               M  1 26
1-mean(predclass3_test != y[-train])
[1] 0.98
```

It misclassified 1 observation and therefore delivers a classification accuracy of 98%.

Limitations of Logistic Regression

Logistic regression is a discriminative classifier. However, in practice, generative classifiers such as naive Bayes often outperform discriminative classifiers such as logistic regression.

Well over a decade ago scholars Andrew Ng and Michael Jordan studied the error properties of logistic regression and naive Bayes models. They found that naive Bayes reaches its asymptotic error bound much faster than the discriminative logistic regression classifier. However, as the sample size grows larger, and larger, logistic regression outperforms the naive Bayes classifier. The scholars illustrated the result for 15 real-world data-sets.

It turns out the generative naive Bayes model reaches its asymptotic bound at a rate $O(logN)$, whilst the discriminative logistic model approaches it bound at a rate of $O(N)$. The practical implication of this is that the naive Bayes model reaches its asymptotic solution for a much smaller data sample than the logistic model. In other words, if you have a large sample try logistic regression. If you have a small sample try naive Bayes.

Summary

Logistic regression is a tool that you will frequently draw upon. It delivers outstanding performance in a range of tasks. In this chapter, you have gained an understanding of how logistic regression works, worked through a number of practical application; and have discovered how to estimate it in R, analyze critical statistics, and use it for classification tasks.

In the next chapter, we outline another popular classification and prediction tool - Support Vector Machines.

Suggested Reading

- **Customer Churn:** Mutanen, Teemu. "Customer churn analysis–a case study." Journal of Product and Brand Management 14.1 (2006): 4-13.
- **Opinion Mining:** Alayba, Abdulaziz M., et al. "Arabic Language Sentiment Analysis on Health Services." arXiv preprint arXiv:1702.03197 (2017).
- **Zoonotic Disease:** Stephen Jones, William Conner, and Bo Song, "Spatially Explicit Nonlinear Models for Explaining the Occurrence of Infectious Zoonotic Diseases," ISRN Biomathematics, vol. 2012, Article ID 132342, 12 pages, 2012. doi:10.5402/2012/132342

Other

- **Logistic Regression and Naive Bayes:** See the paper by Andrew Ng and Michael Jordan who developed the theory and ran 15 experiments to show this to be the case in practice. The paper is entitled "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes." Advances in neural information processing systems 14 (2002): 841.
- **Principal Component Analysis:** I strongly recommend you get a copy and read thoroughly both of the following papers. They are pure solid gold:

- Jolliffe, Ian T. "Discarding variables in a principal component analysis. I: Artificial data." Applied statistics (1972): 160-173.
- Jolliffe, Ian T. "Discarding variables in a principal component analysis. II: Real data." Applied statistics (1973): 21-31.

Chapter 8

Support Vector Machines

THE support vector machine (SVM) is a supervised machine learning algorithm which can be used for both regression and classification. It works by constructing hyperplanes in a multidimensional space. The hyperplanes separate the sample data into different groups.

In this chapter, you will:

- Intuitively understand the role of the input space and feature space in Support Vector Machines.
- Understand the role of the decision hyperplane, Support Vectors, Margin and Slack.
- Review several successful real world applications of Support Vector Machines.
- Accumulate practical hands on experience as you develop a SVM to classify diabetes.

Whilst the technical details surrounding Support Vectors Machines can be challenging, an intuitive understanding of how they work is not. Furthermore, since they can be easily deployed in R, they should be a part of your machine learning toolkit.

Understanding Support Vector Machines

Classification often requires nonlinear decision boundaries to separate examples into different groups. For example, in Figure 8.1, the classification decision boundary that best separates the observations is nonlinear.

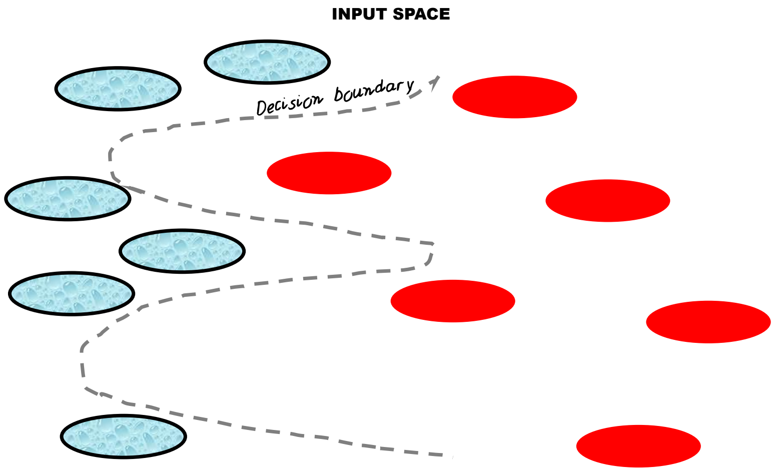


Figure 8.1: Nonlinear boundary required for correct classification

The core idea of SVMs is that they construct an optimal hyperplane for linearly separable data in a multidimensional space using a kernel function. Figure 8.2 illustrates how SVMs achieve this. The left side of the figure represents the original sample (known as the input space) which is mapped to a feature space of a higher dimension before performing a linear classification in that higher dimensional space.

The process of rearranging the objects for optimal separation is known as transformation and occurs through the use of a kernel function. A kernel function provides a nonlinear mapping from inputs x to feature vectors $\Phi(x)$. In mapping from the input space (left part of Figure 8.2) to the feature space (right part of Figure 8.2) the mapped objects can be separated by a straight line (are linearly separable) in the new space.

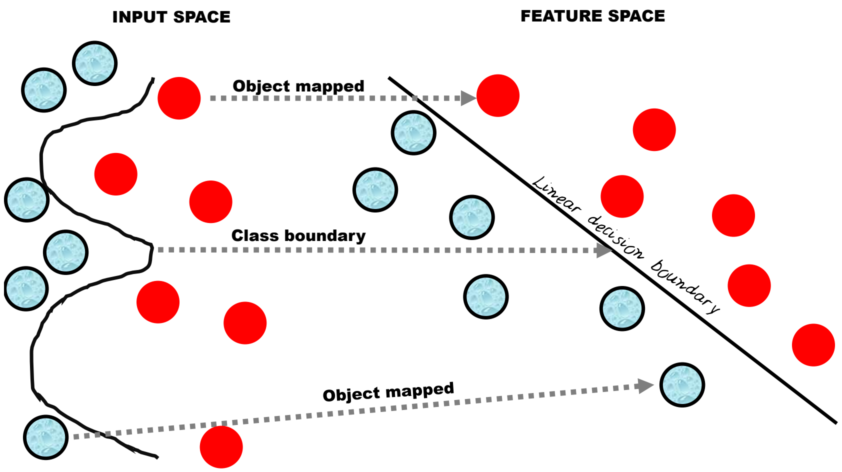


Figure 8.2: Mapping from input to feature space in an SVM

NOTE... ✍

Classification algorithms based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers.

Explaining the Optimal Hyperplane

The SVM finds the decision hyperplane leaving the largest possible fraction of points of the same class on the same side, while maximizing the distance of either class from the hyperplane. As illustrated in Figure 8.3, the support vectors are the data points that lie closest to the optimal hyperplane. These are the most difficult points to classify and directly influence the location of the decision boundary.

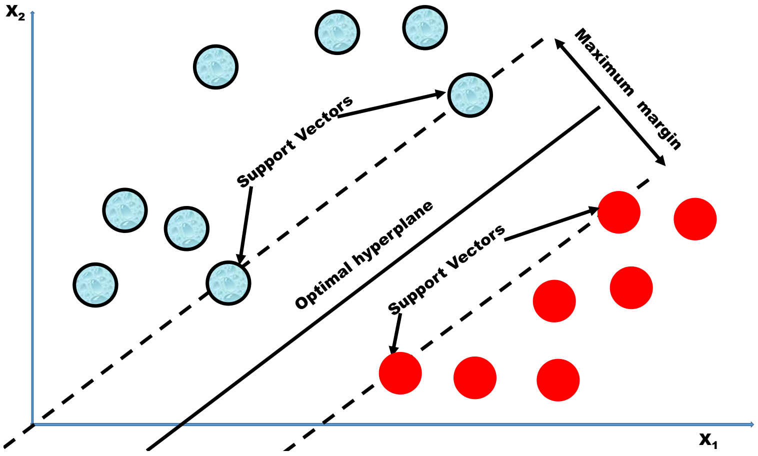


Figure 8.3: Support Vector Machine hyperplane

Maximizing the margin hyperplane

To construct an optimal hyperplane, SVM employs an iterative training algorithm to maximize the margin around the separating hyperplane using the "*difficult*" to classify points close to the decision boundary for support.

Let's look at how this is achieved. Consider a training sample consisting of N patterns $\{(x_1, y_1, ..., (x_N, y_N)\}$, where x is the feature vector, and target $y_i \in \{-1, +1\}$ with corresponding binary labels. The SVM parameters are determined by maximizing the margin hyperplane:

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i \cdot x_j) \tag{8.1}$$

subject to the constraints:

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \; and \; 0 \leq \alpha_i \leq \mathcal{C} \tag{8.2}$$

where $K(x_i, x_j)$ is the kernel function used to map the data from the input space to the feature space; the item $\mathcal{C}$ is a cost/ slack parameter.

NOTE... ✍

There are numerous kernels. Several you will come across in your own work include the Polynomial, Gaussian Radial Basis, Hyperbolic tangent, and linear.

Slack parameter

The variable $\mathcal{C}$, known as the slack parameter, serves as the cost parameter that controls the trade-off between the size of the margin and the classification error. If no slack is allowed (often called a hard margin) and the data are linearly separable, the support vectors are the points which lie along the supporting hyperplanes. In this case, all of the support vectors lie exactly on the margin.

In practice, for complex classifications a hard margin may not yield useful results, and a "*soft*" margin is used. In this case, some proportion of data points are allowed to remain inside the margin. The slack parameter is used to control this proportion. A soft margin results in a larger classification error on the training data set. However, it reduces the likelihood of over-fitting.

Number of support vectors

The total number of support vectors depends on the amount of allowed slack and the distribution of the data. If a large amount of slack is permitted, there will be a larger number of support vectors than the case where very little slack is permitted. Fewer support vectors means faster classification of test points. This is because the computational complexity of the SVM is linear in the number of support vectors.

Advantages of Support Vector Machines

Support vector machines are based on a theoretical model of learning; and they therefore come with certain theoretical guarantees regarding performance. They do not suffer from the curse of dimensionality, and have been popular in text classification problems where very high-dimensional spaces are the norm. Unlike neural networks, which they are often compared with, they are not susceptible to getting trapped in local minima.

Practical Application of Support Vector Machines

One of the best ways to see the value of a complex algorithm such as SVM, is to explore how they have been applied in practice. This section touches on three applications. The first involves stock price prediction. This is followed by an example of breast cancer classification. Finally, SVM flexibility is illustrated in their successfully deployment for signature authentication.

Forecasting Stock Market Direction

Researchers Huang et al use a support vector machine to predict the direction of weekly changes in the NIKKEI 225 stock market index. The index is composed of 225 stocks of the largest Japanese publicly traded companies.

Two independent variables are selected as inputs to the model, weekly changes in the S&P500 index, and weekly changes in the US dollar - Japanese Yen exchange rate.

A total sample size of 676 weekly observations were collected. The researchers use 640 observations to train their support vector machine; and perform an out of sample evaluation on the remaining 36 observations.

As a benchmark, the researchers compare the performance of their model to four other models, a random walk, linear discriminant analysis, quadratic discriminant analysis and a neural network.

The random walk correctly predicts the direction of the stock market 50% of the time, linear discriminant analysis 55%, quadratic discriminant analysis and the neural network 69%, and the support vector machine 73%.

The researchers conclude:

> "*As demonstrated in our empirical analysis, SVM is superior to the other individual classification methods in forecasting weekly movement direction of NIKKEI 225 Index. This is a clear message for financial forecasters and traders, which can lead to a capital gain.*"

Identification of Early Onset Breast Cancer

Breast cancer is often classified according to the number of estrogen receptors present on the tumor. Tumors with a large numbers of receptors are termed estrogen receptor positive (ER+), and estrogen receptor negative (ER-) for few or no receptors. ER status is important because ER+ cancers grow under the influence of estrogen, and may respond well to hormone suppression treatments. This is not the case for ER- cancers as they do not respond to hormone suppression treatments.

Scholars Upstill-Goddard et al investigate whether patients who develop ER+ and ER- tumors show distinct constitutional genetic profiles using genetic single nucleotide polymorphisms data. At the core of their analysis were support vector machines with linear, normalized quadratic polynomial, quadratic polynomial, cubic and radial basis kernels. All five kernel models had an accuracy rate in excess of 93%, see Table 17.

Kernal Type	%Correctly Classified
Linear	93.28 ±3.07
Normalized quadratic polynomial	93.69 ±2.69
Quadratic polynomial	93.89±3.06
Cubic polynomial	94.64±2.94
Radial basis function	95.95±2.61

Table 17: Upstill-Goddard et al's kernels and classification results.

NOTE... ✍

Upstill-Goddard et al test multiple kernels. This is always a good strategy because it is not obvious which kernel is optimal at the onset of a research project.

Signature Authentication

Applied researchers Radhikaet al consider the problem of automatic signature authentication using a variety of algorithms including a support vector machine (SVM). The other algorithms considered included a Bayes classifier (BC), fast Fourier transform (FT), linear discriminant analysis (LD) and principal component analysis (PCA).

Their experiment used a signature database containing 75 subjects with 15 genuine samples and 15 forged samples for each subject. Features were extracted and used as inputs to train and test the various algorithms.

The researchers report a false rejection rate of 8% for SVM,13% for FT, 10% for BC, 11% for PCA and 12% for LD. They observe:

"*Results showed that the SVM classifier yielded the*

> *most promising 8% False Rejection Rate (FRR) and 10% False Acceptance Rate (FAR).*"

Example - Classifying Diabetes

The prevalence of diabetes is increasing worldwide. Among adults over 18 years of age it has risen from 4.7% in 1980 to 8.5% in 2014. Rural areas of developing countries have a low prevalence; however, populations transitioning from traditional to modern lifestyles are at elevated risk.

Diabetes prevalence also varies dramatically by geographic region and ethnic background. For example, type 2 diabetes reaches epidemic proportions in Nauru, in the Aborigines of Australia, and many in American-Indian groups in the United States. Healthcare researcher Eric Ravussin observed:

> "*The Pima Indians of Arizona have the highest reported prevalences of obesity and non-insulin-dependent diabetes mellitus. In parallel with abrupt changes in lifestyle, these prevalences in Arizona Pimas have increased to epidemic proportions during the past decades.*"

In this section, we will build a Support Vector Machine to classify whether a sample of Pima Indian women will test positive for diabetes.

Step 1 – Collecting and Exploring the Data

The sample we use is contained in the `PimaIndiansDiabetes2` data frame contained in the `mlbench` package. The data set was collected by the National Institute of Diabetes and Digestive and Kidney Diseases. It contains 768 observations on 9 variables measured on females at least 21 years old of Pima Indian heritage. Table 18 contains a description of each variable collected.

Name	Description
`pregnant`	Number of times pregnant
`glucose`	Plasma glucose concentration
`pressure`	Diastolic blood pressure (mm Hg)
`triceps`	Triceps skin fold thickness (mm)
`insulin`	2-Hour serum insulin (mu U/ml)
`mass`	Body mass index
`pedigree`	Diabetes pedigree function
`age`	Age (years)
`diabetes`	test for diabetes - Class variable (neg / pos)

Table 18: Response and independent variables in `PimaIndiansDiabetes2` data frame.

Load the data

The data can be loaded into your R session as follows:

```
data("PimaIndiansDiabetes2",
package="mlbench")
```

Let's do a quick check to ensure we have the expected number of columns and rows:

```
ncol(PimaIndiansDiabetes2)
[1] 9
nrow(PimaIndiansDiabetes2)
[1] 768
```

The numbers are in line we what we expected.

The `str` function

We can use the `str` method to check and compactly display the structure of the `PimaIndiansDiabetes2` data frame:

```
str(PimaIndiansDiabetes2)
```

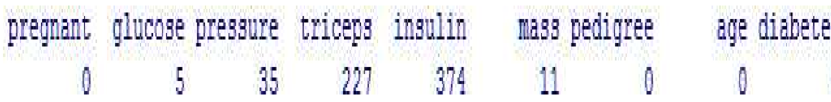
```
'data.frame':   768 obs. of  9 variables:
 $ pregnant: num  6 1 8 1 0 5 3 10 2 8 ...
 $ glucose : num  148 85 183 89 137 116 78 115 197
    125 ...
 $ pressure: num  72 66 64 66 40 74 50 NA 70 96 ...
 $ triceps : num  35 29 NA 23 35 NA 32 NA 45 NA ...
 $ insulin : num  NA NA NA 94 168 NA 88 NA 543 NA ...
 $ mass    : num  33.6 26.6 23.3 28.1 43.1 25.6 31
    35.3 30.5 NA ...
 $ pedigree: num  0.627 0.351 0.672 0.167 2.288 ...
 $ age     : num  50 31 32 21 33 30 26 29 53 54 ...
 $ diabetes: Factor w/ 2 levels "neg","pos": 2 1 2 1
    2 1 2 1 2 2 ...
```

As expected `PimaIndiansDiabetes2` is identified as a data frame with 768 observations on 9 variables. Notice that each row provides details of the name of the attribute, type of attribute and the first few observations. For example, `diabetes` is a factor with two levels `"neg"` (negative) and `"pos"` (positive). We will use it as the classification response variable.

Dealing with missing values

Did you notice the `NA` values in `pressure`, `triceps`, `insulin` and `mass`? These are missing values. We all face the problem of missing data at some point in our work. People refuse or forget to answer a question, data is lost or not recorded properly. It is a fact of data science life! However, there seem to be rather a lot, we better check to see the actual numbers:

```
sapply(PimaIndiansDiabetes2,
function(x)sum(is.na(x)))
```



```
pregnant  glucose pressure  triceps  insulin     mass pedigree      age diabete
       0        5       35      227      374       11        0        0
```

Wow! there are a large number of missing values particularly for the attributes of `insulin` and `triceps`. How should we deal with this?

The most common method and the easiest to apply is to use only those individuals for which we have complete information. An alternative is to impute with a plausible value the missing observations. For example, you might replace the `NA's` with the attribute mean or median. A more sophisticated approach would be to use a distributional model for the data (such as maximum likelihood and multiple imputation).

Given the large number of missing values in `insulin` and `triceps` we remove these two attributes from the sample and use the `na.omit` method to remove any remaining missing values. The cleaned data is stored in the R object `data`:

```
data<-(PimaIndiansDiabetes2)
data$insulin  <- NULL
data$triceps <- NULL
data<-na.omit(data)
```

We should have sufficient observations left to do meaningful analysis. It is always best to check:

```
nrow(data)
[1] 724

ncol(data)
[1] 7
```

So we are left with 724 individuals and (as expected) 7 columns.

Visual inspection

Figure 8.4 shows the distribution by age, number of times pregnant alongside box-plots of glucose, blood pressure and body-mass index.

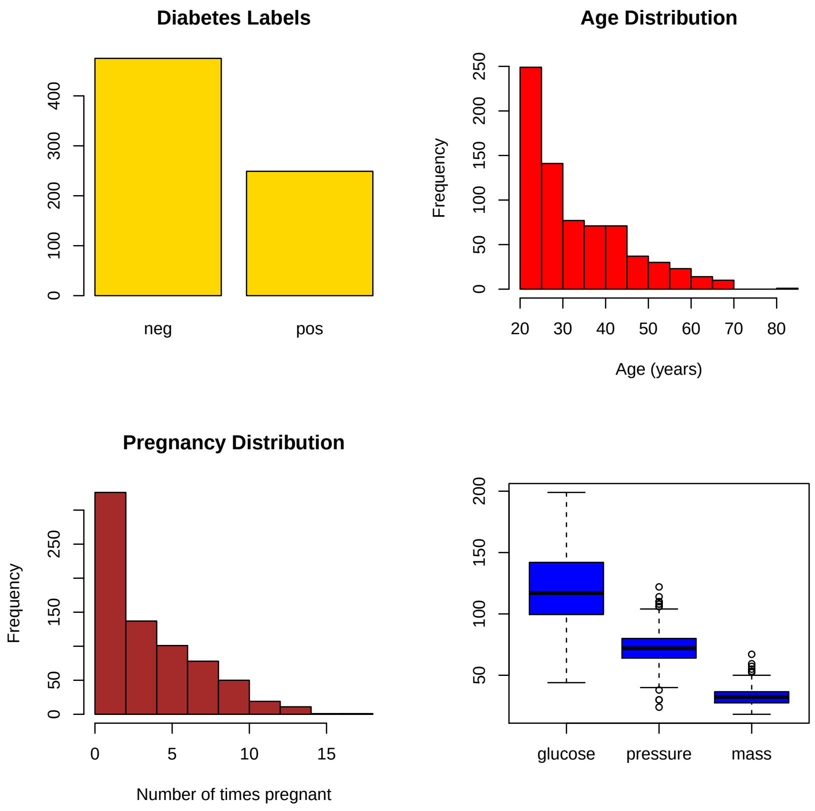


Figure 8.4: Class labels and attributes

Step 2 – Preparing the Data

The next step is to transfer the target variable into a suitable format for use in our SVM model. Here is how to do this:

```
y<-(data$diabetes)
levels(y) <- c("-1","1")
y<-as.numeric(as.character(y))
y <-as.matrix(y)
```

The first line transfers the target variable to the R object y. The labels "`neg`" and "`pos`" are replaced, in the second line, with the values "-1" and "+1". The as.numeric function is called, in the third line, to convert the "-1" and "+1" into double (numeric) values. We need y to be a matrix, and this is achieved using the as.matrix function in the final line.

Scaling the input attributes

Support vector machine kernels generally depend on the inner product of attribute vectors. Very large values might cause numerical problems. For this example, we transform the attributes so that they have a mean of zero, and a variance equal to one. This can be achieved using the scale function:

```
x<-data
x$diabetes  <- NULL
x<-as.matrix(x)
x<-scale(x)
```

The R object x, now contains the mean centered feature data.

Selecting the train and test sets

We use nrow to count the sample observations (as a check it should equal 724). We then set the training sample to select at random without replacement 600 observations. The remaining 124 observations form the test sample:

```
set.seed(103)
n=nrow(x)
train <- sample(1:n, 600, FALSE)
```

Step 3 - Train Model using Train Set

We use the svmpath package to estimate the model. It contains the svmpath function which can use two popular kernels, the polynomial and radial basis function. For our initial model we use the radial basis kernel. It is selected by setting kernel.function = radial.kernel. We also set trace=FALSE to prevent svmpath from printing results to the screen at each learning step:

```
require(svmpath)
fit<-svmpath(x[train,],
y[train,],
kernel.function = radial.kernel,
trace=TRUE)
```

The cost parameter and number of misclassified observations

A nice feature of svmpath is that it computes the estimated value of the SVM cost parameter, and the associated classification error at each step or iteration. These are stored in fit. The cost parameter at each step can be observed by appending $lambda to fit. The number of errors at each step are accessed by appending $Error to fit. Let's take a look at their values for the first three steps in the optimization process:

```
head(fit$lambda,3)
[1] 2.660405 2.528366 2.499826

head(fit$Error,3)
[1] 128 112 112
```

The cost parameter took the value 2.66 at the first step, 2.52 at the second step, and 2.49 at the third step. The number of misclassified observations was 128 at the first step, and 112 for the second and third steps.

Figure 8.5 displays the cost parameter (left panel) and the actual number of misclassified observations (right panel) by step. The cost parameter clearly declines at each step. However, the number of misclassified observations initially falls, reaching a minimum around at around 500 steps. It then begins to rise as the number of steps increases beyond 500.

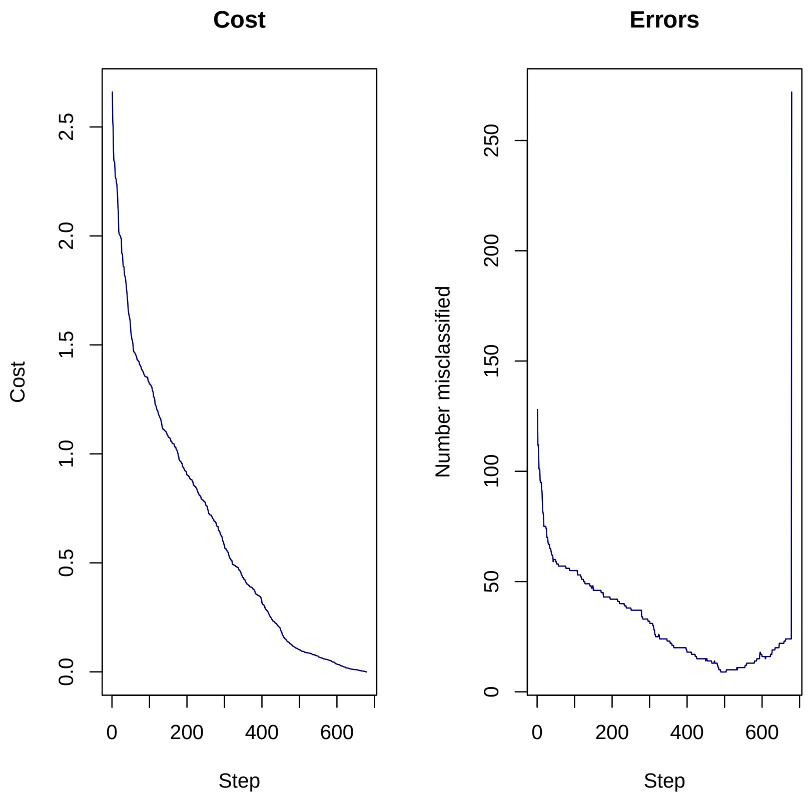


Figure 8.5: Cost (left) and number of error (right) for radial basis function kernel

What is the minimum number of classification errors? And at what step did it occur? It is difficult to tell form the above chart, but there may be multiple steps that attain the minimum value. We can use the `with` method to identify these:

```
with(fit, Error[Error == min(Error)])
 [1] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```

Yep, there are quite a few! It appears for the training sample that at best 9 out of the 600 observations are misclassified by the model. This is approximately 1.5% of the sample.

Selecting the smallest cost parameter for the minimum number of errors

Each minimum error is associated with a unique regularization/ cost value. We would like to use the smallest of these values as a parameter in our test set model. This is achieved via the following code:

```
error<-with(fit,Error[Error == min(Error)])

min_err_row<-which(fit$Error == min(
   fit$Error))

temp_lamdba<-fit$lambda[min_err_row]

loc<-which(fit$lambda[min_err_row] == min(
   fit$lambda[min_err_row]))

lambda<-temp_lamdba[loc]
lambda
[1] 0.09738556
```

Let's spend a moment to review the above code.

1. The first line stores the minimum misclassification error in the R object `error`.
2. The `which` and `min` functions are used in the second line to grab the row numbers of the minimum errors stored in `error`. These values are stored in `min_err_row`.
3. The third line captures the value of the cost parameter associated with `error` by using the row numbers stored in `min_err_row`. These values are stored in `temp_lamdba`.
4. The next line identifies which value in `temp_lamdba` has the minimum value. The location of this value is stored in the R object `loc`.

5. The final two lines use `loc` and `temp_lamdba` to pass the smallest cost parameter to the R object `lambda.` This value [0.097] is then printed to the screen.

The method **`svmpath`** reports the inverse of the kernel regularization parameter (often and somewhat confusingly called gamma in the literature). It corresponds to a gamma of $\frac{1}{0.09738556} = 10.268$.

NOTE... ✍

The regularization value is the penalty parameter of the error term (and **`svmpath`** reports the inverse of this parameter).

Making predictions

Predictions can be made using the **`predict`** function. It takes the fitted model, regularization/ cost value, and sample data. We use the value in **`lambda`** as the model cost parameter. Here is how to use it with the training sample:

```
pred_train<-predict(fit,newx=x[train,],
lambda=lambda,
type="class")
```

We already know the error on the training set should be around 1.5%, so let's check using the **`table`** function:

```
table( y[train,],
pred_train,
dnn =c("Observed",
"Predicted"))


          Predicted
Observed   -1    1
      -1  389    5
      1     4  202
```

The model appears to perform very well on the training sample. It classifies 389 out of 394 cases correctly for class -1; and 202 out of 206 cases correctly for class +1.

Step 4 - Evaluate Model Performance

Let's see how the model performs on the test sample. first, pass the test sample to the `prediction` function:

```
pred_test<-predict(fit,newx=x[-train,],
lambda=lambda,
type="class")
```

Next, use the `table` function to show classification performance:

```
table( y[-train,],
pred_test,
dnn =c("Observed",
"Predicted"))
         Predicted
Observed -1  1
      -1 65 16
      1  27 16
```

Oh dear! The model appears to perform very poorly here. For class `-1` (`neg`), 16 out of 81 examples have been misclassified; And for class `+1`(`pos`), 27 out of 43 cases have been misclassified. The overall error rate is around 35%. The relative performance has declined dramatically from that observed on the training set. What went wrong? Alas, it is a clear case of the model overfitting the data.

As we have just seen, degradation in performance due to over-fitting can be surprisingly large! The key is to remember the primary goal of the training and test samples is to provide a reliable indication of the expected error on future as yet unseen samples. In the case of our model `fit`, we need to seek an alternative.

Step 5 - Improving Model Performance

One of the quickest ways to improve performance is to choose an alternative kernel. Let's estimate another SVM, this time using a radial basis function kernel:

```
fitP<-svmpath(x[train,],
y[train,],
kernel.function = poly.kernel,
trace=FALSE)
```

The object `fitP` contains details of the fitted model. To see if the solution found was linear use:

```
fitP$linear
[1] TRUE
```

Yep, the solution found by this model is linear. This is probably a good sign in relation to overfitting. Over-fit models tend to be too complex for the given data. Compare this with the original model `fit`:

```
fit$linear
[1] FALSE
```

Of course, the fact that the solution is non-linear is not necessarily indicative of overfitting. But for `fit1`, it is a tantalizing clue.

Regularization parameter

We can gain clarity on the issues by assessing the performance of `fitP` on the train and test sets. To do this, we first obtain the regularization parameter and store it in `lambdaP`:

```
error<-with(fitP, Error[Error == min(Error)
   ])

min_err_row<-which(fitP$Error == min(
   fitP$Error))
```

```
temp_lamdba<-fitP$lambda[min_err_row]

loc<-which(fitP$lambda[min_err_row] == min(
   fitP$lambda[min_err_row]))

lambdaP<-temp_lamdba[loc]

lambdaP
[1] 73.83352

error[1]/600
[1] 0.2333333
```

Two things are noteworthy about this result.

- First, the regularization parameter is estimated as: $\frac{1}{73.83352} = 0.0135$.
- Second, the error is estimated to be 23.3% on the train set. This is much higher than we observed on `fit1`. However, it also indicates that there is a much lower likelihood that this model has over-fit the data.

Training set classifications

Now, take a look at the actual predictions using the training sample. To do this, as we have seen already, we pass the model `fitP` and the training sample to the `predict` function. The result is stored in `predP_train`. The `table` function is then called to create and display the confusion matrix:

```
predP_train<-predict(fitP,newx=x[train,],
lambda=lambdaP,
type="class")

table( predP_train,y[train,] ,
dnn =c("Observed" ,
"Predicted"))
```

```
         Predicted
Observed   -1    1
      -1  357  103
      1    37  103
```

The model correctly predicts 357 out of 460 examples for class `-1`; and 103 out of 140 examples for class `+1`. Overall, this gives us the misclassification error rate we saw earlier of 23.3%.

Test set classifications

So how does the model perform on the test sample? Pass the training sample and `fitP` to the predict function. Then use the `table` function as follows:

```
predP<-predict(fitP,newx=x[-train,],
lambda=lambdaP,
type="class")

table( predP,y[-train,] ,
dnn =c("Observed" ,
"Predicted"))
```

```
         Predicted
Observed -1   1
      -1 76  13
      1   5  30
```

The model predicts 76 out of 89 cases correctly for class `-1` (`neg`), and 30 out of 35 cases correctly for class +1 (`pos`). The model, `fitP`, has an overall error rate of around 14.5%.

Selecting an alternative kernel, may seem like a small change, but it can have a dramatic impact on performance. Earlier we saw the researchers Upstill-Goddard et al test multiple kernels.; you will want to do the same. This is because it is not obvious which kernel is optimal at the onset of a research project.

Limitations of Support Vector Machines

Unlike regression type models, SVM model parameters can be difficult to interpret. In this sense, they can seem a little like a "black box". They have a high algorithmic complexity and are difficult to explain. This can hinder their adoption, especially in areas where the clarity on the drivers of classification or prediction performance are important.

SVM performance is very sensitive to the choice of the cost parameter. This is because the decision boundary depends on the value assigned to the cost parameter. Unfortunately, the choice of kernel and parameter values is data dependent. SVMs are also memory intensive, and can take a relatively long time to train.

Summary

SVMs are a non-linear, non-parametric classification technique, which have delivered good results in numerous fields. They offer a robust classification tool developed from a theoretical basis. In this chapter, you have gained an understanding of how SVMs work through a number of practical application; and have discovered how to estimate a classification SVM using R.

In the next chapter, we encounter a very popular, and extremely successful classification algorithm - Random Forests.

Suggested Reading

- **Stock Market:** Huang, Wei, Yoshiteru Nakamori, and Shou-Yang Wang. "Forecasting stock market movement direction with support vector machine." Computers & Operations Research 32.10 (2005): 2513-2522.

- **Breast Cancer**: Upstill-Goddard, Rosanna, et al. "Support vector machine classifier for estrogen receptor positive and negative early-onset breast cancer." (2013): e68606.

- **Signature Authentication**: Radhika, K. R., M. K. Venkatesha, and G. N. Sekhar. "Off-line signature authentication based on moment invariants using support vector machine." Journal of Computer Science 6.3 (2010): 305.

Other

- Additional details on the Pima Indians can be found in Ravussin, Eric, et al. "Effects of a traditional lifestyle on obesity in Pima Indians." Diabetes Care 17.9 (1994): 1067-1074.

- For further discussion on the issues surrounding over fitting in SVMs see G. C. Cawley and N. L. C. Talbot, Preventing over-fitting in model selection via Bayesian regularization of the hyper-parameters, Journal of Machine Learning Research, volume 8, pages 841-861, April 2007.

For alternative approaches to dealing with missing values see:

- Roth, Philip L. "Missing data: A conceptual review for applied psychologists." Personnel psychology 47.3 (1994): 537-560.

- Afifi, A. A., and R. M. Elashoff. "Missing observations in multivariate statistics I. Review of the literature." Journal of the American Statistical Association 61.315 (1966): 595-604.

- Pigott, Therese D. "A review of methods for missing data." Educational research and evaluation 7.4 (2001): 353-383.

- Little, Roderick J., et al. "The prevention and treatment of missing data in clinical trials." New England Journal of Medicine 367.14 (2012): 1355-1360.

Chapter 9

Random Forests

RANDOM Forests are a non-parametric ensemble method, in which a "*forest*" of decision trees are generated by re-sampling the training data. They often deliver significant performance improvements over a single decision tree. Indeed, "*committee methods*" such as random forests, often outperform the single prediction and classifier models discussed earlier in this text.

In this chapter, you will:

- Clarify how random forests work.
- Review ensemble methods.
- Walk through several practical applications using random forests.
- Build random forests to classify Thyroid Function.
- Understand their advantages and limitations.

Before we delve into an intuitive overview of how they work, let's first refresh our memory on ensemble methods.

Understanding Random Forests

Ensemble methods combine the classification and prediction results of many different models. The individual models are known as "weak learners" because individually they have poor predictive performance. A weak learner has a prediction accuracy just above random chance on a classification problem.

The weak learners can be a similar type of model or very different. However, when combined they can form a "strong learner" with high predictive performance. In other words, the performance of an ensemble model is usually better than the performance of the individual models.

Random Forests are an ensemble classifier which fits a large number of decision trees to a data set, and then combines the predictions from all the trees. They can be used for classification and regression. For classification, the decision trees are combined using majority voting with one vote per tree over all the trees in the forest. For regression, forests are created by averaging over trees.

The Random Forests Algorithm

The algorithm begins with the random selection of examples with replacement from the sample data. This is called a "*bootstrapped*" sample. A decision tree is constructed from this subsample. In a typical random sample, approximately 63% of the original observations occur at least once.

Observations in the original data set that are not selected are called out of-bag observations. They are used to estimate the error rate (often called the out of bag error rate), and estimate feature importance.

The process of random selection of examples is repeated many times, with each sub-sample generating a single decision tree. As illustrated in Figure 9.1, each tree is different. This is because at each node, the best split is determined from randomly selected features. This results in a forest of decision trees.

Each tree is grown to the largest extent possible without pruning. Individual trees make a classification decision or prediction. The final predicted class of an observation is made by majority vote for classification, or via a weighted average for regression.

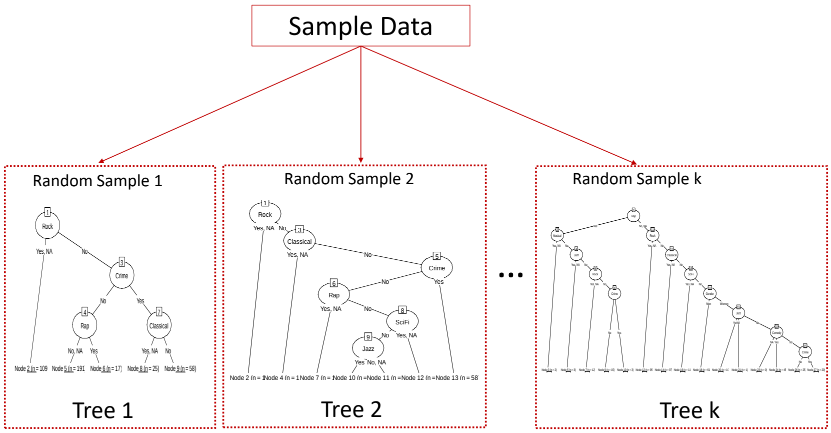


Figure 9.1: Random Forests

Although each individual tree in a forest tends to be less accurate than a classical decision tree (they are weak learners), combining multiple predictions into one aggregate prediction often results in a more accurate forecast (strong learner). Part of the reason is that prediction of a single decision tree tends to be highly sensitive to slight changes in the training set. This is not the case for the average of many trees provided they are uncorrelated. For this reason, Random Forests often decrease overall variance without increasing bias relative to a single decision tree.

NOTE... ✍

Each tree in a random forest is a weak learner. The random forest is a strong learner.

Advantages of Random Forests

Random Forests retain many of the advantages of decision trees. They are extremely easy to build and run, and often give very good results. Various studies have shown that Random Forests reduce classification and regression error on a wide array of data structures and problems. For many prediction tasks they are competitive with Neural Networks and Support Vector Machines. However, they are much faster to train because they have fewer parameters.

They require minimal data preparation. Missing values are handled, as are continuous and categorical target variables. They run efficiently on large databases, even with thousands of features, and are robust to outliers.

Unlike an individual decision tree, Random Forests do not require pruning to generalize well. Overfitting is less of a problem because they generate an unbiased estimate of the generalization error from the training sample as the forest is constructed. This error rate (called out of the bag error rate) tends to be very accurate provided a reasonable number of trees is used to construct the forest.

Practical Application of Random Forests

The range of applications for which Random Forests are appropriate spans the entire range of business, social sciences, and the natural sciences. In this section, we discuss three diverse applications. The first, involves their use to predict the condition of city sewers. Then we discuss their use in health care research, specifically classifying the presence or absence of Glaucoma. Finally, we highlight a study that used R, and compared the performance of Random Forests with logistic regression.

Waste-water Deterioration Modeling

Researcher Vitorino et al build a Random Forest model to predict the condition of individual sewers, and to determine the expected length of sewers in poor condition. The data consisted of two sources, a sewer data table and a inspection data table.

The sewer table contained information on the sewer identification code, zone, construction material, diameter, installation date and a selection of user defined covariates. The inspection data table contained information on the sewer identification code, date of last inspection and condition of sewer at date of last inspection.

The Random Forest was trained using all available data and limited to 50 trees. It was then used to predict the condition of individual sewer pipes. The researchers predict the distribution of sewer pipe in poor condition by type of material used for construction. The top three highest ranked materials were CIP (31.83%) followed by unknown material (27.94%) and RPM (23.89%).

Glaucoma Prediction

Glaucoma is the second most common cause of blindness. As glaucomatous visual field (VF) damage is irreversible, the early diagnosis of glaucoma is essential. Sugimoto et al develop a random forest classifier to predict the presence of (VF) deterioration in glaucoma suspects using optical coherence tomography (OCT) data.

The study investigated 293 eyes of 179 live patients referred to the University of Tokyo Hospital for glaucoma. The Random Forest algorithm with 10,000 trees was used to classify the presence or absence of glaucomatous VF damage using 237 different OCT measurements ; Age, gender, axial length and eight other right/left eye metrics were also included as training attributes.

The researchers report a receiver operating characteristic curve value of 0.9 for the random forest. This compared well to the value of 0.75 for an individual decision tree.

Identifying Obesity Risk Factors

Health researchers Kanerva et al investigate 4720 Finnish subjects who completed health questionnaires about leisure time physical activity, smoking status, and educational attainment. Weight and height were measured by experienced nurses. The researchers use the `randomForest` package in R to predict obesity. The results were compared with a logistic regression model.

The researchers observe that the random forest and logistic regression had very similar classification power, for example the estimated error rates for the models were 42% for men (Random Forests) versus logistic regression 43% for men.

Example - Classifying Thyroid Function

Your thyroid gland sits just below the Adam's apple. For several thousand years, it was unclear as to it's precise function. As Dr. Edward Albert Schafer, speaking before the Sixty-third Annual Meeting of the British Medical Association held during August 1895 explained:

"*That the thyroid gland is a secreting-gland no one* *who studies its structure and its model of development can well doubt...it has all the features of structure of secreating-glands...It may be fairly supposed, then that the pituitary body also furnishes to the blood an internal secretion, and it would appear that this internal secretion tends to increase the contraction of the heart and arteries and to influence the nutrition of some of the tissues.*"

Today, it is understood that the thyroid gland secretes hormones which influence metabolic rate and protein synthesis. The status of the thyroid can be assessed by laboratory tests. The tests determine whether a patient's thyroid function is euthyroidism (normal), hypothyroidism (under active thyroid) or hyperthyroidism (overactive). In this section, we build Random Forests to classify thyroid function from laboratory tests.

Step 1 – Collecting and Exploring the Data

The `thyroid` data-frame in the `mclust` package was constructed from five laboratory tests administered to a sample of 215 patients. The tests were used to predict whether a patient's thyroid function could be correctly classified. Table 19 provides details of the features, and class variable.

Variable	Description	Type
`Diagnosis`	Hypo, Normal, and Hyper	Class
`RT3U`	T3-resin uptake test (percentage)	Feature
`T4`	Total Serum thyroxin	Feature
`T3`	Total serum triiodothyronine	Feature
`TSH`	Basal thyroid-stimulating hormone	Feature
`DTSH`	Maximal absolute difference of TSH	Feature

Table 19: Features and Class in `thyroid`

Let's load the data and take a look at the first few observations:

```
data("thyroid",package="mclust")

head(thyroid)
  Diagnosis RT3U   T4  T3 TSH DTSH
1    Normal  107 10.1 2.2 0.9  2.7
2    Normal  113  9.9 3.1 2.0  5.9
3    Normal  127 12.9 2.4 1.4  0.6
4    Normal  109  5.3 1.6 1.4  1.5
5    Normal  105  7.3 1.5 1.5 -0.1
6    Normal  105  6.1 2.1 1.4  7.0
```

Figure 9.2 displays some of the characteristics of the features.

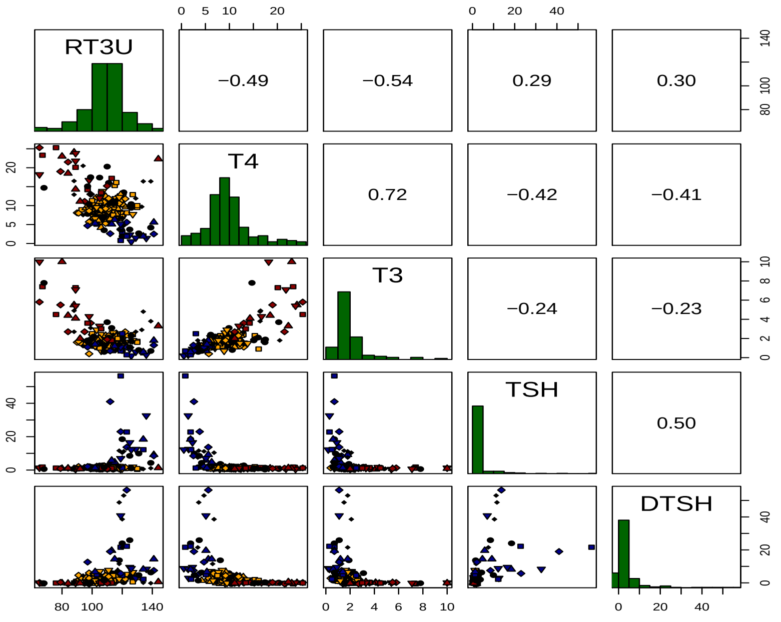


Figure 9.2: Characteristics of features in `thyroid`

The features are generally modestly correlated. However, T3 and T4 exhibit a pairwise correlation of 0.72; And the pair TSH DTSH have a correlation of 0.5. With the exception of RT3U, the features appear to be right skewed. The scatter-plots indicate a complex set of inter- feature relationships.

Step 2 – Preparing the Data

We will select 150 examples without replacement for the training set. The remainder set aside for the test set:

```
set.seed(2018)
N=nrow(thyroid)
train <- sample(1:N, 150, FALSE)
```

Step 3 - Train Model using Train Set

The randomForest package can be used to build the model. In building the random forest model we have two primary options to choose. The first is the number of trees; the second is the number of features to randomly select. We build a model with a forest of 800 trees:

```
library (randomForest)
num.trees=800
fit<- randomForest(Diagnosis  ~.,
data = thyroid[train,],
ntree=num.trees,
mtry=4)
```

For each tree in the forest, a randomly selected subset of features is used to split each node. This is controlled by the mytry parameter. In the above code we set mytry=4 to randomly select four of the six features.

Step 4 - Evaluate Model Performance

Details on the model can be viewed using the print statement:

```
print(fit)

Call:
randomForest(formula = Diagnosis ~ .,
data = thyroid[train, ],
ntree = num.trees, mtry = 4)

Type of random forest: classification
Number of trees: 800
No. of variables tried at each split: 4

OOB estimate of  error rate: 4.67%

Confusion matrix:
       Hypo Normal Hyper class.error
Hypo     22      1     0  0.04347826
Normal    1    106     1  0.01851852
Hyper     0      4    15  0.21052632
```

The first part of the output reminds us of the formula, number of trees (800), and the number of randomly selected features (variables) used at each split. This is followed by the out of the bag error rate, and the confusion matrix.

The model has an overall error of around 4.7%. Their is considerable variability in accuracy by class. For `Hypo` the error rate is 4.3%, for `Normal` 1%, and for `Hyper` 21%.

Test set performance

To assess how the model performed on the test set, we re-run it using the test set data:

```
fit_test<- randomForest(Diagnosis ~.,
data = thyroid[-train,],
ntree=num.trees,
mtry=4)

print(fit_test)
```

```
Call:
randomForest(formula = Diagnosis ~ .,
data = thyroid[-train,      ],
ntree = num.trees, mtry = 4)

Type of random forest: classification
Number of trees: 800
No. of variables tried at each split: 4

        OOB estimate of  error rate: 9.23%
Confusion matrix:
       Hypo Normal Hyper class.error
Hypo      4      3     0  0.42857143
Normal    1     40     1  0.04761905
Hyper     0      1    15  0.06250000
```

Of the 65 observations in the test set, 6 were classified incorrectly leading to an out of the bag error rate of 9.23%.

Step 5 - Improving Model Performance

Although over 90% of the observations were correctly classified, we would still like to improve performance further. In many circumstances, you might try adding new features or examples. This option is not available here.

One thing to notice is that smaller subsets of randomly selected features produces less correlation between the trees. This is important because the greater the inter-tree correlation, the greater the random forest error rate.

Unfortunately, as the number of features is reduced, the predictive power of the model may also decrease. Choosing the number of features is therefore a delicate trade-off which requires some experimental.

So how do you choose `mtry`? One rule of thumb suggests halving your initial guess, and also doubling it. Since we only have six feature, doubling is out of the question. Let's go down the half as many route and set `mtry = 2`, in order to select two random features at each split:

```
fit2<- randomForest(Diagnosis ~.,
data = thyroid[train,],
ntree=num.trees,
importance=TRUE,mtry=2)
print(fit2)

Call:
randomForest(formula = Diagnosis ~ .,
data = thyroid[train, ],
ntree = num.trees,
importance = TRUE,
mtry = 2)

Type of random forest: classification
Number of trees: 800
No. of variables tried at each split: 2

OOB estimate of  error rate: 3.33%

Confusion matrix:
       Hypo Normal Hyper class.error
Hypo     22      1     0  0.04347826
Normal    0    108     0  0.00000000
Hyper     0      4    15  0.21052632
```

The error rate at 3.3% is lower than for `fit`. The model correctly classifies all of the `Normal` cases. However, the error rate on `Hypo` and `Hyper` are similar to the values observed on the train set for `fit`.

Variable importance

We can describe our trained model `fit2` in terms of the features by ranking them according to their splitting efficiency. This can be achieved via the `varImpPlot` function. It returns two plots of variable importance, one using the average decrease in accuracy of including a feature, and the second using the average decrease in the Gini coefficient:

```
varImpPlot(fit2)
```

Figure 9.3 shows the resultant plots. It indicates T4, followed by T3 are the most important features. Notice, in this example both the accuracy metric and Gini metric agree on the order of feature importance.

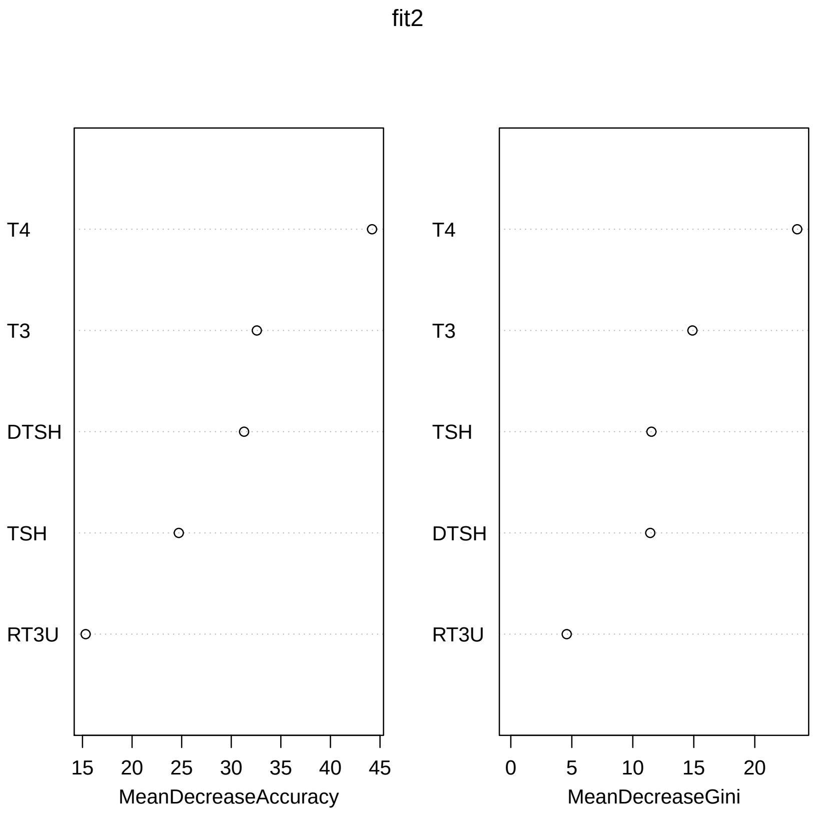


Figure 9.3: Variable importance plot for `thyroid` model `fit2`

Test set performance

The lower training set error rate is encouraging. Let's take a look at how `fit2` performed with the test set data:

```
fit2_test<- randomForest(Diagnosis ~.,
data = thyroid[-train,],
ntree=num.trees,
mtry=2)
print(fit2_test)

Call:
randomForest(formula = Diagnosis ~ .,
data = thyroid[-train,       ],
ntree = num.trees, mtry = 2)

Type of random forest: classification
Number of trees: 800
No. of variables tried at each split: 2

OOB estimate of  error rate: 4.62%

Confusion matrix:
       Hypo Normal Hyper class.error
Hypo      5      2     0   0.2857143
Normal    0     42     0   0.0000000
Hyper     0      1    15   0.0625000
```

Overall, the use of two randomly selected features has reduced the test set error rate to 4.62%.

Limitations of Random Forests

Although Random Forests can be used for regression problems, they cannot extrapolate outside of the range of the target/ feature variables. Neither are they competitive when the relationship between the target variable and features variables is linear.

Unlike decision trees, the classification rules generated by a random forest are generally incomprehensible. Thus, it can feel like a “black box” technique. For this reason, their use is more challenging in application areas where clarity on rule generation, variable importance, and how variables interact is important. They are also biased towards features with a larger number of classes.

Summary

Random Forests are a highly successful machine learning algorithm. They have gained considerable attention due to their strong performance; and are deployed in a diverse range of areas.

In the next chapter, we discuss a more general ensemble technique that combines many weak classifiers to create a strong classifier. This algorithm is known as boosting.

Suggested Reading

- **Glaucoma Prediction:** Sugimoto, Koichiro, et al. "Cross-sectional study: Does combining optical coherence tomography measurements using the ‘Random Forest’decision tree classifier improve the prediction of the presence of perimetric deterioration in glaucoma suspects?." BMJ open 3.10 (2013): e003114.

- **Identifying Obesity Risk Factors:** Kanerva, N., et al. "Random forest analysis in identifying the importance of obesity risk factors." European Journal of Public Health 23.suppl 1 (2013): ckt124-042.

- **Waste-water Deterioration Modeling:** Vitorino, D., et al. "A Random Forest Algorithm Applied to Condition-based Waste-water Deterioration Modeling

and Forecasting." Procedia Engineering 89 (2014): 401-410.

Other

- **Dr. Schafer's Thyroid Gland Presentation:** Medical News, Volume 67. British Medical Association. 1895.

Chapter 10

Boosting

BOOSTING is a machine learning technique for improving the performance of a learning algorithm. It is an ensemble technique that can be used for classification or regression. The discovery of boosting several decades ago led to dramatic improvements in predictive performance.

In this chapter, you will:

- Learn how boosting works.
- Find out about the classic Adaboost algorithm.
- Examine the usage of boosting in several real-world case studies.
- Get to play with Google's Deep Boosting algorithm to classify sonar returns.

Boosting can be used with many types of learning algorithm, and often results in a dramatic improvement in performance. Let's jump right in, and see how it works.

Understanding Boosting

Boosting algorithms combine many weak classifiers to create a strong learner. A weak learner, is slightly better than random guessing in prediction accuracy. On the other hand, a strong learner is able, given enough training data, to produce highly accurate predictions.

The Basic Approach

To begin, a simple model $\hat{h}^1(x)$ is built from the training data of features x. The weak learner $\hat{h}^1(x)$ is often called the base algorithm. It is often a single decision tree or tree stump. A stump is a single node with two leaves. However, any machine learning algorithm can be used as the base algorithm.

Greedy learning

The algorithm proceeds in a greedy fashion, which means that at each step a basis function that leads to the largest reduction of misclassification error is added.

Therefore, if $\hat{h}^1(x)$ misclassified some data, we train another copy of it $\hat{h}^2(x)$ to correct the errors made by the first model. The model $\hat{h}^2(x)$ is trained on re-weighted data where the misclassified observations receive higher weights. This forces $\hat{h}^2(x)$ to focus on observations that were difficult for $\hat{h}^1(x)$ to classify.

At each iteration, the focus on correctly classifying difficult observations intensifies. The boosting algorithm continues adding new models until no further improvements can be made, or the maximum number of iterations, specified by the user, have been reached.

Sequential learning

Boosting algorithms therefore "boost" classification accuracy by focusing on misclassified observations. They sequentially apply a learning algorithm (weak learner) to re-weighted versions of the training data, and then take a weighted majority vote to generate the prediction. The process works as follows:

Step	*Sample*		*Model*
1	*original data*	$\rightarrow$	$\hat{h}^1(x)$
2	*reweighted data* 1	$\rightarrow$	$\hat{h}^2(x)$
.	.	.	.
.	.	.	.
.	.	.	.
m	*reweighted data* m	$\rightarrow$	$\hat{h}^m(x)$

Final prediction

In the final step, there are m classifiers. Their results are combined into a weighted prediction:

$$\hat{H}(x) = \sum_{i=1}^{m} \alpha^i \hat{h}^i(x),$$

where $\hat{h}^i(x)$ is the output of weak classifier i. We see that the final classifier $\hat{H}(x)$ is the weighted sum of m weak classifiers. The weights are determined by the coefficient α^i. It is designed to give a larger weight to classifiers that have higher classification accuracy.

NOTE...

Boosting grows decision trees in sequence, with later trees dependent on the results of previous trees. This is different from a random forest which grows trees in parallel.

The Adaboost Algorithm

There are a large number of boosting algorithms. The Adaboost algorithm is very popular, and was the first adaptive boosting algorithm. It is used for binary classification where the target takes the values $y \in \{-1, +1\}$. Given the training sample $(x_1, y_1), ...(x_n, y_n)$, where x is the feature vector. It initializes the sample weights to $w^1 = 1/n$.

The first classifier $\hat{h}^1(x)$ is fitted using the weights on the training data, and the model classification error ε^1 is computed.

The sample weights are adjusted using:

$$w^2 = \frac{w^1 \exp\left(-\alpha^1 y_k \hat{h}^1(x_k)\right)}{Z^1}$$

where Z is a normalization factor:

$$Z^1 = \sum_{k=1}^{n} w^1 \exp\left(-\alpha^1 y_k \hat{h}^1(x_k)\right)$$

The second classifier $\hat{h}^2(x)$ is fitted to the sample data using the weights w^2. The process is repeated until no further improvements can be made, or the maximum number of iterations is reached.

Weight coefficient

The weight coefficient α^i is calculated as:

$$\alpha^i = 0.5\, log\left(\frac{1 - \varepsilon^i}{\varepsilon^i}\right),$$

The numerator is the classification accuracy of $\hat{h}^i(x)$. Therefore, Adaboost sets α_i to half the log ratio of accuracy to error.

Figure 10.1 plots the weight coefficient for various error values. It is zero for a random guess, i.e. when the error rate = 0.5. Models with error rates of less than 50% receive a negative weight. As the error rate approaches 0, the weight increases exponentially fast. In other words, the smaller the error the more weight is given to $\hat{h}^i(x)$. The opposite holds for a large error which receive an exponentially decreasing negative rate.

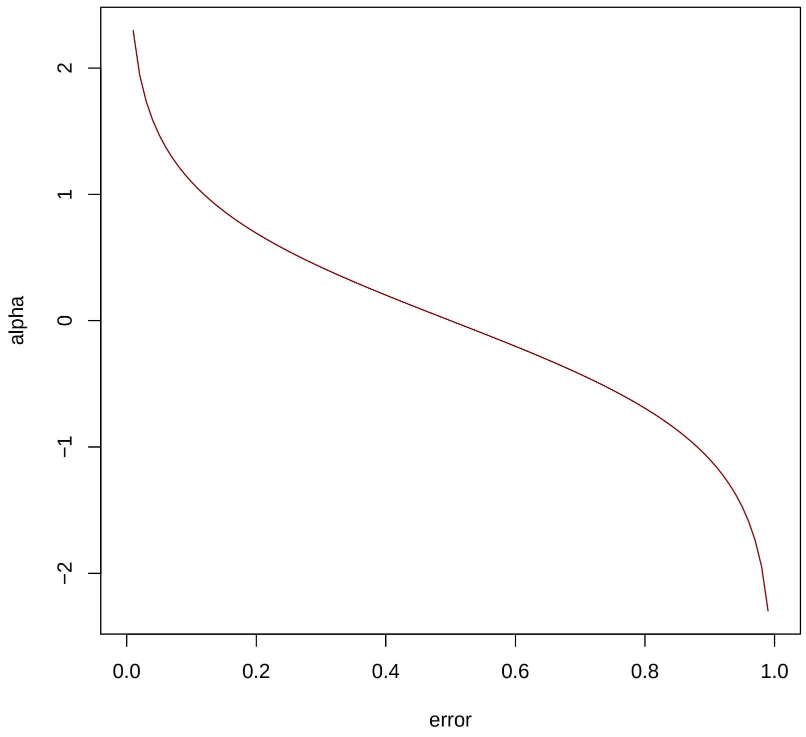


Figure 10.1: Adaboost weight coefficient

Final prediction

The final prediction is:

$$\hat{H}(x) = sign\left(\sum_{i=1}^{m} \alpha^i \hat{h}^i(x)\right)$$

This is simply a weighted linear combination of the weak classifiers, with the predicted class determined by the sign of this sum.

NOTE... ✍

Adaboost uses exponential loss, which is also an upper bound on classification error.

Advantages of Boosting

Boosting is a very simple approach that is easy to use. Unlike neural networks, it does not require sophisticated nonlinear optimization, and works well without the need for time consuming fine tuning. Relative to other machine learning methods such as linear regression, it tends to be resistant to overfitting. Furthermore, it is insensitive to uninformative features.

An interesting side benefit is that it can also be used for outlier detection. The misclassified observations after m iterations are candidates for possible outliers.

Practical Application of Boosting

Boosting algorithms are used in numerous areas outside of the original computer science applications. In this section, we highlight its use in three diverse areas - sonar communications, basketball, and logo recognition.

Reverberation Suppression

Reverberation suppression is an issue in sonar communications. As an acoustic signal is radiated through a body of water, it becomes "*weaker*" due to reflection and density of the water.

The empirical mode decomposition (EMD) algorithm is a traditional filtering technique used to manage reverberation suppression. A noise corrupted signal is applied to EMD, and intrinsic mode functions are generated. The key is to separate the signal from the noise. Noisy intrinsic mode functions are high frequency component signals, whilst signal-led intrinsic mode functions are low frequency component signals. The selection of the appropriate intrinsic mode functions, which are used for signal reconstruction, is often done manually.

Scholars Cheepurupalli et al use Ada Boost to automatically classify "noise" & "signal" intrinsic mode functions. The results were very encouraging as they found that combining Ada Boost with EMD increases the likelihood of correct detection. The researchers conclude:

> "*...that the reconstruction of the chirp signal even at low input SNR* [signal to noise] *conditions is achieved with the use of Ada Boost based EMD as a de-noising technique.*"

Basketball Player Detection

Sports fans Markoski et al investigate player detection during basketball games using a variant of adaboost called the gentle adaboost algorithm. Tree stumps were used as the base model. A total of 6000 examples that contain the players entire body, and 6000 examples of players upper body only, were used to train the algorithm.

The researchers observe for images that contain the players whole body, the algorithm was unable to reduce the level of false positives below 50%. In other words, flipping a coin would have been as accurate. However, a set of test images using players upper body obtained an accuracy of 70.5%. This is considerably better than chance. However, a 30% error rate is still relatively high. Therefore, the researchers conclude that the use of the gentle adaboost algorithm for detecting players upper body results in a relatively large number of false positive observations.

Vehicle Logo Recognition

Sam et al consider the problem of automatic vehicle logo recognition. The researchers use a boosting variant called the modest Adaboost algorithm. A total of 500 vehicle images were used in the training procedure. A total of 200 images were used in the test set with 184 images recognized successfully, see Table 20. This implies a misclassification error rate of around 9%.

Manufacturer	Correct	Mistaken
Audi	20	0
BMW	16	4
Honda	19	1
KIA	17	3
Mazda	18	2
Mitsubishi	20	0
Nissan	17	3
Suzuki	18	2
Toyota	19	1
Volkswagen	20	0
Total	184	16

Table 20: Logo recognition rate reported by Sam et al

Example - Classifying Sonar Signals

Sonar listening devices have been around since Lewis Nixon's 1906 contraption was used to detect icebergs. Determining the difference between metallic and non-metallic returns can be a challenge. In this section, we use Google's deep Boosting algorithm to help out with the task.

Step 1 – Collecting and Exploring the Data

The `Sonar` dataset in the `mlbench` package contains 208 observations on sonar returns collected from a metal cylinder, and a cylindrical shaped rock positioned on a sandy ocean floor.

Sonar returns were collected at a range of 10 meters and obtained from the cylinder at aspect angles spanning 90°, and from the rock at aspect angles spanning 180°. In total, 60 numerical features were collected.

Summarize the target variable

First, load the data and use the `table` function to summarize the sonar returns:

```
data("Sonar",package="mlbench")
table(Sonar$Class)
```

```
  M   R
111  97
```

The sonar returns are composed of 111 metallic returns labeled as "`M`"; and 97 rock returns labeled with the letter "`R`".

Figure 10.2 shows a correlation plot of the 60 features. The inter-feature correlations range from moderately negative to highly positive.

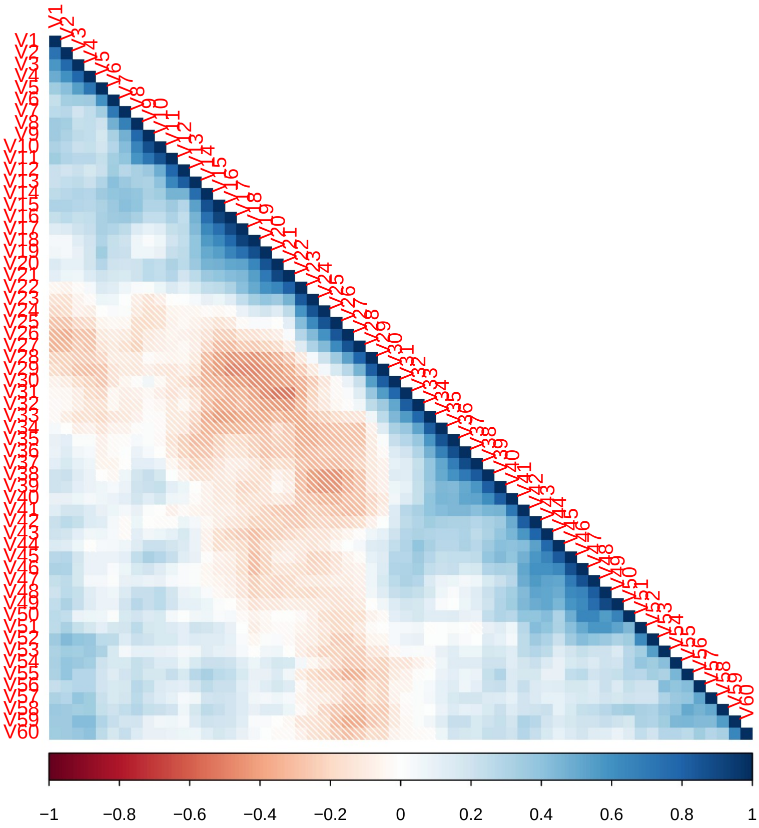


Figure 10.2: Correlation plot of `Sonar` attributes.

Step 2 – Preparing the Data

One of the advantages of boosting algorithms is that they can be used with little data preparation. For our analysis, we simply transfer the sample data to the R object `dataSample`. Then select at random with replacement 150 examples for the train set, with the remainder used as the test set:

```
dataSample<-(Sonar)
set.seed(2016)
n=nrow(dataSample)
```

```
train <- sample(1:n, 150, FALSE)
```

Step 3 - Train Model using Train Set

We use the `deepboost` package which is based on Google's Deep Boosting algorithm. We use it to fit an adaboost model via the `deepboost` function. The function takes the standard R formula as the first argument. We use all of the features available in the sample:

```
require(deepboost)
fit<-deepboost(Class~.,
dataSample[train,],
tree_depth = 5,
num_iter = 10,
lambda = 0.0,
loss_type = "e")
```

Here is a brief summary of the above code:

- The depth of the tree is controlled by `tree_depth`, and the number of iterations is set to 10.

- The parameter `lambda` is used for regularization of tree depth. It is not used in Adaboost, and so we set it to zero.

- The Adaboost algorithm uses exponential loss, specified by `loss_type = "e"`.

As the model runs, it will report the error, average tree size, and number of trees used for each iteration. You can view a summary of this information via the `print` function:

```
deepboost.print(fit)
[1] "Model error: 0"
[1] "Average tree size: 30.5555553436279"
[1] "Number of trees: 9"
$error
```

```
[1] 0

$avg_tree_size
[1] 30.55556

$num_trees
[1] 9
```

It reports a model error of 0, which informs us it was able to classify all of the train set observations correctly. The average tree size is just over 30, and 9 trees voted in the final model.

Step 4 - Evaluate Model Performance

Assessing performance is straightforward. We use the `predict` and `table` functions to view the confusion matrix for the training data:

```
pred_train<-predict(fit,
dataSample[train,1:60])

table(dataSample$Class[train],
pred_train)
   pred_train
     M  R
  M 77  0
  R  0 73
```

As previously indicated, the model correctly classifies all of the observations so that the misclassification error is zero.

Test set performance

As we saw earlier, great training set performance does not always translate into great test set performance. Let's take a look:

```
pred_test<-predict(fit,
dataSample[-train,1:60])
```

```
table(dataSample$Class[-train],
pred_test)
 pred_test
     M  R
  M 26  8
  R 11 13
```

Of the 58 examples in the test set, 19 are incorrectly classified resulting in an error rate of over 30%! What went wrong?

Step 5 - Improving Model Performance

It appears our first model overfit the data by a wide margin. One way to deal with this is to reduce the number of iterations. One rule of thumb is to try half the original number. In this case that suggests we set `num_iter = 10`. Experimentation is the key to success in machine learning, so we try a logistic loss function by setting `loss_type = "l"`. Let's see how well this models performs on the training sample:

```
fit1<-deepboost(Class~.,
dataSample[train,],
tree_depth = 5,
num_iter = 5,
beta = 0,
lambda = 0.0,
loss_type = "l")

pred1_train<-predict(fit1,
dataSample[train,1:60])

table(dataSample$Class[train],
pred1_train)
   pred1_train
     M  R
  M 77  0
  R   0 73
```

Performance is exactly the same as for the first model - all of the training set observations are correctly classified. The real question is whether it improved things in terms of the test set. Let's take a look:

```
pred1_test<-predict(fit1,
dataSample[-train,1:60])

table(dataSample$Class[-train],
pred1_test)
   pred1_test
     M  R
  M 30  4
  R  5 19
```

The model has not delivered a perfect classification result. However, the error, at just over 15%, is almost half of that observed in the first model.

NOTE... ✍

Experiment with other parameters of the `deepboost` function. Can you get the Misclassification error rate below 5%?

Limitations of Boosting

Similar to random forests, a major downside is that we lose the simple interpretability we observe with decision trees or linear regression. Boosting in general can be susceptible to noise in the data. Furthermore, when there are a large number of outliers in the data, the emphasis on the hard examples i.e. outliers, can erode overall performance.

Summary

Boosting algorithms are a vital technique in your machine learning tool box. The can deliver exceptional performance. However, just like any other technique, they are subject to overfitting, and there is no guarantee that they will work well "out of the box". Experimentation and benchmarking performance against alternatives is the best option.

In the next chapter, we discuss a non-supervised technique useful for identifying groups of observations from unlabeled data. The techniques is K-means clustering.

Suggested Reading

- **Basketball Player Detection:** Markoski, Branko, et al. "Application of Ada Boost Algorithm in Basketball Player Detection." Acta Polytechnica Hungarica 12.1 (2015).
- **Reverberation Suppression:** Cheepurupalli, Kusma Kumari, and Raja Rajeswari Konduri. "Noisy reverberation suppression using adaboost based EMD in underwater scenario." International Journal of Oceanography 2014 (2014).
- **Vehicle Logo Recognition:** Sam, Kam-Tong, and Xiao-Lin Tian. "Vehicle logo recognition using modest adaboost and radial tchebichef moments." International Conference on Machine Learning and Computing (ICMLC 2012). 2012.

Other

- **Google's Deep Boosting Algorithm:** Cortes, Corinna, Mehryar Mohri, and Umar Syed. "Deep boosting." 31st International Conference on Machine Learning, ICML 2014. International Machine Learning Society (IMLS), 2014.

- **Overfitting:** See for example:
  - Buhlmann P, Hothorn T. Boosting Algorithms: Regularization, Prediction and Model Fitting (with Discussion). Statistical Science. 2007;22:477{522.
  - Zhou ZH. Ensemble Methods: Foundations and Algorithms. CRC Machine Learning & Pattern Recognition. Chapman & Hall; 2012.
  - Schapire RE, Freund Y. Boosting: Foundations and Algorithms. MIT Press; 2012.

Chapter 11

K- Means Clustering

CLUSTER analysis is the Swiss Army knife of machine learning. Clustering is essentially an exploratory data analysis tool which groups objects into similar groups or clusters. Cluster analysis itself is not one specific algorithm but consists of a wide variety of approaches. One of the most popular is K-means clustering.

In this chapter, you will:

- Review supervised and unsupervised learning.
- Explore how K-means clustering works.
- Survey several real-world uses of the algorithm.
- Use the algorithm with R to identify clusters of countries by the well-being of their population.

Cluster analysis can be applied to a wide variety of problems, and can help you gain deeper insights from your data. In Biology, it can be used to find structure in DNA; in marketing it can be used to create customer or product segments. Now. let's get started!

Understanding K- Means Clustering

In order to understand K-means clustering we need a quick refresh of supervised and unsupervised learning. In supervised learning, you have both class labels and features. For example, in a medical setting you might have labels on the health status of patients (“Excellent”, “Good”, “Poor”), alongside features such as age, weight, and so on. Most of the machine learning techniques discussed in this text have been supervised learning algorithms.

In unsupervised learning, there are no class labels from which to compare test and training set performance. For example, you might have a group of features such as age and weight for several patients, as shown in Figure 11.1. The goal is to use these features to organize patients into similar groups. How might you attempt this?

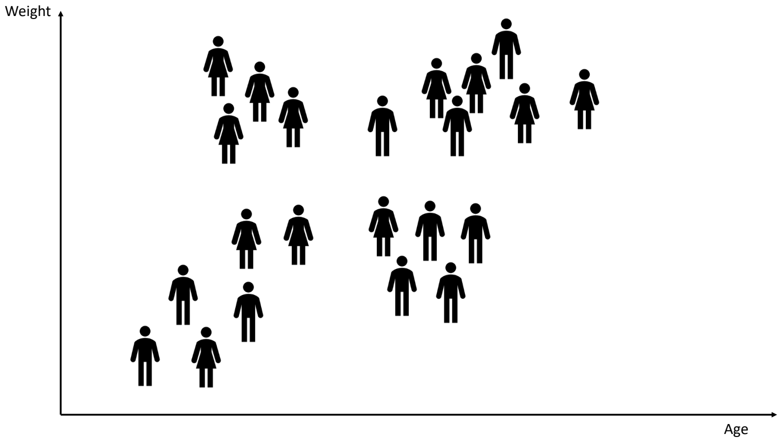


Figure 11.1: Unlabeled patient data

A simple solution

One simple solution would be to define two groups. Patients under 30 years old, and less than 250 pounds are assigned to one group; and patients 30 years or older, and over 250 pounds to another group. In effect, you will have defined two clusters, as illustrated in Figure 11.2.

Whether these two clusters are useful depends, in part, on the meaning that can be attached to them. Do they have an interesting interpretation? Can they be used in further analysis? Do they throw light on a challenging problem?

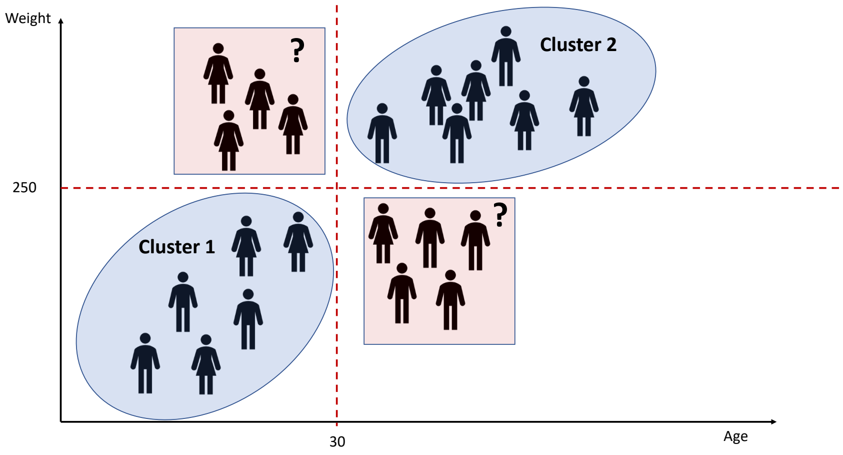


Figure 11.2: Patient clusters using a simple rule

Notice in Figure 11.2, that our simple rule leaves out patients who are less than 30 years old and heavier than 250 pounds. It also leaves out a group of patients who are older than 30 and weigh less than 250 pounds. We would like a rule that uses all of the sample data and creates clusters of patients who are similar. This is precisely the function of K-means clustering.

How K-means Clustering Works

A clustering algorithm partitions a sample into distinct, exclusive clusters so that the data points in each cluster are similar to each other. K- means clustering is an unsupervised machine learning algorithm which attempts to automatically identify useful clusters of observations.

Let's use Figure 11.1 to illustrate the approach. The key parameter is the number of clusters. You choose this value ahead of running the algorithm. Suppose we set k=4, for Figure 11.1, here is how the algorithm works:

Step 1: Randomly choose the k cluster means

Pick at random k cluster means (often called centroids). A centroid is the center of a cluster. Figure 11.3 illustrate the randomly chosen centroids. Each centroid represents a random guess of the central location of each of the 4 clusters. As with any random guess, they may be good, bad or indifferent.

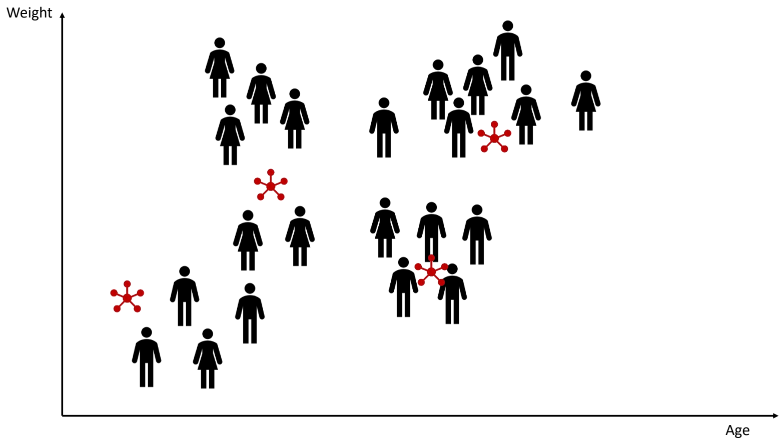


Figure 11.3: Randomly chosen centroids

Step 2: Assign observations

Each sample observation is assigned to the closest centroid. How do we determine "*closeness*"? It is determined using a distance metric. Typically, this is the squared Euclidean metric of Equation 3.1.

Figure 11.4 illustrates the assignment of our observations to each cluster. Cluster 1 has six observations, and cluster 4 has four observations. Assignment is based on the proximity of an observation to the nearest centroid.

At the end of this step every observation has been assigned to a cluster. In this way, the algorithm uses all of the observations in a data sample.

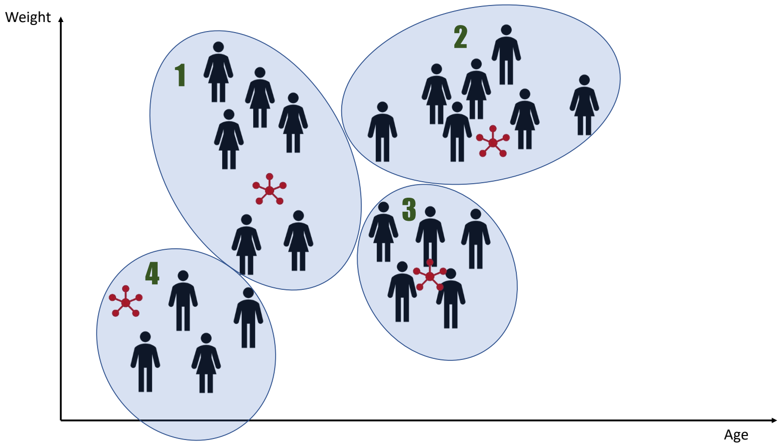


Figure 11.4: Assignment of sample data

NOTE... ✍

Other distance metrics are sometimes specified such as the Manhattan distance given in equation 3.2.

Step 3: Re-calculate centroids

The algorithm now uses the sample observations in each cluster to calculate the center (mean) of each cluster. This is done by taking the average position of all the points in the given cluster. These new centroids are illustrated in Figure 11.5. In this example, there is no change in the values of the centroids for cluster 2 and 3. However, this is not the case for cluster 1 and 4. The mean of cluster 1 shifts upwards on the illustration; whilst the mean of cluster 4 shifts rightwards.

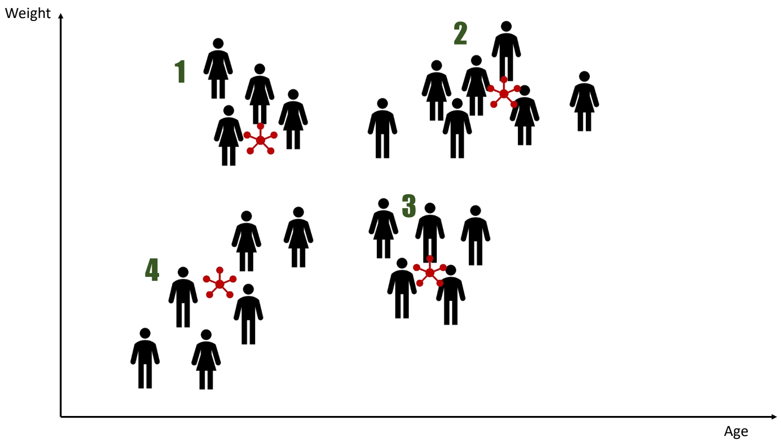


Figure 11.5: Centroids calculated from sample data

NOTE... ✍

If a cluster is left with zero observations, the centroid is randomly moved to a new location.

Step 4: Re-calculate the distance metrics

The next step is to recalculate the distance metrics for each observation and assign them to the nearest centroid. In Figure 11.6 we see that cluster 4 gains two new observations, at the expense of cluster 1. There is no change to cluster 2 or 3.

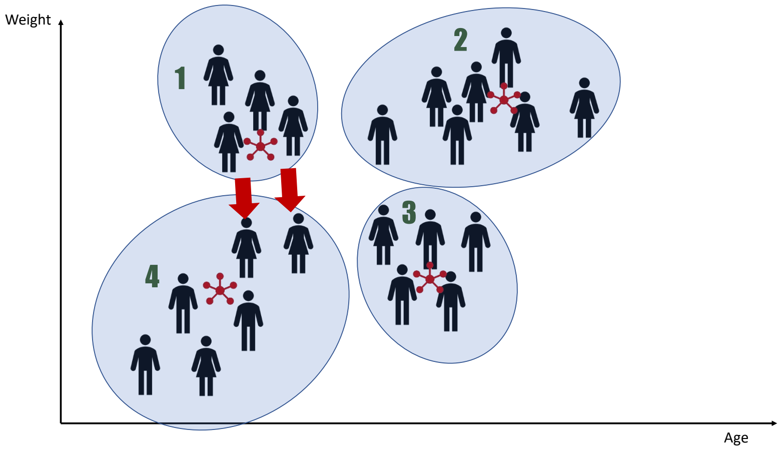


Figure 11.6: Cluster re-assignment

The algorithm repeats step 3-4 until no reassignment occurs. At this point the algorithm stops, and reports the final clusters.

NOTE... ✍

The K-means algorithm was original developed as a method for vector quantization in communication (pulse-code modulation) applications.

Advantages of K- Means Clustering

Much of the popularity of k-means lies in the fact that the algorithm is extremely fast and can therefore be easily applied to large data-sets. It is also very intuitive because the algorithm optimizes intra-cluster similarity which allows you to easily sort your data into similar groups. Furthermore, it only requires you to specify the number of clusters.

Practical Application of K- Means Clustering

K- means clustering has been used to solve several mysteries. Two of the most famous being the mystery of the Rasberry crazy ant, and the secret behavior of European shags. It has also been used to throw light on the painful disorder Fibromyalgia. We discuss each of these cases in this section.

Pinpointing the Rasberry Crazy Ant

Back in 1938, residents of Brownsville, a Gulf Coast town located at the southernmost tip of Texas, spotted thousands of tiny critters described as ants that "acted crazy". The origin and identity of the ants remained a mystery.

Several decades later, and deeper into Texas, the mystery was resolved, and K-means clustering played a role.

Tom Rasberry - catalyst for change

Things began to change after the "*crazy ants*" descended in mass on Pasadena, a city just outside of Houston, Texas. Entomologist, Tom Rasberry, who the ants in Texas are named after, discovered millions of the critters crawling all over an industrial estate. He raised the alarm, saying:

> "*In my opinion these ants pose a clear and present danger to our way of life, and the time for real action was years ago. If we don't act now the consequences could be irreversible.*"

And academic, Dr. John La Polla, Associate Professor at Towson university observed:

> "*In the past decade, Houston, Texas has been virtually overrun by an unidentified ant species, the sudden appearance and enormous population sizes*

> *and densities of which have received national media attention. The Rasberry Crazy Ant, as it has become known due to its uncertain species status, has since spread to neighboring states and is still a major concern to pest control officials.*"

The Rasberry Crazy Ant (known as the tawny crazy ant in the other Gulf Coast States), is a very small ant, around 3.2 mm long. Much smaller that the giant-sized Fire Ant Texans are use to battling. As Dr. John La Polla and his colleagues warned:

> "*Particularly worrisome was the preliminary observation that this crazy ant can successfully compete with and even displace the Red Imported Fire Ant, Solenopsis invicta Buren, one of the most costly invasive arthropods in the United States and generally considered to be one of the worst invasive insects in the world.* "

Invasive species cause an estimated $120 billion in environmental damage annually in the United States alone. The Rasberry Crazy Ant was adding to the bill. But, before it could be effectively dealt with, it had to be identified. Dr. John La Polla and several other researchers solved the mystery. They identified the ant as Nylanderia fulva (Mayr) using morphometric and molecular sequence data.

A role for K-means clustering

DNA sequence data and morphometic measurements were taken on a sample of the Rasberry Crazy Ants. The data, alongside measurements on a variety of ant species were transformed, and a K-means algorithm with four clusters applied.

The researchers merged cluster 3 and 4, as they only contained native North American ant species.

> "*K-means clustering identified four strongly differentiated clusters, clearly discriminating all North American Nylanderia species*"

Cluster 2 was more interesting. It offered a big hint as to the potential species of the Rasberry Crazy ant because it contained the species Nylanderia fulva. Although, the k-means clustering analysis was not conclusive, it helped guide the researchers toward their eventual conclusion - the Rasberry Crazy ant is the pest species Nylanderia fulva originating out of Colombia, South America.

Automatic Shags

One of the fascinating aspects of being involved in machine learning and statistical analysis, is that you come into contact with so many really interesting professionals. Take for example, Ethologists, they spend their days studying animal behavior. Jane Goodall, is perhaps, the most famous Ethologist. She has spent her life studying chimpanzees in the African bush. Imagine getting paid to do that! Ranks right next to England winning the World Cup in my books. I don't know about you, but I'd study beach life!

Anyway, Japanese Ethologist Kentaro Sakamoto took a fancy to European shags; and headed to Scotland's Isle of May for a piece of the action. The European or common shag (Phalacrocorax aristotelis) is a rock loving, fish eating, species of cormorant. The Japanese species has been used for centuries in traditional fishing.

The traditional and modern approach

The traditional approach, epitomized by Jane Goodall, involves studying animals directly in their natural habitat. However, as Kentaro quickly discovered on arrival at the 1.1 mile by 546 yards wide lump of windswept, wave splashed rock that makes up the Isle of May, the traditional approach can be a challenge:

"...this approach is difficult, often impossible, in the case of behaviors which occur in remote areas and/or at great depth or altitude."

Rather than spending several decades crouched down in a bird-watching hideout on the craggy, damp, cold and blustery Isle of May, Kentaro and other researchers devised a method to collect data remotely. They attached accelerometers (data loggers) to sixteen individual birds, as shown in Figure 11.7:

"Accelerometers are particularly useful in this respect because they can record the dynamic motion of a body in e.g. flight, walking, or swimming."

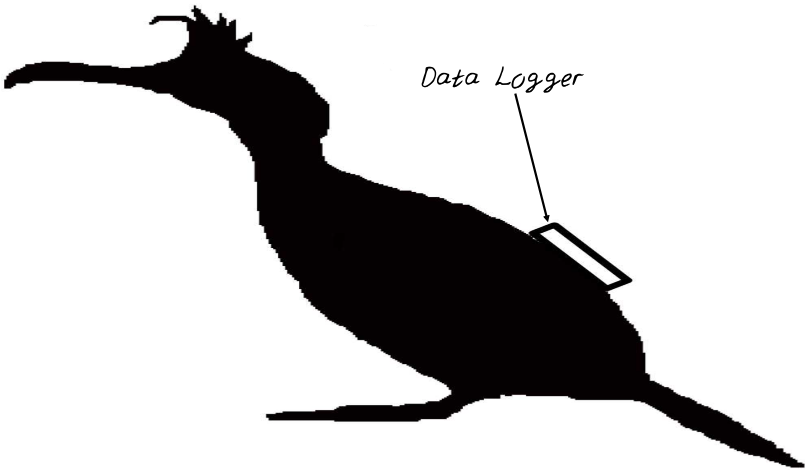


Figure 11.7: Position of the logger attachment. Adapted from Sakamoto et al. See **Automatic Shags** in suggested reading at the end of this chapter for full citation.

Scottish Shag behavior

A spectrum was generated from the acceleration signals using the continuous wavelet transformation. The k-means clustering algorithm was then used as to cluster each second of the spectrum. The researchers set k = 20, to capture 20 separate clusters; and the distance between observations and centroids was calculated using the squared Euclidean metric.

Each cluster was then interpreted as a specific bird behavior such as walking, swimming, sleeping, and so on. An ethogram is a catalog of behaviors or actions exhibited by an animal used in ethology. In effect, the k-means algorithm was used to create an automatic ethogram of bird behavior. This is a fascinating application of k-means clustering.

Kentaro excitedly concludes:

> "*To date, quantifying the behavior of wild animals that are hard to track has been extremely challenging. In the context of conservation biology, lack of information about the foraging ecology of an endangered species may hinder the development of an effective conservation strategy. Our approach has the potential to shed light on hitherto unknown aspects of the lives of such animals. It is noteworthy that our procedure employs an unsupervised clustering algorithm opening up the possibility to extract novel behavior patterns that researchers have never observed directly.* "

Identifying Fibromyalgia Subgroups

Fibromyalgia (FM) is a painful disorder which causes fatigue, sleep, memory and mood issues. It impacts men, women and children. Unfortunately, at present, there is no known cure for the affliction. Spanish researchers Docampo et al used k-means clustering to shed new light on the characteristics of the condition.

The sample consisted of 1,446 FM cases collected on 48 variables from Fibromyalgia units of five Spanish Hospitals. The majority of the variables were dichotomous. Non-dichotomous variables were therefore converted into binary features. These were then combined to create eleven separate indices of psychiatric and quality of life shown in Table 21. Individual patient scores were derived from these indices.

Item	Index
1	Fibromyalgia Impact
2	Fatigue Impact
3	Pain Level
4	Fatigue Level
5	Life Quality - Physical
6	Life Quality - Mental
7	Anxiety
8	Depression
9	Sleep Quality
10	Years of Disease
11	Number of Tender Points

Table 21: Indices used in Fibromyalgia study of Docampo et al

The K-means algorithm, with k=3, was then applied to the scores. The hope was that the resultant clusters would shed new light on characteristics of the disease.

Figure 11.8, shows the three clusters, plotted against the first two principal components. All data points, fall neatly into each of the clusters, and there is no overlapping of clusters. In other words, the clusters were able to identify specific dimensions of the underlying data.

The interpretation of clusters is somewhat subjective. However, often it is possible to assign a useful meaning. The researchers suggested that:

> "*Based on their composition, the dimensions were labeled as: FM symptoms and their characteristics (Dimension 1: "symptomatology"), familial and personal comorbidities (Dimension 2: "comorbidities") and FM core clinical scales (Dimension 3:*

"scales")...the resulting patient clusters could indicate different forms of the disease, relevant to future research, and might have an impact on clinical assessment."

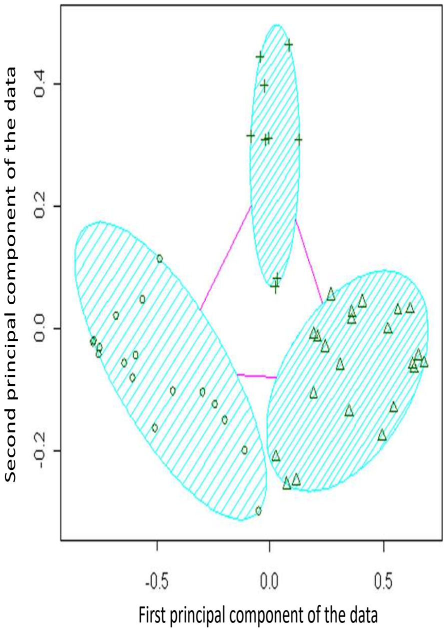


Figure 11.8: Cluster analysis results for Fibromyalgia. Adapted from Docampo et al. See **Fibromyalgia** in suggested reading at the end of this chapter for full citation.

Example - Classifying the Well-being of Citizens

Life expectancy and the child mortality rate are two important measures of the well-being of a population. For several years the United Nations Children's Fund has reported both metrics annually for every country in the world. In this section, we apply the K-means algorithm to data on Life expectancy and child mortality for countries with Gross National Income less than $1000 US dollars per annum per capita.

Step 1 – Collecting and Exploring the Data

The sample is contained in the `unicef` dataframe from the `ks` package. It contains data on the under-five mortality rate and life expectancy for each country. The under-five mortality rate measures the probability of dying between birth and exactly five years of age, expressed per 1,000 live births.

Let's load the data, and take a look at the first few observations:

```
data("unicef",package="ks")
head(unicef)
            Under-5 Ave life exp
Afghanistan     257           43
Angola          260           45
Armenia          35           73
Azerbaijan      105           72
Bangladesh       77           60
Benin           158           54
```

We see that Afghanistan has an under-five mortality rate of 257, and a life expectancy of 43 years. Whilst Armenia has a much lower under-five mortality rate, and a life expectancy of 73 years old.

The `summary` function provides a nice overview of the entire sample:

```
summary(unicef)
    Under-5        Ave life exp
 Min.   : 19.0   Min.   :39.00
 1st Qu.: 76.0   1st Qu.:47.00
 Median :122.0   Median :54.00
 Mean    :125.6   Mean    :55.71
 3rd Qu.:180.0   3rd Qu.:65.00
 Max.    :316.0   Max.    :73.00
```

The under-five mortality rate has a very large spread. It ranges from a high of 316 per 1,000 live births, to 19 per 1,000 live births. We see a similar large range in life expectancy, a minimum value of 39 years and a maximum value of 73 years.

Figure 11.9 shows the scatter plot of the data, alongside the regression line of `Ave life exp` on `Under-5`. It exhibits a pronounced negative slope, as under five mortality increases, life expectancy decreases.

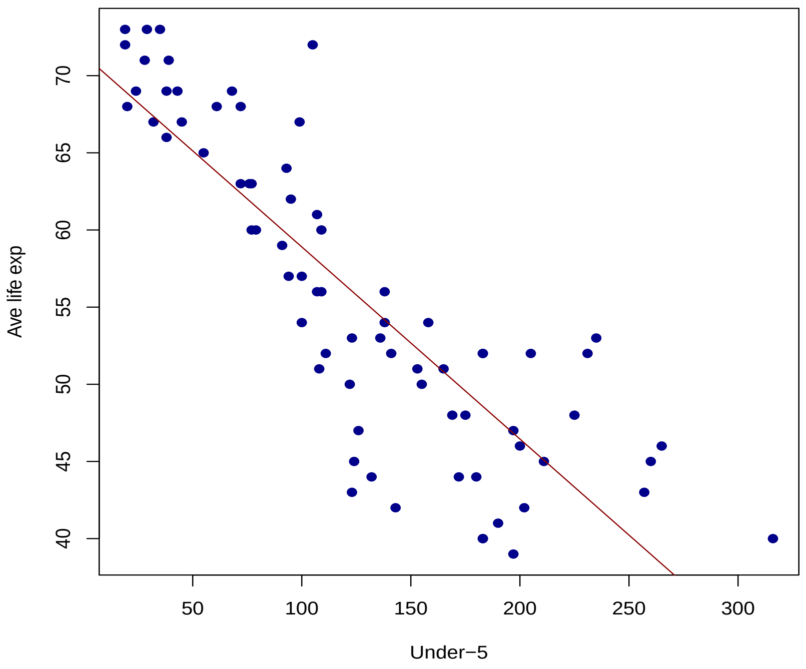


Figure 11.9: Scatter plot and regression line of features in `unicef`

Step 2 – Preparing the Data

K-means uses the squared Euclidean distance to allocate objects to clusters. This requires the data to have roughly the same scale. Figure 11.10 (left) shows a box-plot of the two features. Quite clearly, the ranges differ. The data should to be scaled prior to using the K-means algorithm. A simple way to achieve this is via the scale function:

```
x<-scale(unicef)
```

Figure 11.10 (right) shows the box-plots for the scaled features. Most of the data for each feature now lies in a similar range.

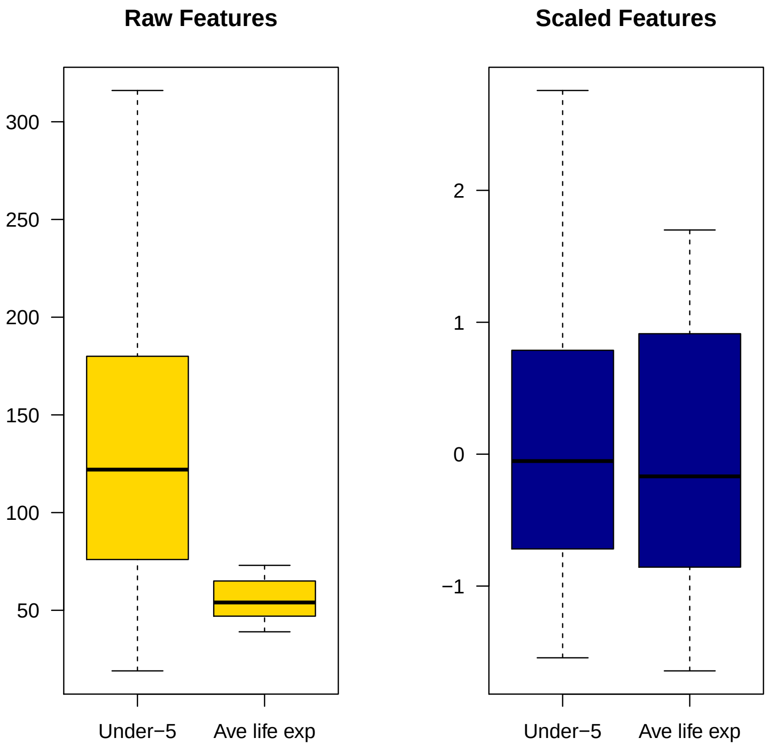


Figure 11.10: Box-plot of raw and scaled features for `unicef`

Step 3 - Train Model using Sample Data

How many clusters should you choose? Choosing the optimal value for k is best done by first inspecting the data. We saw earlier that Japanese Ethologist Kentaro Sakamoto set k = 20, to capture the behavioral activities of European shags; and Dr. John La Polla used four clusters in to help identify the Rasberry Crazy Ant. Our initial guess is six clusters, so that `krange=6`. Now let's get to the meat of the matter. Here is how to specify a K-means model in the `fpc` package:

```
require(fpc)
set.seed(2018)
fit<-kmeansruns(x,krange=6,runs=100)
```

Much of this will be familiar to you by now. However, notice that the `kmeansrun` function calls the `kmeans` function from the `stats` package. The `kmeansrun` function is preferred because it runs the K-means algorithm using multiple random starts. In this example, we use 100 random starts via the `runs` argument.

The `summary` function provides an overview of the fitted model:

```
summary(fit)
             Length  Class   Mode
cluster       73      -none- numeric
centers       12      -none- numeric
totss          1      -none- numeric
withinss       6      -none- numeric
tot.withinss   1      -none- numeric
betweenss      1      -none- numeric
size           6      -none- numeric
iter           1      -none- numeric
ifault         1      -none- numeric
crit           6      -none- numeric
bestk          1      -none- numeric
```

The output is a list of vectors, where each component has a different length. For example, the `cluster` component has a length 73, the number of countries in the sample. It contains the cluster assignments of each country. To see the first few, use the `head` function:

```
head(fit$cluster,3)
Afghanistan       Angola     Armenia
          1            1           6
```

For example, Afghanistan is assigned to cluster 1, as is Angola. Armenia is assigned to cluster 6.

Step 4 - Evaluate Model Performance

The `tidy` function from the `broom` package, offers a very quick way to summarize a K-means model:

```
library(broom)
tidy(fit)
```

```
           x1          x2 size  withinss cluster
1  2.15608153 -1.2006514    4 0.6914159       1
2 -0.51689922  0.5425438   13 1.3211591       2
3  1.05185474 -0.6107632   12 2.1466456       3
4  0.58877066 -1.3302481   11 2.2580384       4
5  0.03934516 -0.3059877   15 1.9392911       5
6 -1.19964154  1.3500688   18 2.8104510       6
```

It reports the centroids of each cluster, along with the number of observations in each cluster, the within cluster error sum of squares, and the cluster assignment value.
The first thing to notice is that the centroids appear to be well spread out across the x1 (`Under -5`), x2 (`Ave life exp)` sample space. Most of the clusters have at least 11 observations (countries). The exception is the first cluster, which only contains four countries.

Figure 11.11 plots the sample data color coded by cluster, alongside the cluster centroids. The model appears to separate the data well.

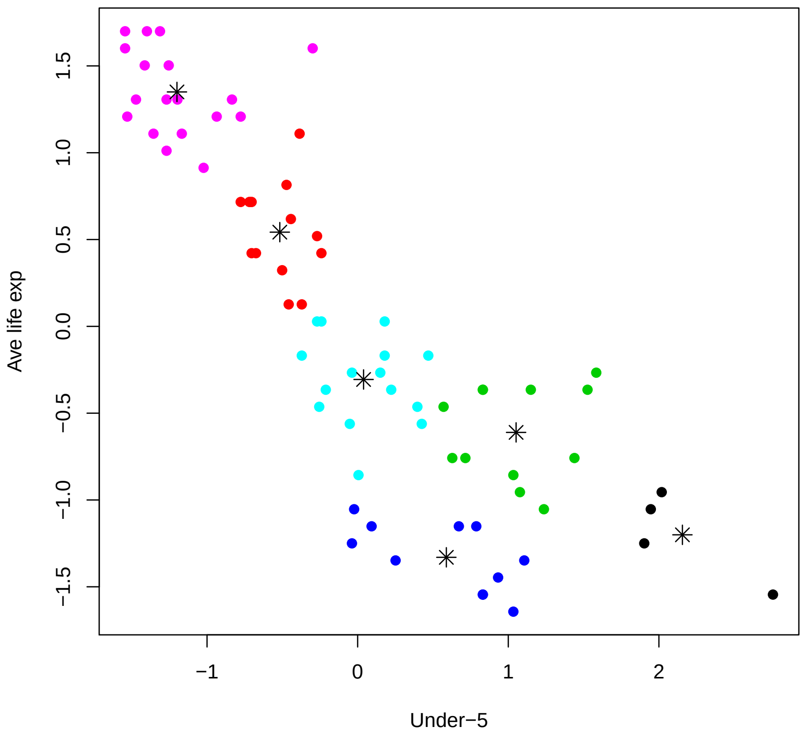


Figure 11.11: Scatter plot of data color coded by cluster data and cluster centers

Step 5 - Improving Model Performance

Use of K-means requires the determination of the number of clusters. In practice, there is no simple answer as to how to choose this value. It is useful to try different approaches.

A scree plot, as illustrated for our sample in Figure 11.12, can often be useful. It plots the within cluster sum of squares against the number of clusters. The idea is to look for a sharp bend or elbow. A sharp "bend" indicates a radical change in slope. In other words, although adding clusters beyond the bend will reduce the within cluster sum of squares, it will do so at a less rapid rate. The optimal value of k occurs at the elbow.

In practice, the identification of the bend is often difficult to discern. However, in Figure 11.12, it appears to occur at k = 2.

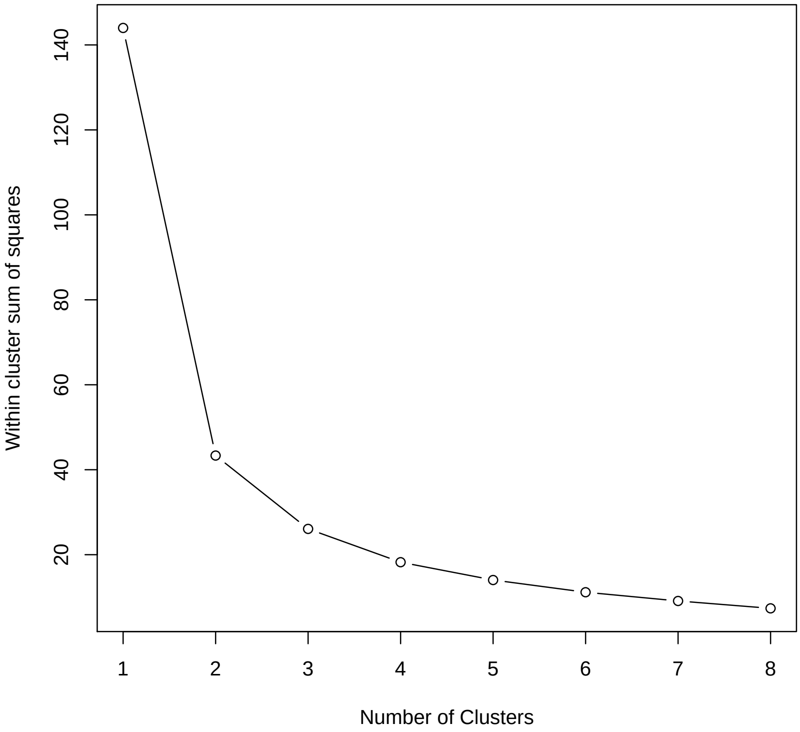


Figure 11.12: Scree plot for the sample data

Using silhouette length to choose k

A complementary approach is to specify a potential range for k, and use a metric such as the average width of a silhouette plot to select the appropriate number of clusters. A silhouette plot measures how close each point in one cluster are to points in the neighboring clusters. It has a range of -1 to +1. A value close to +1 indicates an observation is far apart from the neighboring clusters, a value of zero indicates the observation is on the decision boundary; and negative values suggest the observation might have been assigned to the wrong cluster.

If we assume, the correct number of clusters lies between 2 and 10, we can call the `kmeansruns` function setting `krange` to range between 2 and 10. The `criterion` argument is used to specify the silhouette width as the selection criteria. The model is run using 100 random starting points:

```
set.seed(2018)
fitasw <- kmeansruns(x,krange=2:10,
critout=TRUE,
runs=100,
criterion="asw")
```

The output reports the average silhouette width for each cluster:

```
2  clusters  0.5705245
3  clusters  0.4656673
4  clusters  0.4391071
5  clusters  0.4344615
6  clusters  0.4311283
7  clusters  0.4543943
8  clusters  0.4260914
9  clusters  0.4406647
10  clusters  0.4312051
```

The optimal number of clusters is selected as the maximum value. The maximum value of 0.57 occurs for two clusters.

Thus, we have the same result as that obtained using the scree plot.

Silhouette and Cluster plots

Figure 11.13, shows the silhouette plots for two clusters. The bars, break cleanly into two groups, and they are all positive, with no negative values. This is further illustrated in a cluster plot of Figure 11.14. It plots the data using the first and second principal components.

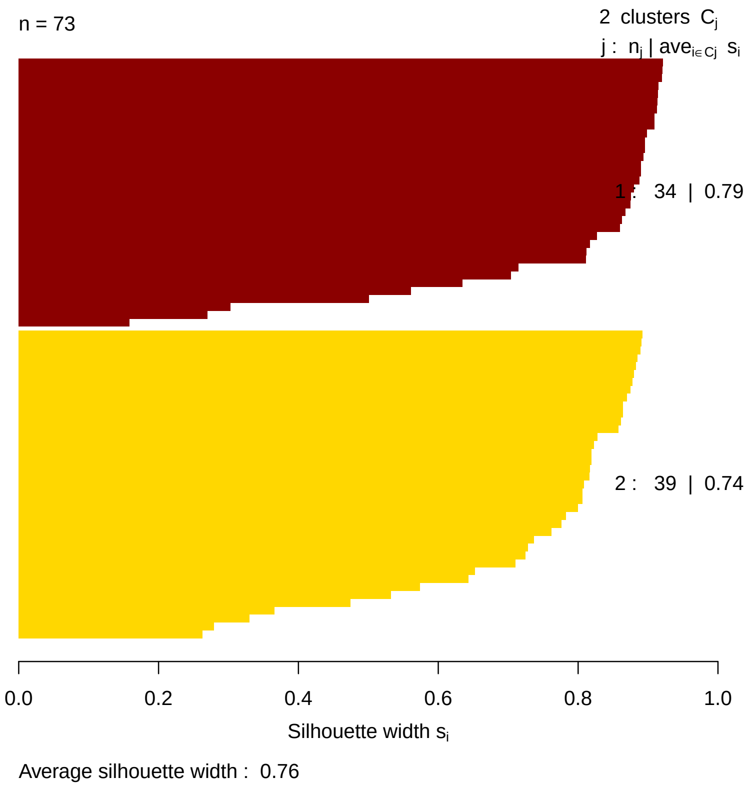


Figure 11.13: Silhouette plot for two clusters

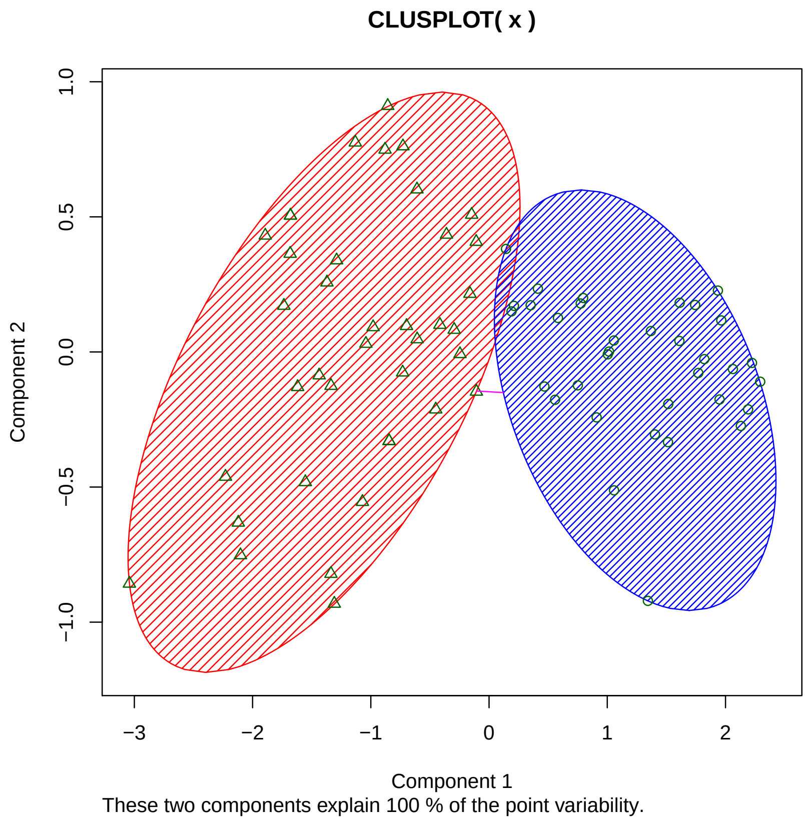


Figure 11.14: Cluster plot

NOTE... ✍

You can create a cluster plot like Figure 11.8 and Figure 11.14 via the `clusplot` function in the `cluster` package.

Working with the optimal model

To make progress, we really need to know a little about the empirical characteristics of the two groups. The centroids provide some insight, they are stored in `fitasw`:

```
round(fitasw$centers,3)

  Under-5 Ave life exp
1  -0.859        0.919
2   0.749       -0.801
```

The values inform us that the first cluster represents those countries that have a lower than average infant mortality rate, and a higher than average life expectancy. The second cluster is the exact opposite. It contains countries with a higher than average infant mortality, and lower than average life expectancy. Therefore, we have a very clean and intuitive interpretation of the two clusters.

Recall, we used scaled data, let's look at the implications using the unscaled data. First, let's combine the countries by clusters, and the actual raw `unicef` data into a single dataframe:

```
com_view<-cbind(fitasw$cluster,unicef)
colnames(com_view)<-c("cluster",
"Median_Under_5",
"Median_Life_Exp")
head(com_view)
```

```
            cluster  Median_Under_5  Median_Life_Exp
Afghanistan       2             257               43
Angola            2             260               45
Armenia           1              35               73
Azerbaijan        1             105               72
Bangladesh        1              77               60
Benin             2             158               54
```

To make things a little easier, I've use the `colnames` function to create column names that are a little easier to work with.

Next, let's look at the median infant mortality and median life expectancy for each of the two clusters. The `table` function comes in handy here:

```
tab1<-aggregate(data = com_view,
Median_Under_5~ cluster,
median)

tab2<-aggregate(data = com_view,
Median_Life_Exp~ cluster,
median)

med<-cbind(tab1,tab2)
med<-med[,-3]
med
  cluster Median_Under_5 Median_Life_Exp
1       1             72            66.5
2       2            175            48.0
```

Cluster 1, has a median life expectancy of 66.5 years, with an infant mortality of 72. Contrast this with cluster 2, it has a life expectancy of only 48 years, with more than 2.4 times the infant mortality of the countries in cluster 1.

We can use a similar method to view the within cluster variance of each feature:

```
var1<-aggregate(data = com_view,
Median_Under_5~ cluster,
var)

var2<-aggregate(data = com_view,
Median_Life_Exp~ cluster,
var)

var<-cbind(var1,var2)
var<-var[,-3]
round(var,3)
```

```
  cluster Median_Under_5 Median_Life_Exp
1       1        975.316          31.087
2       2       2299.798          22.726
```

The number that immediately jumps off the page is the very high variability of infant mortality rates in cluster 2. This might suggest policy interventions focused on reducing this variability may have a significant chance of improving the situation in these countries.

The variability of life expectancy, for cluster 2 is less than that for cluster 1. In other words, you are more certain to die by 48 in cluster 1, than you are to die by 66.5 in cluster 2. This may suggest policy aimed at increasing overall life expectancy for countries in cluster two should be a top international public health priority.

Next, we want to see the countries in each cluster, to ensure we agree with their categorization. A fast way to do this is to use the `lapply` function with `fit$cluster`. Here is how to do this:

```
country <-rownames(x)
final_clusters <- lapply(1:2,
function(cluster)
country[fitasw$cluster==cluster])
```

The R object `final_clusters` contains the countries in each cluster. Here are the countries in the first cluster:

```
final_clusters[1]
[[1]]
 [1] "Armenia"          "Azerbaijan"
 [3] "Bangladesh"       "Bhutan"
 [5] "Bolivia"          "China"
 [7] "Comoros"          "Georgia"
 [9] "Ghana"            "Guyana"
[11] "Honduras"         "India"
[13] "Indonesia"        "North_Korea"
[15] "Kyrgyzstan"       "Laos"
```

```
[17] "Moldova"              "Mongolia"
[19] "Myanmar"              "Nepal"
[21] "Nicaragua"            "Pakistan"
[23] "Papua_New_Guinea"     "Solomon_Islands"
[25] "Sri_Lanka"            "Sudan"
[27] "Syria"                "Tajikistan"
[29] "Turkmenistan"         "Ukraine"
[31] "Uzbekistan"           "Vietnam"
[33] "Yemen"                "Yugoslavia"
```

And here are the countries in the second cluster:

```
final_clusters[2]
[[1]]
 [1] "Afghanistan"            "Angola"
 [3] "Benin"                  "Burkina_Faso"
 [5] "Burundi"                "Cambodia"
 [7] "Cameroon"               "Centrafrique"
 [9] "Chad"                   "Congo"
[11] "Congo_DR"               "Cote_d'Ivoire"
[13] "Djibouti"               "Equatorial_Guinea"
[15] "Eritrea"                "Ethiopia"
[17] "Gambia"                 "Guinea"
[19] "Guinea-Bissau"          "Haiti"
[21] "Kenya"                  "Lesotho"
[23] "Liberia"                "Madagascar"
[25] "Malawi"                 "Mali"
[27] "Mauritania"             "Mozambique"
[29] "Niger"                  "Nigeria"
[31] "Rwanda"                 "Senegal"
[33] "Sierra_Leone"           "Somalia"
[35] "Tanzania"               "Togo"
[37] "Uganda"                 "Zambia"
[39] "Zimbabwe"
```

From the results above we can see that the use of 2 clusters leads to a well defined set of countries that are relatively distinct when it comes to infant mortality and life expectancy.

It is only natural to think about the next steps from this sort of output. You could start to initiate further research to understand why these countries differ, and use it to generate ideas as to what to do about it.

Limitations of K- Means Clustering

The K-means clustering algorithm will always converge to a solution, but it is likely to find a solution that is only locally optimal for a given set of data. The solution is sensitive to the initial centroids chosen. It is important to remember that it may not to find the optimal solution on a single run of the algorithm. Therefore, multiple runs of the model are very important.

Outliers and noisy data can unduly influence the cluster centroids. Such observations should be identified and removed from the dataset prior to using the K-means algorithm.

Summary

K-means clustering is a very popular tool for cluster analysis due to its conceptual simplicity and computational efficiency. It divides a sample into clusters that minimize the within cluster sum of squares. It remains one of the most popular machine learning algorithms, and is used in numerous fields.

In the next chapter, we discuss several tips to help enhance the performance of machine learning algorithms.

Suggested Reading

- **Automatic Shags:** Sakamoto KQ, Sato K, Ishizuka M, Watanuki Y, Takahashi A, Daunt F, et al. (2009) Can Ethograms Be Automatically Generated Using Body Acceleration Data from Free-Ranging Birds? PLoS ONE 4(4): e5379. https://doi.org/10.1371/journal.pone.0005379

- **Fibromyalgia:** Docampo E, Collado A, Escaramís G, Carbonell J, Rivera J, Vidal J, et al. (2013) Cluster Analysis of Clinical Data Identifies Fibromyalgia Subgroups. PLoS ONE 8(9): e74873. https://doi.org/10.1371/journal.pone.0074873

- **Rasberry Crazy Ant:** Gotzek D, Brady SG, Kallal RJ, LaPolla JS (2012) The Importance of Using Multiple Approaches for Identifying Emerging Invasive Species: The Case of the Rasberry Crazy Ant in the United States. PLoS ONE 7(9): e45314. https://doi.org/10.1371/journal.pone.0045314

Other

- **Nylanderia fulva (Mayr):** A review of the biology, taxonomy, ecology of the ant can be found in Wang, Zinan et al. "A Review of the Tawny Crazy Ant, Nylanderia Fulva, an Emergent Ant Invader in the Southern United States: Is Biological Control a Feasible Management Option?" Ed. Mary L. Cornelius. Insects 7.4 (2016): 77. PMC. Web. 25 Apr. 2017.

- **Original use of K-means:** Lloyd SP. Least squares quantization in PCM. IEEE Transactions on Information Theory. 1982;28(2):129–137.

Chapter 12

Tips to Enhance Performance

CHOOSING the very best model for a particular machine learning challenge requires the selected model to perform well on new, as yet unseen observations; in other words, it will need to generalize well. Volumes have been written on this challenging task.

In this chapter, we distill down some of the very best ideas and techniques. You will:

- Learn about Occam's Razor alongside several useful interpretations.
- Identify the cause and dangers of data snooping.
- Discover the practical implications of the No Free Lunch Theorem.
- Review the bias-variance trade-off.
- Learn about, and perform cross validation using R.

As you read through chapter, you will encounter several ideas, tips and tricks that can help you choose, construct and deploy useful machine learning algorithms.

Clarifying Occam's Razor

In my very first lecture at graduate school, the brilliant mathematics professor who was to share with us the inner working of measure theory, began with the words:

> "*Lex parsimoniae. Entia non sunt multiplicanda praeter necessitatem.*"

Which roughly translates to:

> "*The law of parsimony or economy. Entities should not be multiplied unnecessarily.*"

This principal is the famous Occam's Razor; named in honor of the late middle ages scholar William of Ockham.

Useful Interpretations

Here are several interpretations of Occam's Razor relevant to empirical modeling:

- If a smaller set of attributes fits the observations sufficiently well use those attributes. Avoid "*stacking*" additional attributes to improve the fit of a model.
- Select the modeling approach that makes the fewest assumptions.
- Only retain that subset of assumptions which make a clear difference to the predictions.
- In selecting between models that explain a phenomenon equal well, it is usually best to start with the simplest one.
- If two or more models have the same prediction accuracy, choose the simplest model.

Occam's razor works as a guiding principle. It suggests taking the most parsimonious option in any given situation with the hope that it will be the right choice most of the time. Although, like any rule of thumb, it may not always be optimal.

Model Clarity

The appealing thing about Occam's Razor, as a rule of thumb, is that it neatly captures the idea that in building a machine learning model you should try to find the smallest set of features which provide an adequate description of the data. This idea ties nicely into the interpretability of a model. In general, the more parameters and the greater the complexity of a model, the more difficult it is to interpret.

Interpretability, in terms of some underlying theory, can be an intoxicating appeal, especially if like me you were formally trained in disciplines with a heavy bias towards deductive theories.

However, ease of interpretability may or may not be important. It depends essentially on the domain of application. For example, a model which purports to explain economic patterns might be designed to be understood through the lens of economic theory. On the other hand, a real-time harbor porpoise identification and recognition model has no such constraints. Model parsimony may be a worthwhile goal, but it is one that cannot always be achieved.

Understanding Data Snooping

Data snooping refers to unwittingly assigning meaning to spurious correlations or patterns in a sample. You may have come across the idea in your statistics 101 class when the lecturer presented an example involving the local stork population and number of babies born. Another popular example involves the late actress Elizabeth Taylor's marriages. She remarried in

1951, 1953, 1958, 1960, 1965, 1976, and 1977; all coincided with stock market gains. But would you have comfort in a model that told you to invest when the celebratory of the day gets remarried? Similarly, a model that predicts the number of babies delivered in a local hospital as a consequence of the ebb and flow of the local stork population should be viewed with suspicion.

The Danger of Snooping

Several times over my career, I have been brought in by a large investment firm to figure out what went wrong with a quantitative investment model. Here is what frequently occurred: The head of the organization, typically a administrative/ MBA type, forms a small team of freshly minted Economics/Finance PhD's to build an automatic investment fund. The goal is to apply the very latest in quantitative techniques to make money.

After several months of intensive research, the head of the team announces they have built the model, tested it extensively on "real data" and are "ready to go". The model is "seeded" and turned on. Within a matter of days (in the worst case) it has lost millions of dollars; Over the course of the next several months even more money is lost. After around 18 months the model is terminated, never recouping the original losses, and the team of "all stars" is disbanded.

What went wrong? In every single case I can recall, the problem pointed straight back to extensive data snooping and ignorance surrounding its impact on model selection, and the ability to accurately predict future observations. The challenge is that in practice data snooping is not as obvious as the textbook examples involving Elizabeth Taylor or Storks and babies.

The Benefit of a Validation Set

Whenever a "optimal" model is obtained by an extensive search over the sample data, there is always the danger that the observed good performance may be partially or completely due to chance. This is because the probability that a "*good*" result arises by chance grows with the number of combinations tested - if you sequentially flip a sufficiently large number of coins, a sequence that always comes up heads will eventually emerge.

To reduce the likelihood of data snooping, at the very minimum divide the sample randomly into a training and test set. The training sample (sometimes called in-sample) contains the data that will be used to train the various models. The test sample (also known as the out-of-sample set) is used to test the models selected during the training phase. For illustrative purposes, we followed this approach throughout this text.

However, if possible you should divide the sample into three sets, with the third set called the validation set. It is used to assess the final selected model. The use of training, test and validation samples reduces even further the likelihood that the optimum model will suffers from severe data snooping bias.

Practical Implications of the No Free Lunch Theorem

Roughly stated, the No Free Lunch theorem states that:

> *In the lack of any prior knowledge on average all predictive algorithms that search for the minimum classification error have identical performance according to any measure.*

This simple means that a data sample that can be classified well by a specific machine learning algorithm, can also be classified well by other machine learning algorithms. For example, the Naive Bayes classifier might perform equally well as logistic regression.

Don't Fall in Love

Don't fall in love with a specific class of models. I once worked for an individual who would only use econometric (regression) models. As Professor George Box warns:

> "*Statisticians, like artists, have the bad habit of falling in love with their models.*"

However, falling in love with a model (or type of model) is very dangerous. You must exhibit the qualities of Odysseus in Homer's Odyssey, no matter how beautiful a model is to you, or how much time you have invested in it, do not be seduced by the Sirens song.

All Models Are Limited

I once worked for a very wise professor, who for a time, left the academic world in order to seek a fortune in the investment industry. Such moves are rarely totally satisfactory for the academic type; and within a few years the wise professor was back in the university classroom with the horrors of corporate cubical life a receding memory. In building predictive models, the wise professor clearly understood the no free lunch theorem:

> "*A single machine learning model just won't do.*" the wise professor would say. "*We must build an array of models, so that when one fails to work, the others might.*"

Any learning algorithm has a limited scope of application. No learning algorithm can be guaranteed to succeed on all learnable tasks. Build you toolkit to contain a wide variety of learning algorithms. Then try a variety on your data.

NOTE... ✍

If your toolkit is empty or needs a little boost pick up a copy of my book 92 Applied Predictive Modelling Techniques in R.

Work with Domain Experts

Domain expertise is the crucial ingredient, but often missed, by eager data scientists. As twenty-five-year business veteran of analytics Micheal Koukounas explains in his excellent book Real-World Analytics:

> "*At the time, I was managing a team of data scientists, really smart mathematicians and statisticians. One Monday morning in the middle of the fraud attack, I came into the office to find that one of my team members had spent the weekend building a fraud model to predict which of these transactions had a high probability of being fraudulent...He had constructed a sophisticated neural net model that performed very well...Unfortunately,...we could not use the model...*[It] *worked well in the laboratory environment but was not, to our regret, practical in the real world...In order for it to work, the petroleum company would have had to invest millions of dollars...Moreover, the company would have had to mandate all their gas stations and franchisees to invest thousands of bucks in net point-of-sales and communication equipment...*"

You can avoid this type of potentially costly blunder by seeking the assistance of domain experts as you develop your models.

Explaining the Bias Variance Trade-off

The bias variance decomposition is a useful tool for understanding classifier behavior. It turns out the expected misclassification rate can be decomposed into two components, a reducible error and irreducible error:

$$\tilde{R}_{emp}(\theta) = \sum_{x} P(x) \left(\overbrace{\sigma_x^2}^{\text{irreducible error}} + \underbrace{bias_x^2 + variance_x}_{\text{Reducible error}} \right)$$

Irreducible error or inherent uncertainty is associated with the natural variability in the phenomenon under study, and is therefore beyond our control. I like to think of it as the background noise inherent in the system.

For example, if you are going to build an age classification model for Emperor Penguin's using weight and height as attributes, you will expect to see some natural variation between the weight and height of all one year old birds.

Interpreting Reducible Error

Reducible error, as the name suggests, can be minimized. It can be decomposed into error due to squared bias and error due to variance. Figure 12.1 illustrates the idea of bias and variance using Kung Fu throwing Stars.

The object is to throw these lethal weapons accurately, on target, to hit the bulls-eye every time. A novice might throw the darts with high variance and high bias (bottom right panel); whilst the Sifu Master, who has studied the art for many years, will hit the target every time (bottom left panel), and therefore has low variance and low bias. The hope is that our model exhibits the characteristics of the Sifu Master.

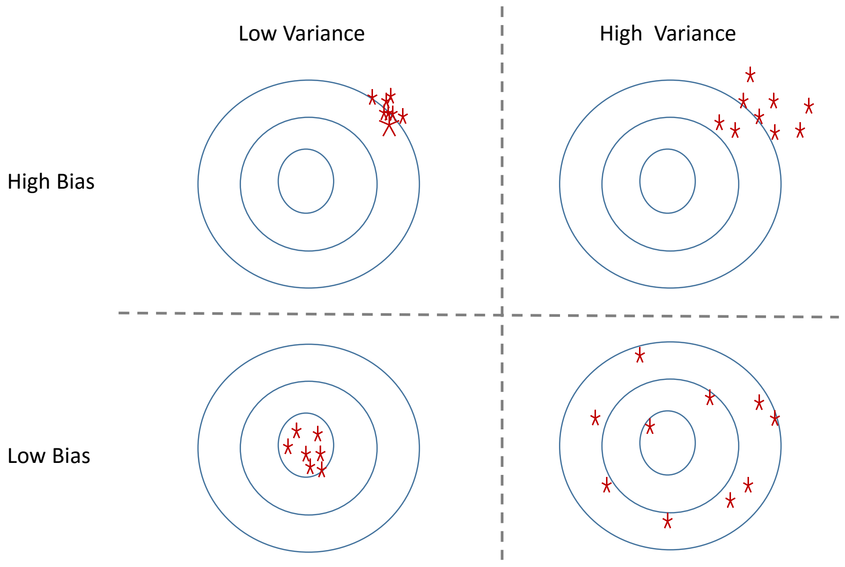


Figure 12.1: Kung Fu throwing star's illustration of bias and variance

Simplifying Bias

Bias relates to the ability of your model to approximate the actual observations. If on average, you model makes the prediction accurately, it has low bias. If on the other hand, no matter how many observations you include, it tends to over or undershoot, it is biased.

Statisticians tend to think of bias as a form of model selection error, and this is helpful because if your model perfectly represented the sample data, the bias would be zero. Bias is large if your model produces predictions that are consistently wrong.

For example, if we select a classifier such as Linear discriminant analysis for the decision boundary of Figure 8.2, the bias can be expected to be high because it can only model a linear hyperplane. Therefore, a large number of points will be consistently misclassified.

Scholars Christopher Manning, Prabhakar Raghavan and

Hinrich Schütze in their excellent book, *Introduction to information retrieval*, explain:

> *"We can think of bias as resulting from our domain knowledge (or lack thereof) that we build into the classifier. If we know that the true boundary between the two classes is linear, then a learning method that produces linear classifiers is more likely to succeed than a nonlinear method. But if the true class boundary is not linear and we incorrectly bias the classifier to be linear, then classification accuracy will be low on average."*

The temptation might be to always select a non-linear classifier by default, say for example a support vector machine. Notice that such a strategy would violate Occam's Razor (see page 310); and in practice would introduce bias if the true decision plane was linear by overfitting the data. In other words, such models would still be subject to model selection error.

Illuminating Variance

Whilst bias is a measure of classification accuracy, variance is a measure of classification consistency. It measures the stability of the predictions of your machine learning model. Low variance implies high consistency of the classifications. High variance implies low consistency.

In turns out that linear classification models tend to have low variance because for different training samples from the same underlying probability distribution (or population), they produce similar decision hyperplanes. An algorithm such as KNN has low bias because it does not assume anything about the distribution of the data points. However, it has high variance, because it will change its prediction in response to the composition of the training sample.

A trade-off

Models that exhibit low variance and high bias under-fit; whilst models that exhibit high variance and low bias overfit. Model building is often a trade-off between bias and variance. Frequently, if we select an algorithm to reduce bias we will often also increase variance. Therefore, the differences in performance between different learning algorithms can be interpreted as trade-off between bias and variance. This trade-off helps explain why there is no single universal machine learning method.

Practical steps

If the algorithm has high bias, the following actions might help:

- Add more features
- Try an alternative (more sophisticated model).

If the model has high variance try these suggestions:

- Use fewer features
- Increase the number of training examples.

The Secret of the Hold Out Technique

A handful of generations ago it was common practice to train a learning model and estimate the expected error rate on the very same sample. A researcher would collect data on multiple attributes, then run a classifier and happily observe a very low classification error rate. To the sound of much rejoicing a paper would be published highlighting the predictive power of the newly discovered model. However, researchers in several fields of scholarly activity soon observed that training an algorithm and evaluating its statistical performance on the same data tends to yield optimistic results. Today, validation techniques provide tools to better estimate the expected error rate.

In the hold out method the dataset is split into two non-overlapping groups - a training set used to train the classifier, and a test set used to estimate the error rate of the trained classifier. Professor Stone captures well the idea of a hold out sample when he states:

> "*An example of controlled division* [hold out validation] *is provided by the cautious statistician who sets aside a randomly selected part of his sample without looking at it and then plays without inhibition on what is left, confident in the knowledge that the set-aside data will deliver an unbiased judgment on the efficacy of his analysis.*"

Figure 12.2 illustrates the situation.

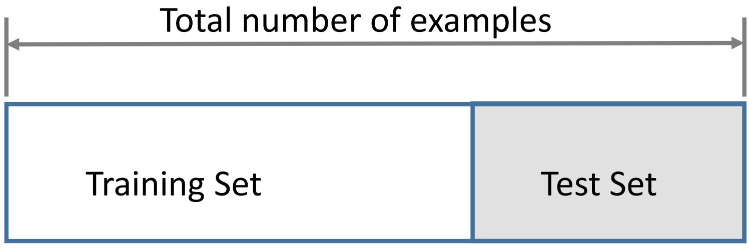


Figure 12.2: The Hold Out Method

Asymptotics

If the test set is a representative sample the error will be unbiased asymptotically (as the sample gets bigger). How close the test error is to the generalization error critically depends on the sample size.

NOTE... ✍

S.C Larson, back in 1931, used a hold out method by dividing a sample from the Mississippi Survey into a training and test set to provide more accurate error estimates. It would be several decades before this became standard practice.

Test and training set

How should you divide the data between the training set and the test set for hold out? There is no clear answer. I typically use anywhere from 10-50% of the available observations.

Caution is required if you use a low proportion of the overall sample. This is because hold out involves a single train and test experiment. The estimate of the error rate may be dependent on how the data is initially split. A sub-optimal split might result in a misleading error rate. The observations selected for inclusion in the test set might be dis-proportionally too easy or too difficult to classify.

In situations where you have a small sample you may not have sufficient observations to set aside for a test set. You may have to use all the available data simply to train the model.

NOTE... ✍

The limitations of the holdout technique can be damped by using cross validation.

The Art of Effective Cross Validation

In the textbook setting you have sufficient data to train and validate your models using a training set; and have ample data for assessing the quality of the model via a test set. Implicit in this scenario is a large and diverse sample from which accurate estimates of model parameters and error rates can be obtained.

However, in practice you will frequently face data poor situations, where scarcity of samples rather than abundance is the rule, and it will not be possible to set aside a portion of the dataset purely for testing. Cross validation techniques are useful here because they seek to extract the most information possible about the expected misclassification error.

k-fold Cross Validation

In k-fold cross-validation the training sample is partitioned into k equally sized segments. For each of the k segments, $k-1$ folds are used for training and the remaining one-fold used for testing. In this way, a total of k experiments can be performed and all the examples in the dataset are eventually used for both training and testing. Figure 12.3 demonstrates an example with $k = 5$. The darker sections indicate the test sets, while the lighter sections represent data used for training.

The estimate of the classification error rate is obtained as the average of the separate k estimates:

$$\tilde{R}^*_{test}(\theta) = \frac{1}{k}\sum_{i=1}^{k}\tilde{R}_{test,i}(\theta)$$

where $\tilde{R}_{test,i}(\theta)$ is the classification error on the ith test set.

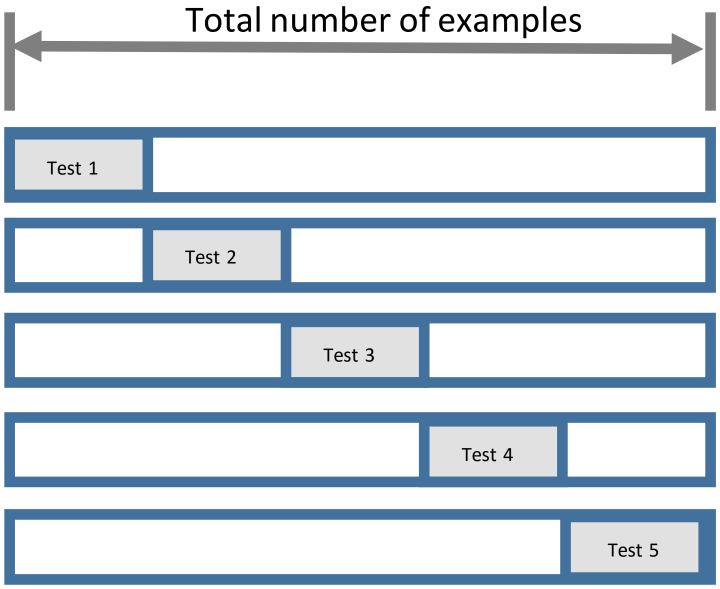


Figure 12.3: Five fold cross validation

Choosing the Number of Folds

In practice, the choice of the number of folds depends on the size of the dataset. For very sparse datasets, we may have to use leave-one-out cross validation (discussed below) in order to train the classifier on as many examples as possible.

In general, the larger the number of folds the smaller the bias of the true error rate estimator will be. Unfortunately, the variance of the error rate estimator tends to grow as the number of folds increase. In actual practice 5 or 10-fold cross-validation are most commonly specified.

Leave One Out Cross Validation

Back in the early 1970's Professor Mervyn Stone of University College London suggested the use of leave-one-out cross-validation for estimating model parameters and for assessing their predictive error. In this case, `k` is set equal to the number of examples in the training set.

At each iteration, nearly all the data except for a single observation are used for training. The model is then tested on that single observation - one or zero for success or failure, respectively. The results of the error estimates are then averaged to estimate the overall classification error.

Although leave one out cross validation typically results in an accurate estimate (almost unbiased) of the error, it tends to have high variance. However, it is a particularly useful tool when the sample data are scarce, for example in Bioinformatics where only a few dozen data samples might be available. Of course, given that the learning algorithm must be estimated for every data point, it comes at a larger computational cost than 5 or 10 fold cross validation.

Example - Classifying Purple Rock Crabs with Naive Bayes

Let's pull some of these ideas together with an applied example in R. We use the `crabs` data set we saw previously in the chapter on logistic regression.

Step 1 – Collecting and Exploring the Data

First load the packages and data:

```
library("MASS")
data(crabs)
```

Step 2 – Preparing the Data

As we did earlier, first calculate the first and second principal components for use as the our sample features:

```
pca <- prcomp(crabs[,4:8],
center = TRUE,
scale. = TRUE)
pca1<-pca$x[,1]
pca2<-pca$x[,2]
data<-as.data.frame(crabs$sex)
colnames(data)<-c("sex")
data<-cbind(data,pca1,pca2)
```

The R object `data` contains the class labels and the first and second principal components.

We use a standard train - test split, with 150 of the 200 observations set aside for training. The remainder will be used for the test set.

```
set.seed(2016)
n=nrow(crabs)
train_size=150
train <- sample(1:n, train_size, FALSE)

train_sample<-data[train,2:3]
class_train<-data$sex[train]

test_sample<-data[-train,2:3]
class_test<-data$sex[-train]
```

Step 3 - Train Model using Train Set

Following on from the idea of testing multiple machine learning models on your data, let's use the naive Bayes model from the `klaR` package. We can compare the performance to the logistic regression model we developed earlier.

Since we only have 200 observation in total, we will perform leave on out cross validation on the training set data data. The `caret` package is your go to tool for cross validation:

```
library("klaR")
library("caret")
```

Cross validation is carried out using the `train` function. Here is how to use it:

```
fit<- train(train_sample,
class_train,"nb",
trControl=trainControl(method="cv",
number=train_size))
```

Let's spend a moment to run though this.

- The `train` function takes the sample data as the first argument.
- The second argument is the machine learning algorithm you want to use. We set it equal to `nb` for naive Bayes.
- The third parameter is used to set up cross validation. We set method = "cv" to indicate we want to perform cross validation.
- The parameter `number` stores the number of folds to use. For 10 fold cross validation, you would set number equal to 10; for five fold cross validation, you would set it equal to 5. We want to perform leave on out cross validation, so we set `number` equal to the number of observations in the training sample.

Step 4 - Evaluate Model Performance

Once the cross validation has finished, you can view the average class accuracy:

```
(round(fit$results$Accuracy,3))
[1] 0.947 0.933
```

On average, the model classifies both female and male crabs with an accuracy in excess of 90%.

You can also look at how the optimal model performed using the train set data via the `table` and `predict` functions:

```
pred<-predict(fit$finalModel,
train_sample)$class
table(pred,class_train)
    class_train
pred  F  M
   F 66  4
   M  4 76
```

Of the 150 examples, the optimal cross validated model incorrectly classifies 4 male and 4 female crabs. This leads to an overall accuracy of 94.6%. Not bad, and similar to the model we developed using logistic regression.

NOTE... ✍

If you see the message: "`There were missing values in resampled performance measures.`" It indicates you may have a re-sample where one of the outcome classes has zero samples.

Step 5 - Assessing Test Set Performance

Now, let's see how the model performed on the test set:

```
pred<-predict(fit$finalModel,
test_sample)$class
table(pred,class_test)
   class_test
pred  F  M
   F 28  0
   M  2 20
```

Again, performance is good, and exactly in line with that achieved by our logistic regression model. The performance numbers give us some support for the idea that NBC or logistic regression can be used to classify the sex of purple rock crabs.

Summary

This chapter presented several ideas for assisting you create great machine learning algorithms. As we wrap up our introduction to machine learning with R, here is a simple rule of thumb to support your future success with empirical modeling. Split your sample into three parts:

1. the training set to build the classifier;

2. the validation set to fine tune model parameters and pick an algorithm;

3. the test set to estimate the future error rate. Remember to run the model through the test set once only or else report the results on every run.

Keep experimenting with new learning algorithms. A great place to keep up-to date with recent events is via my newsletter (visit www.AusCov.com to sign up). And be sure to grab your free copy of the latest issue of the R Journal (https://journal.r-project.org/).

Good Luck!

Suggested Reading

- **Bias:** Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. Vol. 1. No. 1. Cambridge: Cambridge university press, 2008.

- **Domain Expertise:** Koukounas, Michael.(2014). Real-World Analytics. Full Court Press.

- **Hold Out Method:** Stone, Mervyn. "Cross-validatory choice and assessment of statistical predictions." Journal of the royal statistical society. Series B (Methodological) (1974): 111-147.

Other Mississippi Survey:

- **Mississippi Survey:** Larson, Selmer C. "The shrinkage of the coefficient of multiple correlation." Journal of Educational Psychology 22.1 (1931): 45.

Congratulations!

You made it to the end!
Here are three things you can do next.

1. Pick up your FREE copy of **12 Resources to Supercharge Your Productivity in R** at *http://www.auscov.com*

2. Gift a copy of this book to your friends, co-workers, teammates or your entire organization.

3. If you found this book useful and have a moment to spare, I would really appreciate a short review. Your help in spreading the word is gratefully received.

I've spoken to thousands of people over the past few years. I'd love to hear your experiences using the ideas in this book. Contact me with your stories, questions and suggestions at *Info@NigelDLewis.com*.

Dr. N.D. Lewis

Good luck!

P.S. Thanks for allowing me to partner with you on your machine learning journey.

Index