AI Sciences

Machine Learning with R

STEP BY STEP GUIDE FOR NEWBIES

AI SCIENCES

Dominic Lordy

MACHINE LEARNING WITH R

Step By Step Guide For Newbies

Dominic Lordy

How to contact us

If you find any damage, editing issues or any other issues in this book contain please immediately notify our customer service by email at:

contact@aisciences.com

Our goal is to provide high-quality books for your technical learning in computer science subjects.

Thank you so much for buying this book.

Preface

“Some people call this artificial intelligence, but the reality is this technology will enhance us. So instead of artificial intelligence, I think we'll augment our intelligence.”

——Ginni Rometty

The main purpose of this book is to provide the reader with the most fundamental knowledge of machine learning with R so that they can understand what these are all about.

Book Objectives

This book will help you:

- Have an appreciation for machine learning and an understanding of their fundamental principles.
- Have an elementary grasp of machine learning concepts and algorithms.
- Have achieved a technical background in machine learning and also deep learning

Target Users

The book designed for a variety of target audiences. The most suitable users would include:

- Newbies in computer science techniques and machine learning
- Professionals in machine learning and social sciences
- Professors, lecturers or tutors who are looking to find better ways to explain the content to their students in the simplest and easiest way
- Students and academicians, especially those focusing on machine learning practical guide using R

Is this book for me?

If you want to smash machine learning from scratch, this book is for you. Little programming experience is required. If you already wrote a few lines of code and recognize basic programming statements, you’ll be OK.

First Printing, 2017

Edited by Davies Company
Ebook Converted and Cover by Pixels Studio
Publised by Createspace Publishing

ISBN-13: 978-1720424604
ISBN-10: 1720424608

To my father and mother: David and Maria, I love you both.

Author Biography

Dominic is a computer scientist and a member of the R Foundation. He built ML's packages. He also has great interest to other data science languages such as Python. Now, he is a writer and educator. He also promotes some Machine Learning applications.

Table of Contents

Introduction

Before getting started you must know what R is, R is a programming language and in this language, there is no intuitive graphical user interface with buttons those you can click to perform different functions. However, in this kind of environment with a little practice it makes easy to quickly code scripts and methods for various statistical purposes. Well, you know in command prompts a line begins with > is considered as input and is most of the cases we don't conclude output, but you should try out command yourself and see what happens. If there is any command in command line that you don't want to execute simply press down arrow to clear the line and while pressing the up arrow will give you the previous executed command.

Getting Started

The first step is to download the packages from R website link is given http://www.r-project.org/. Once you have opened the website the menu is on the left, you have to click on CRAN to "Download packages". After that choose a location close at you. At MIT, you can go with University of Toronto under Canada. This drives you to directions on the best way to download R for Linux, Mac, or Windows.

Now once the R is open, if you want to figure out your current directory, you have to type

```
getwd ().
```

If you want to change the directory use setwd (). There is a common thing that you have to Note "C:" is only for windows and different in the case of MAC. The example is given below:

```
> setwd("C:\\Datasets")
```

Installing and Loading Packages

In R almost all the Functions are grouped into packages and when you start the R most of the packages are loaded automatically. The common functions are "base", "utils", "graphics" and "stats". Once the package is installed most of the essential and frequently used functions are installed with those packages. If you want to use other useful methods of R

language you must have to download and install additional packages. For your better understanding, let me explain this with an example, an important classification method Support Vector Machine is contained in a package called “e1071”. To install this package, there is a simple step click “Packages” in the top menu, then “Install package(s)...” When you have asked to select CRAN mirror, you have to choose a location close to you such as “Canada (ON).” Finally, the last step is to select “e1071”. Now you have to load this package and there is a very simple step to do that, simply type **library (e1071)** at the command prompt**. There is one thing that you need to remember you just to install package once but if you want to use it you have to load it every time you started R.**

Running Code

You can run any type of the code in R by simply typing different types of command in command line, but this does not allow you to save your code or repeat it and even does not allow you to share your code. Rather than this, go to the “File” on the top menu and click for the “New script”. Now a new window will open up which you can easily save an R file. Now if you want to execute the code that you have written in this window simple highlight the piece of the code that you wish to run and then press Ctrl-R on a PC or Command-Enter on a MAC. If you want to run the entire script that you written in the new window just make sure that your script window in on the top of all other windows then go to “Edit” and click “Run all”. All the lines that are run appear in red at the command prompt.

Help in R

All the functions in R very well documented in case if you want to study details of the respective function. For this if you need to find documentation of any particular function simply type “?” Followed directly by the function in command prompt. For example: “? Function name (In command prompt)”. If you want help on the “sum” method type? Sum. Once you have executed this command a pop window will appear that will contain details on both input and output of that respective function. If you are getting any error while executing this command it means that there is insufficient or invalid input, so for that case use the documentation to figure out how to call function properly.

Now if you want to run the certain algorithm in R but don’t know the name of the proper function simply do the google search with R plus

algorithm name usually brings up the information on which function to use.

Datasets

In Machine learning when you test any learning algorithm you should use a variety of different datasets, but in case of R it is different. Basically, R has its own datasets and if you want to view the list of that datasets type data () in the command prompt and list will appear. For example, you have seen a dataset of cars from the list and you want to load that data simple type data (Cars), and view the data by typing cars.

There is another useful source of data available at UCI Machine learning repository, this directory contains couple thousands of datasets and most of them are from real applications on science and business.

Name	Rows	Cols	Data
Iris	150	4	Real
Wine	178	13	Integer, Real
Haberman's Survival	306	3	Integer
Housing	506	14	Categorical, Integer, Real
Blood Transfusion Service Center	748	4	Integer
Car Evaluation	1728	6	Categorical
Mushroom	8124	119	Binary
Pen-based Recognition of Handwritten Digits	10992	16	Integer

You don't have to limit your datasets to only R datasets, you can download your own datasets from UCI and other useful resources and use R to study those datasets. One thing you need to remember is that in Housing and Mushrooms datasets, there is an additional class attribute column that is not included in the column counts.

Basic Functions

This section covers how to use basic R methods to create data table, how to analyze that tables and how to plot the data. This illustration is done with examples. If you want to see values of any variable, vector, or any matrix you only have to enter its name is command line; you have to do this often until you feel comfortable with how each data structure is being stored. If you want to see all the objects in your workspace type Is ().

Creating Data

Now to create date you have to make variable and assign values to them for example if you want to create the variable x and want to assign 1 value to this variable Type x <- 1. Now if you want to create vector [1, 2, 3, 4, 5], and want to call vector v. There are lots of methods to accomplish this task some are given below:

```
> v <- 1:5
> v <- c(1,2,3,4,5)
# c can be used to concatenate multiple vectors
> v <- seq(from=1,to=5,by=1)
> v0 <- c(0,0,0,0,0,0)
> v0 <- seq(from=0,to=0,length.out=6)
```

Now if you want to combine vectors into matrices using cbind and rbind. For example, if v1, v2, v3 and v4 are vectors with the same length, we can combine them in matrices by using them either as a columns or rows.

```
> v1 <- c(1,2,3,4,5)
> v2 <- c(6,7,8,9,10)
> v3 <- c(11,12,13,14,15)
> v4 <- c(16,17,18,19,20)
> cbind(v1,v2,v3,v4)
> rbind(v1,v2,v3,v4)
> matrix (v, nrow=4, ncol=5, byrow=TRUE)
```

There is another useful technique to name the columns and rows of the datasets using **colnames** and **rownames**. For this we have to save the matrix as matrix20 and once we have done this now we can name the

columns and rows.

```
> matrix20 <- matrix (v, nrow=4, ncol=5, byrow=TRUE)
> colnames(matrix20) <- c("Col1","Col2","Col3","Col4","Col5")
> rownames(matrix20) <- c("Row1","Row2","Row3","Row4")


> v [3] # third element of v
> matrix20[,"Col2"]
# Second column of matrix20
> matrix20["Row4",]
# Fourth row of matrix20
> matrix20["Row3","Col1"]
# Element in third row and first column of matrix20
> matrix20[3,1]
> length(v1) > nrow(matrix20) > ncol(matrix20)
```

From the beginning you have been working with external datasets, you will definitely need a function that will read in a data tables from a text file. For example, if you want to read in the Haberman's Survival dataset (from the UCI Repository). Use the read.table function: dataset

```
<-read.table("C:\\Datasets\\haberman.csv", header=FALSE, sep=",")
```

The first argument in this defines the location (full path) of the file. Now if the first row the data contains the columns names, then the second argument must be **header = true** otherwise it will be **header = false.** The argument of this function is a delimiter. For example if the data is separated by spaces of tabs, then the argument will be **sep = ""** and **sep = "\t"** respectively.

```
dataset <- read.csv("C:\\Datasets\\haberman.csv", header=FALSE)
```

If you want to write a table to the file use write.table. Type? write.table to see details about this function.

Sampling

In R language there are number of the functions for sampling from probability distributions. For example, there are several commands that will generate random vectors of the user specified length n from distributions. The common function in R (normal, exponential, poisson, uniform, binomial) with user-specified parameters. Apart from these there are other distributions as well.

```
> norm_vec <- rnorm(n=10, mean=5, sd=2)
> exp_vec <- rexp(n=100, rate=3)
 > pois_vec <- rpois(n=50, lambda=6)
> unif_vec <- runif(n=20, min=1, max=9)
> bin_vec <- rbinom(n=20, size=1000, prob=0.7)
> Sample (v, size=25, replace=FALSE)
```

Now if want to sample with the replacement set the **replace** argument with **True** value. There is another function to generate same random vector each time you can call one of the random functions listed below pick a "seed" for the random number generator using set.seed, for example set.seed(100).

Analyzing Data

Now if you want to analyze the data suppose you want to compute the mean, variance, standard deviation, and minimum, maximum, and sum of a set of numbers, you can use **mean, var, sd, min, max, and sum.** There two more functions to analyze the date for example rowSum and colSum to find the row and column sums of the matrix. Now if you want to compute the absolute value and square root of the given numbers use **abs** and **sqrt.**

In R like other programing languages you can also write if and else statements, and for and while loops. For example, given below there is example that will show the even numbers between 1 and 10

```
> for (i in 1:10){
+ if (i %% 2 == 0){
+ cat(paste(i, "is even.\n", sep=" "))
# use paste to concatenate strings + } + }


> which(v >= 0)
v > v[which(v >= 0)]
```

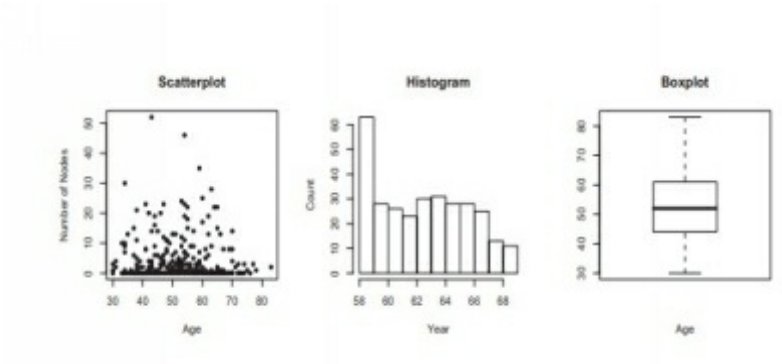
Plotting Data

To demonstrate the plotting functions, we will use Haberman's Survival data. In this data set each row represents the patients who had a surgery for breast cancer. The main three features of this datasets are: at the time of the surgery what is the age of patient, the year of the surgery and the

number of the positive auxiliary node that are detected. Here we will plot:

- Scatterplot of the first and third features
- Histogram of the second feature
- Boxplot of the first feature

```
> plot(dataset[,1], dataset[,3], main="Scatterplot", xlab="Age", ylab="Number of Nodes", pch=20)
> hist(dataset[,2], main="Histogram", xlab="Year", ylab="Count")
> boxplot(dataset[,1], main="Boxplot", xlab="Age")
```



Formulas

To express the form of a model there are certain functions that have formulas as part of their argument. This is explained with the help of the simple example. Suppose there is a response variable y and you have three independent variable x1, x2 and x3. If you want to show that y depends linearly on x1, x2 and x3 you would use y ~ x1 + x2 + x3, where y, x1, x2, and x3 are also column names in your data matrix. If you want to know the details of how to capture nonlinear models type? Formula.

Linear Regression

The most common approach in a statistical learning that everyone uses is linear regressions. In R we use lm functions to generate these models. The general form of linear regression model in R is given below:

$$Y = \beta 0 + \beta 1X1 + \beta 2X2 + \cdots + \beta kXk + \varepsilon,$$

where ε is mostly distributed with zero mean and some variance σ2.

```
> lm_model <- lm(y ~ x1 + x2, data=as.data.frame(cbind(y,x1,x2)))
> summary(lm_model)
```

Machine Learning Algorithms

There are lots of machine learning algorithm that can be implemented in R but in this section, we have covered the most commonly used ones. The list of these algorithms are given below:

- Prediction
- Apriori
- Logistic Regression
- K-Means Clustering
- AdaBoost
- Naive Bayes

Prediction

The basic approach we will use to implement this algorithm is that first we will generate the model using training data and then predict values for test data. In R if we want to make predictions we use **predict** function. If you want to see the documentation of this function go to the help pages for algorithm and then scroll through the content menu on the left side on help page to find corresponding **predict** function. Or simply type? predict.name, where name is the function corresponding to the algorithm. Basically, in this the first argument is the variable in we have saved our model and the second argument represents matrix or data frame of test data. Before making any prediction about the date we must examine the date first, by examining the data our aims are getting inside the data that means discovering the structure of data, behaviors of data, patterns in data, outliers and other information to help us choosing a forecasting model. A key assumption of time series prediction is that the data is Stationary. In stationary process joint probability distribution does not change over time. The tools to determine the stationarity of data include calculating the Auto-correlation of the data or using the Augmented Dickey–Fuller test. Another important factor in data which is to explore is to determine if our data follow a trend or contains a seasonal behavior. Some models that we have are not capable enough to accurately predict on the data which contains a trend or periodic pattern and thus one will typically remove these behaviorS through transforming the data.

There is one important point to remember when you call the function you

can just type **predict** instead of **predict.name.** The example is given below:

```
> predicted_values <- predict(lm_model, newdata=as.data.frame(cbind(x1_test, x2_test)))
```

Apriori

The basic purpose of using the Apriori algorithm is that it provides level wise search for frequent datasets. This algorithm is commonly used in machine learning as it used to find the associations between many variables. Mainly it is used by grocery stores, retailers or anyone else with large transactional database. You cannot run this algorithm directly in R you must have to install the **arules** package and load it. You can see the above section on how to install and load packages.

The function Apriori() will produce the best set of the rules for given transactional data. It also shows the support, confidence and lift of those rules. These three measures can be used to decide the relative strength of the rules. So here are the set of the formulas to generate these:

Support=Number of transactions with both A and BTotal number of transa

Confidence=Number of transactions with both A and BTotal number of tra

ExpectedConfidence=Number of transactions with BTotal number of transa

$$Lift = Confidence Expected\ Confidence = P(A \cap B) P(A).P(B)$$

The example given below will explain how to apply this algorithm on mushroom datasets. Before applying this algorithm, note that dataset must be binary incidence matrix; and their columns name must correspond to “items” that make up the “transactions”. Once you run the following commands summary of the results and a list of the generated rules will be shown:

```
> dataset <- read.csv("C:\\Datasets\\mushroom.csv", header = TRUE)
> mushroom_rules <- apriori(as.matrix(dataset), parameter = list(supp = 0.8, conf = 0.9))
> summary(mushroom_rules) > inspect(mushroom_rules)
```

Logistic Regression

To run this algorithm in R language you don’t have to install extra

packages.

```
> glm_mod <-glm(y ~   x1+x2, family=binomial(link="logit"), data=as.data.frame(cbind(y,x1,x2)))
```

Step by Step implementation of this algorithm on a dataset.

```
#Load data
data <- read.csv("data.csv")

#Create plot
plot(data$score.1,data$score.2,col=as.factor(data$label),xlab="Score-1",ylab="Score-2")
#Predictor variables
X <- as.matrix(data[,c(1,2)])

#Add ones to X
X <- cbind(rep(1,nrow(X)),X)

#Response variable
Y <- as.matrix(data$label)
#Sigmoid function
sigmoid <- function(z)
{
g <- 1/(1+exp(-z))
return(g)
}
#Cost Function
cost <- function(theta)
{
m <- nrow(X)
g <- sigmoid(X%*%theta)
J <- (1/m)*sum((-Y*log(g)) - ((1-Y)*log(1-g)))
return(J)
}
#Intial theta
initial_theta <- rep(0,ncol(X))

#Cost at inital theta
cost(initial_theta)
# Derive theta using gradient descent using optim function
```

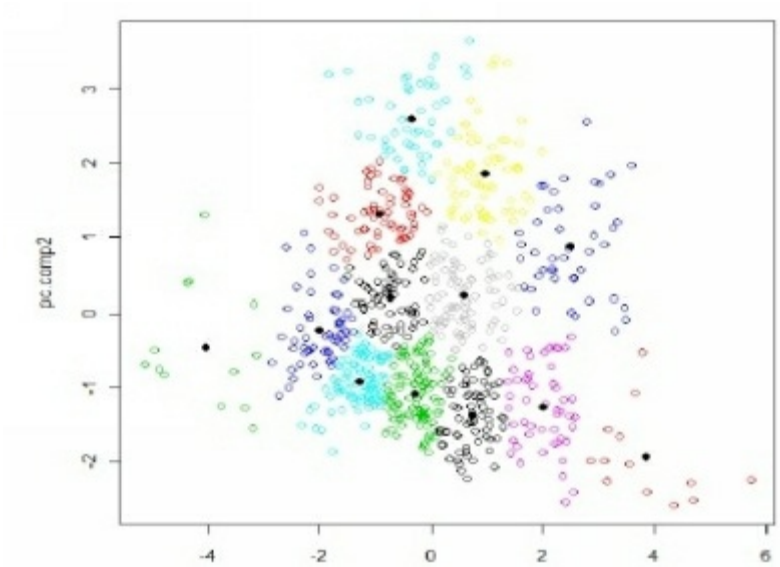
```
theta_optim <- optim(par=initial_theta,fn=cost)

#set theta
theta <- theta_optim$par

#cost at optimal value of the theta
theta_optim$value
# probability of admission for student
prob <- sigmoid(t(c(1,45,85))%*%theta)

x <- cbind(x_train,y_train)
# Train the model using the training sets and check score
logistic <- glm(y_train ~ ., data = x,family='binomial')
summary(logistic)
#Predict Output
predicted= predict(logistic,x_test)
```

K-Means Clustering

To implement this algorithm, you also don't have to install extra packages if x is the data matrix and m is the number of clusters then your command on command line will be:

```
> kmeans_model <- kmeans(x=X, centers=m)

# set the working directory
setwd("C:/STAT 897D data mining")
# comma delimited data and no header for each variable
RawData <- read.table("diabetes.data",sep = ",",header=FALSE)

responseY <- RawData[,dim(RawData)[2]]
predictorX <- RawData[,1:(dim(RawData)[2]-1)]

pca <- princomp(predictorX, cor=T) # principal components analysis using correlation matrix
pc.comp <- pca$scores
pc.comp1 <- -1*pc.comp[,1] # principal component 1 scores (negated for convenience)
pc.comp2 <- -1*pc.comp[,2] # principal component 2 scores (negated for convenience)

X <- cbind(pc.comp1, pc.comp2)
cl <- kmeans(X,13)
cl$cluster
plot(pc.comp1, pc.comp2,col=cl$cluster)
points(cl$centers, pch=16)
```

```
library(knn)
x <- cbind(x_train,y_train)
# Fitting model
fit <-knn(y_train ~ ., data = x,k=5)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```



Now to generate K-means clustering models for data characterization use h20.kmeans(). On dependent variable this algorithm dose not rely.

Example

```
1 > h2o.kmeans(training_frame = iris.hex, k = 3, x =
1:4)
2 Model Details:
3 ==============
4 H2OClusteringModel: kmeans
5 Model ID: K-means_model_R_1441989204383_30
6 Model Summary:
7 number_of_rows number_of_clusters number_of_
categorical_columns number_of_iterations within_
cluster_sum_of_squares
8 1 150 3
0 8
139.09920
9 total_sum_of_squares between_cluster_sum_of_squares
```

```
10 1 596.00000 456.90080
11 H2OClusteringMetrics: kmeans
12 ** Reported on training data. **
13 Total Within SS: 139.0992
14 Between SS: 456.9008
15 Total SS: 596
16 Centroid Statistics:
17 centroid size within_cluster_sum_of_squares
18 1 1 44.00000 43.34674
19 2 2 50.00000 47.35062
20 3 3 56.00000 48.40184
```

AdaBoost

There are various numbers of the boosting functions in R language. Given below is the one example in which implementation is done using decision trees as a base classifier. For this we have to install and load the **rpart** package. Aslo, the boosting the function **ada** is in the **ada** package. Suppose, x be the matrix of the features and labels be a vector of 0-1 class labels. The command is:

```
> boost_model <- ada(x=X, y=labels)

require(caret)

x <- cbind(x_train,y_train)

# Fitting model

TrainControl <- trainControl( method = "repeatedcv", number = 10, repeats = 4)

model<- train(y ~ ., data = x, method = "xgbLinear", trControl = TrainControl,verbose = FALSE)

OR

model<- train(y ~ ., data = x, method = "xgbTree", trControl = TrainControl,verbose = FALSE)

predicted <- predict(model, x_test)
```

Naive Bayes

To run this algorithm, you must have to install and load e1071 package.

```
> nB_model <- naiveBayes(y ~  x1 + x2, data=as.data.frame(cbind(y,x1,x2)))
```

```
library(e1071)
x <- cbind(x_train,y_train)
# Fitting model
fit <-naiveBayes(y_train ~ ., data = x)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Generalized Liner Model (GLM)

Basically, these types of models are used generally for many types of data analysis use cases. Most of the time some data can be analyzed by using this model, but there is an issue with these models in their results cannot be as accurate if the variables are complex. For instance, if the dependent variable has non-continuous distribution, if the effect of the predictor is not linear, so in both cases generalized linear model can produce better accuracy results as compared to general linear model.

The core concept behind these models is that they estimate regression models for the outcomes followed by the exponential distribution in general. They also include Poisson, binomial, gamma, tweedie distribution along with Gaussian distribution, so depending on the distribution and link function choice each depends on the different purpose either for prediction or classification. For developing linear model we have to generate generalized linear model using exponential distribution h20.glm().

Example

```
1 > prostate.hex <- h2o.importFile(path = "https://raw.
github.com/h2oai/h2o/master/smalldata/logreg/
prostate.csv" , destination_frame = "prostate.hex"
)
2
3 > prostate.glm<-h2o.glm(y = "CAPSULE", x = c("AGE","
```

RACE","PSA","DCAPS"), training_frame = prostate.
hex, family = "binomial", nfolds = 10, alpha =
0.5)
4 > prostate.glm@model$cross_validation_metrics
5 H2OBinomialMetrics: glm
6 ** Reported on cross-validation data. **
7 Description: 10-fold cross-validation on training data
8 MSE: 0.2093902
9 R^2: 0.1294247
10 LogLoss: 0.6095525
11 AUC: 0.6909965
12 Gini: 0.381993
13 Null Deviance: 513.8229
14 Residual Deviance: 463.2599
15 AIC: 473.2599
16 Confusion Matrix for F1-optimal threshold:
17 0 1 Error Rate
18 0 122 105 0.462555 =105/227
19 1 41 112 0.267974 =41/153
20 Totals 163 217 0.384211 =146/380
21 Maximum Metrics:
22 metric threshold value idx
23 1 max f1 0.312978 0.605405 216
24 2 max f2 0.138305 0.772727 377
25 3 max f0point5 0.400689 0.628141 110
26 4 max accuracy 0.400689 0.700000 110
5 max precision 0.998848 1.000000 0
28 6 max absolute_MCC 0.400689 0.357638 110

Data with R

Objects

Till now we have seen that R works with the objects that are characterized by their names and their content. R also works with the objects that have attributes which specify the kind of data represented by an object. In order to have a better understanding of these objects consider the variable having value 1, 2, 3: but this variable could be integer variable or it would be coding of categorical variables.

So, from the above statements it is clear that statistical analysis of this variable will not be same in both cases with R. So, the attributes we are telling us the important information about the object. In simple and general words' action of a function on the object depends on the attribute of the latter.

The all objects we have contains two attributes, mode and length. There are four types of the modes and it is the basic element of the objects, types are numeric, logical, complex and character. There are other types of modes that exists, but they don't represent the data, for example functions and expression. Now the other attribute length, which defines the number of the elements of the objects. So to display both attributes we can use the **mode and length** function.

```
> x <- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1
> A <- "Gomphotherium"; compar <- TRUE; z <- 1i
> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
[1] "complex"
```

Whatever the mode, missing data are represented by NA (not available).

A very large numeric value can be specified with an exponential notation:

```
> N <- 2.1e23
> N
[1] 2.1e+23
R correctly represents non-finite numeric value
> x <- 5/0
> x
[1] Inf
> exp(x)
[1] Inf
> exp(-x)
[1] 0
> x - x
[1] NaN
```

The input with the double quote is value of the mode character.

```
> x <- "Double quotes \" delimitate R’s strings."
> x
[1] "Double quotes \" delimitate R’s strings."
> cat(x)
```

Double quotes " delimitate R’s strings.
Now the variable of the mode character can be delimited with the single quotes.

```
> x <- ’Double quotes " delimitate R\’s strings.’
> x
[1] "Double quotes \" delimitate R’s strings."
```

Working Directory

The working directory is a very common and important concept if you have worked in any of the language environment. In the context of the data import and export the working directory is the default location for the file for which it can be read by the R. And it is also the default location of the file where it will be exported. So until now unless you specify some explicit path to work, working directory will come into the picture. Therefore you need to make sure that you are working in the correct directory, otherwise you can run into problems. So, in R you need to know

a couple of commands. First is the getwd command, which is used to get the current working directory. So you can use this command to check the directory in which you're working.

```
getwd() #get working directory
```

The second command is `setwd` command. That can be used to set the working directory as per your requirement.

```
#Set working directory

setwd("C:\DataScience\\R\\Working with Directory") #windows
setwd("C:/DataScience/R/ Working with Directory ") #windows
setwd("/Users/UserName/Documents/Folder") #on mac or linux
setwd(file.path("C:","DataScience","R"," Working Directory"))
```

Importing CSV Files

Let's take an example to see the usage of this function.

```
#Set working directory
setwd(file.path("C:","DataScience","R"," Working Directory"))

#Set file path
file <- file.path("data","SampleCSVFile.csv")
csv.data <- read.csv(file)
str(csv.data)
csv.data
```

Importing from Tables

```
#Set working directory
setwd(file.path("C:","DataScience","R"," Working Directory"))
#Set file path
file <- file.path("data","SampleTable.txt")
table.data <- read.table(file,
header=TRUE,
skip=1,
```

```
colClasses=c("character","factor","numeric","integer","integer"))
str(table.data)
```

Importing XML Files

```
#Set working directory
setwd(file.path("C:","DataScience","R"," Working Directory"))
#Set file path
file <- file.path("data","SampleXML.xml")
install.packages("XML")
library(XML)
xml.data <- xmlToDataFrame(file,
colClasses=c("character","integer","integer"),
stringsAsFactors=FALSE)
str(xml.data)
```

Importing from excel files

```
#Set working directory
setwd(file.path("C:","DataScience","R"," Working Directory"))
#Set file path
file <- file.path("data","SampleXLSX.xlsx")
install.packages("XLConnect")
library(XLConnect)
excel.data <- readWorksheetFromFile(file,
                        sheet=1,
                        startRow=2)
str(excel.data)
excel.data <- transform(my.data,
                student.gender = as.factor(student.gender),
                student.physics.marks = as.integer(student.physics.marks),
                student.chemistry.marks = as.integer(student.chemistry.marks))
str(excel.data)
#Other packages : xlsReadWrite, xlsx , gdata
```

Saving data

To write an object in a file we use a function in R that is write.table.

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
```

eol = "\n", na = "NA", dec = ".", row.names = TRUE,

col.names = TRUE, qmethod = c("escape", "double"))

x	the name of the object to be written
file	the name of the file (by default the object is displayed on the screen)
append	if TRUE adds the data without erasing those possibly existing in the file
quote	a logical or a numeric vector: if TRUE the variables of mode character and the factors are written within "", otherwise the numeric vector indicates the numbers of the variables to write within "" (in both cases the names of the variables are written within "" but not if quote = FALSE)
sep	the field separator used in the file
eol	the character to be used at the end of each line ("\n" is a carriage-return)
na	the character to be used for missing data
dec	the character used for the decimal point
row.names	a logical indicating whether the names of the lines are written in the file
col.names	id. for the names of the columns
qmethod	specifies, if quote=TRUE, how double quotes " included in variables of mode character are treated: if "escape" (or "e", the default) each " is replaced by \", if "d" each " is replaced by ""

To write object in a file in a very simple way there is a command in R write(x, file="data.txt") it can used where in this command X is the name of the object. We have two more commands in R while writing data in the file nc (or ncol), the purpose of this command is to define the number of columns in the file and the other one is appending (a logical), the purpose of this command is to add data without deleting the possible data we have in the file. In order the record the group of objects we have another command in R and that is save(x, y, z, file= "xyz.RData"). In R data can be later loaded to memory using this command load("xyz.RData"). The function save.image() is a short-cut for save(list =ls(all=TRUE), file=".RData").

Generating data

In order to generate random data, we use different R commands for example sequence of the integer from 1 to 30 can be generated as follows:

```
> x <- 1:30
```

Once this command is executed the resulting variable X has 30 elements. In order to perform simple arithmetic operations in R we can use these commands:

```
> 1:10-1
[1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
[1] 1 2 3 4 5 6 7 8 9
```

The function seq can generate sequences of real numbers as follows:

```
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Where in this data first number will indicate the beginning of the sequence and second number will indicate the END of the sequence and the third number is incremental in order to generate sequence.

```
> seq(length=9, from=1, to=5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

One can also type directly the values using the function c:

```
> c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

At this stage it is also possible to use method scan by simply enter data from the keyboard.

```
> z <- scan()
1: 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

10: Read 9 items

```
> z
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

The function rep creates a vector with all its elements identical:

```
> rep(1, 30)
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The function sequence creates a series of sequences of integers each ending by the numbers given as arguments:

```
> sequence(4:5)
[1] 1 2 3 4 1 2 3 4 5
> sequence(c(10,5))
[1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5
```

To generate regular series of the factors we use the gl method in R. The documentation of this method is as follows gl(k, n) where K is representing number of levels and N in representing replication at each level. So, two use this method two attributes can be used, length which describes the number of the data produced and other one is labels which defines the names of the levels of the factors. Example is as follows:

```
> gl(3, 5)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
> gl(3, 5, length=30)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

```
Levels: 1 2 3
> gl(2, 6, label=c("Male", "Female"))
[1] Male Male Male Male Male Male
[7] Female Female Female Female Female Female
Levels: Male Female
> gl(2, 10)
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels: 1 2
> gl(2, 1, length=20)
[1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels: 1 2
> gl(2, 2, length=20)
[1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Levels: 1 2
```

Finally, expand.grid() creates a data frame with all combinations of vectors or factors given as arguments:

```
> expand.grid(h=c(60,80), w=c(100, 300), sex=c("Male", "Female"))
h w sex
1 60 100 Male
2 80 100 Male
3 60 300 Male
4 80 300 Male
5 60 100 Female
6 80 100 Female
7 60 300 Female
8 80 300 Female
```

law	function
Gaussian (normal)	`rnorm(n, mean=0, sd=1)`
exponential	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (F)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomial	`rbinom(n, size, prob)`
multinomial	`rmultinom(n, size, prob)`
geometric	`rgeom(n, prob)`
hypergeometric	`rhyper(nn, m, n, k)`
logistic	`rlogis(n, location=0, scale=1)`
lognormal	`rlnorm(n, meanlog=0, sdlog=1)`
negative binomial	`rnbinom(n, size, prob)`
uniform	`runif(n, min=0, max=1)`
Wilcoxon's statistics	`rwilcox(nn, m, n)`, `rsignrank(nn, n)`

In statistics is easy to generate random data and R can provide the functionality. These functions are of the form rfunc(n, p1, p2, ...), where func indicates the probability distribution, n the number of data generated, and p1, p2, . . . are the values of the parameters of the distribution. The above table gives the details for each distribution, and the possible default values (if none default value is indicated, this means that the parameter must be specified by the user). Most of these functions have counterparts obtained by replacing the letter r with d, p or q to get, respectively, the probability density (dfunc(x, ...)), the cumulative probability density (pfunc(x, ...)), and the value of quantile (qfunc(p, ...), with $0 < p < 1$). The last two series of functions can be used to find critical values or P-values of statistical tests. For instance, the critical values for a two-tailed test following a normal distribution at the 5% threshold are:

```
> qnorm(0.025)
[1] -1.959964
> qnorm(0.975)
[1] 1.959964
```

For the one-tailed version of the same test, either qnorm(0.05) or 1 - qnorm(0.95) will be used depending on the form of the alternative hypothesis.

The P-value of a test, say χ

2 = 3.84 with df = 1, is:

```
> 1 - pchisq(3.84, 1)
```

```
[1] 0.05004352
```

Graphics with R

In this section we will discuss about the amazing feature of R, that how R deals with graphics. Basically, R offers a huge variety of graphics. In order to get an idea how it really works one can simply type a command demo(graphics) or demo(persp), but it is not possible to discuss in details about the graphics in R because it is one complete vast subject but we will cover some basic functions. In R each graphical function has a large number of the options, making the productions of graphics very flexible.

Now coming to the topic, the result of the graphical method cannot be assigned to an object in R but on contrary, it will send to graphic device, by graphic device we mean a graphics window or a file.

Let's start with the basic there are two types of graphical functions in R. the high-level plotting functions which create a new graph and on the other hand low level plotting function, whose purpose is to add elements to an existing graph. In R graphs are produced with respect to their graphical parameters which are defined by default and can be modified according to our use.

So, in this section later we will be covering first time how to manage graphics and graphical devices in R. After we will be discussing some important graphical functions and parameters.

Managing Graphics

In R when we execute any graphical function and we haven't open any graphical device R itself opens a graphical window and displays the graphs. To open a graphical device in R we may use appropriate functions. It depends on your operating system what type of graphical devices are available for you. The graphical windows are called X11 under Unix/Linux and windows under Windows. In all cases, one can open a graphical window with the command x11() which also works under Windows because of an alias towards the command windows(). To open a graphical device which is a file it totally depends on function format, postscript(), pdf(), png(),, So in order to find the list of available graphical device there is a command in R that is ?device.

All subsequent graphs are displayed on last open active graphical device.

In order to obtain the list of last active open graphical devices, there is a function in R dev.list().

```
> x11(); x11(); pdf()
> dev.list()
X11 X11 pdf
2 3 4
```

The figures displayed are the device numbers which must be used to change the active device. Now to know what an active device is type this command in R:

```
> dev.cur()
pdf
4
```

and to change the active device:

```
> dev.set(3)
X11
3
```

The function dev.off() closes a device: by default the active device is closed, otherwise this is the one which number is given as an argument to the function. R then displays the number of the new active device:

```
> dev.off(2)
X11
3
> dev.off()
pdf
4
```

In order to partition the active graphical devices in R we use the following function:

```
> split.screen(c(1, 2))
```

Once this command is executed it divides the screen in two parts with screen(1) and screen(2). In order to delete the last shown graph type erase.screen(). If you are using multiple graphical devices, then these functions are incompatible. They have limited use for example in order to explore the graphical data.

In order to partition the active graphic window into several parts we use layout partitioning functions. The main argument of this layout method is

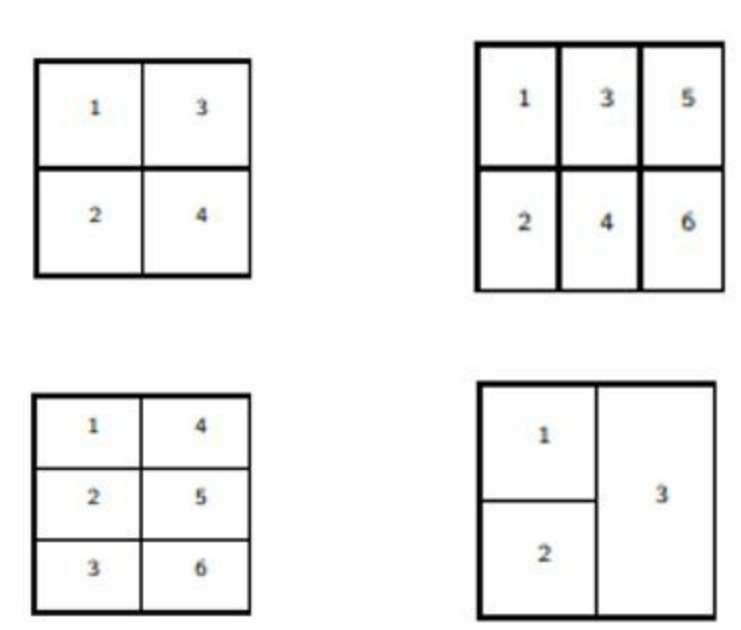
matrix indicating the integer number of sub-windows. For example, in order to divide the window in four parts.

```
> layout(matrix(1:4, 2, 2))
> mat <- matrix(1:4, 2, 2)
> mat
[,1] [,2]
[1,] 1 3
[2,] 2 4
> layout(mat)
```

Now if you want to see the layout partioning you can use R command

```
> layout.show(4)
> layout(matrix(1:6, 3, 2))
> layout.show(6)
> layout(matrix(1:6, 2, 3))
> layout.show(6)
> m <- matrix(c(1:3, 3), 2, 2)
> layout(m)
> layout.show(3)
```

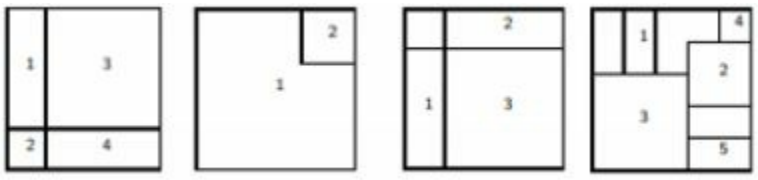
The results of the above codes are given below:



The All examples of portioning the windows we have discusses above we haven't use n byrow of matrix(), the purpose of this command is to

numbered column wise the sub windows. There is another command matrix(..., byrow=TRUE) this command makes the sub windows numbered row wise. We will see this with the help of the example:

```
matrix(c(2, 1,4, 3), 2, 2).
> m <- matrix(1:4, 2, 2)
> layout(m, widths=c(1, 3),
heights=c(3, 1))
> layout.show(4)


> m <- matrix(c(1,1,2,1),2,2)
> layout(m, widths=c(2, 1),
heights=c(1, 2))
> layout.show(2)


> m <- matrix(0:3, 2, 2)
> layout(m, c(1, 3), c(1, 3))
> layout.show(3)


> m <- matrix(scan(), 5, 5)
1: 0 0 3 3 3 1 1 3 3 3
11: 0 0 3 3 3 0 2 2 0 5
21: 4 2 2 0 5
26:
Read 25 items
> layout(m)
> layout.show(5)
```



Graphical functions

The basic high level graphical function in R, are listed below with the help of table. The details of each function are given in front of it. The purpose of sharing this list is that you can use any function according to your use.

`plot(x)`	plot of the values of `x` (on the y-axis) ordered on the x-axis
`plot(x, y)`	bivariate plot of `x` (on the x-axis) and `y` (on the y-axis)
`sunflowerplot(x, y)`	id. but the points with similar coordinates are drawn as a flower which petal number represents the number of points
`pie(x)`	circular pie-chart
`boxplot(x)`	"box-and-whiskers" plot
`stripchart(x)`	plot of the values of `x` on a line (an alternative to `boxplot()` for small sample sizes)
`coplot(x~y | z)`	bivariate plot of `x` and `y` for each value (or interval of values) of `z`
`interaction.plot (f1, f2, y)`	if `f1` and `f2` are factors, plots the means of `y` (on the y-axis) with respect to the values of `f1` (on the x-axis) and of `f2` (different curves); the option `fun` allows to choose the summary statistic of `y` (by default `fun=mean`)
`matplot(x,y)`	bivariate plot of the first column of `x` *vs.* the first one of `y`, the second one of `x` *vs.* the second one of `y`, etc.
`dotchart(x)`	if `x` is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column)
`fourfoldplot(x)`	visualizes, with quarters of circles, the association between two dichotomous variables for different populations (`x` must be an array with `dim=c(2, 2, k)`, or a matrix with `dim=c(2, 2)` if $k = 1$)
`assocplot(x)`	Cohen–Friendly graph showing the deviations from independence of rows and columns in a two dimensional contingency table
`mosaicplot(x)`	'mosaic' graph of the residuals from a log-linear regression of a contingency table
`pairs(x)`	if `x` is a matrix or a data frame, draws all possible bivariate plots between the columns of `x`
`plot.ts(x)`	if `x` is an object of class `"ts"`, plot of `x` with respect to time, `x` may be multivariate but the series must have the same frequency and dates
`ts.plot(x)`	id. but if `x` is multivariate the series may have different dates and must have the same frequency
`hist(x)`	histogram of the frequencies of `x`
`barplot(x)`	histogram of the values of `x`
`qqnorm(x)`	quantiles of `x` with respect to the values expected under a normal law
`qqplot(x, y)`	quantiles of `y` with respect to the quantiles of `x`
`contour(x, y, z)`	contour plot (data are interpolated to draw the curves), `x` and `y` must be vectors and `z` must be a matrix so that `dim(z)=c(length(x), length(y))` (`x` and `y` may be omitted)
`filled.contour (x, y, z)`	id. but the areas between the contours are coloured, and a legend of the colours is drawn as well
`image(x, y, z)`	id. but the actual data are represented with colours
`persp(x, y, z)`	id. but in perspective

For each function if you need any help on-line help is available in R.

- add=FALSE if TRUE superposes the plot on the previous one (if it exists)
- axes=TRUE if FALSE does not draw the axes and the box
- type="p" specifies the type of plot, "p": points, "l": lines, "b": points connected by lines, "o": id. but the lines are over the points, "h": vertical lines, "s": steps, the data are represented by the top of the vertical lines, "S": id. But the data are represented by the bottom of the vertical lines

- xlim=, ylim= specifies the lower and upper limits of the axes, for example
- with xlim=c(1, 10) or xlim=range(x)
- xlab=, ylab= annotates the axes, must be variables of mode character
- main= main title, must be a variable of mode character
- sub= sub-title (written in a smaller font)

Low-level plotting commands

In R there is list of graphical methods that affects an already existing graph and these are called low level plotting commands.

`points(x, y)`	adds points (the option `type=` can be used)
`lines(x, y)`	id. but with lines
`text(x, y, labels, ...)`	adds text given by `labels` at coordinates (x,y); a typical use is: `plot(x, y, type="n"); text(x, y, names)`
`mtext(text, side=3, line=0, ...)`	adds text given by `text` in the margin specified by `side` (see `axis()` below); `line` specifies the line from the plotting area
`segments(x0, y0, x1, y1)`	draws lines from points (x0,y0) to points (x1,y1)
`arrows(x0, y0, x1, y1, angle= 30, code=2)`	id. with arrows at points (x0,y0) if `code=2`, at points (x1,y1) if `code=1`, or both if `code=3`; `angle` controls the angle from the shaft of the arrow to the edge of the arrow head
`abline(a,b)`	draws a line of slope `b` and intercept `a`
`abline(h=y)`	draws a horizontal line at ordinate `y`
`abline(v=x)`	draws a vertical line at abcissa `x`
`abline(lm.obj)`	draws the regression line given by `lm.obj` (see section 5)

Graphical parameters

In R if we want to improve the presentation of the graphics there are list of graphical parameters that are used for this purpose. These parameters can be used as an option or if you want to change the graphical presentation permanently this can be done with the help of this function par. Let illustrate this with the help of the example.

```
> par(bg="yellow")
```

Once this command is executed all the graphs that are plotted their background turns yellow. There are almost 73 graphical parameters in R but we have shown some of them in the table given below because these parameters are used on frequent bases.

`adj`	controls text justification with respect to the left border of the text so that `0` is left-justified, `0.5` is centred, `1` is right-justified, values > 1 move the text further to the left, and negative values further to the right; if two values are given (e.g., `c(0, 0)`) the second one controls vertical justification with respect to the text baseline
`bg`	specifies the colour of the background (e.g., `bg="red"`, `bg="blue"`; the list of the 657 available colours is displayed with `colors()`)
`bty`	controls the type of box drawn around the plot, allowed values are: `"o"`, `"l"`, `"7"`, `"c"`, `"u"` ou `"]"` (the box looks like the corresponding character); if `bty="n"` the box is not drawn
`cex`	a value controlling the size of texts and symbols with respect to the default; the following parameters have the same control for numbers on the axes, `cex.axis`, the axis labels, `cex.lab`, the title, `cex.main`, and the sub-title, `cex.sub`
`col`	controls the colour of symbols; as for `cex` there are: `col.axis`, `col.lab`, `col.main`, `col.sub`
`font`	an integer which controls the style of text (1: normal, 2: italics, 3: bold, 4: bold italics); as for `cex` there are: `font.axis`, `font.lab`, `font.main`, `font.sub`
`las`	an integer which controls the orientation of the axis labels (0: parallel to the axes, 1: horizontal, 2: perpendicular to the axes, 3: vertical)
`lty`	controls the type of lines, can be an integer (1: solid, 2: dashed, 3: dotted, 4: dotdash, 5: longdash, 6: twodash), or a string of up to eight characters (between `"0"` and `"9"`) which specifies alternatively the length, in points or pixels, of the drawn elements and the blanks, for example `lty="44"` will have the same effet than `lty=2`
`lwd`	a numeric which controls the width of lines
`mar`	a vector of 4 numeric values which control the space between the axes and the border of the graph of the form `c(bottom, left, top, right)`, the default values are `c(5.1, 4.1, 4.1, 2.1)`
`mfcol`	a vector of the form `c(nr,nc)` which partitions the graphic window as a matrix of `nr` lines and `nc` columns, the plots are then drawn in columns (see section 4.1.2)
`mfrow`	id. but the plots are then drawn in line (see section 4.1.2)
`pch`	controls the type of symbol, either an integer between 1 and 25, or any single character within `""` (Fig. 2)
`ps`	an integer which controls the size in points of texts and symbols

Programming with R in Practice

Now in both sections above we have discussed a lot about R functionalities now it's time to implement some basic R code. Now let's talk about language and programming.

Loops and Vectorization

Like all other programming language R also has a control structure which are not dissimilar to those of C programming language. Let's start with a simple piece of code in R, for example: we have a vector x and for each element of x with value b we want to assign the value 1 to another variable y otherwise 0. To write this problem code in R we first have to create the same length of vector y as of x.

```
y <- numeric(length(x))
for (i in 1:length(x)) if (x[i] == b) y[i] <- 0 else y[i] <- 1
```

Several instructions can be executed if they are placed within braces:

```
for (i in 1:length(x)) {
y[i] <- 0
...
}
if (x[i] == b) {
y[i] <- 0
...
}
```

Another possible situation is to execute an instruction as long as a condition is true:

```
while (myfun > minimum) {
...
}
```

In most of the cases we can avoid loops and control structures, thanks to R for this wonderful feature, Vectorization. The purpose of vectorization is to make loops implicit in expressions, and we have seen many cases of this. To have a better understanding of this let us consider the addition of two vectors.

```
> z <- x + y
```

This addition could be written with a loop, as this is done in most languages:

```
> z <- numeric(length(x))
> for (i in 1:length(z)) z[i] <- x[i] + y[i]
```

In This case because of the indexing system problem it is important to create vector z. Now we can see this explicit loop will work only if x and y vector are of same lengths.

```
> y[x == b] <- 0
> y[x != b] <- 1
```

In simple words vectored expression is computationally more efficient as compared to loops, particularly with a large set of data.

```
> x <- rnorm(10, -5, 0.1)
> y <- rnorm(10, 5, 2)
> X <- cbind(x, y) # the columns of X keep the names "x" and "y"
> apply(X, 2, mean)
x y
-4.975132 4.932979
> apply(X, 2, sd)
x y
0.0755153 2.1388071
lapply() acts on a list: its syntax is similar to apply and it returns a list.
> forms <- list(y ~ x, y ~ poly(x, 2))
> lapply(forms, lm)
[[1]]
Call:
FUN(formula = X[[1]])
Coefficients:
(Intercept) x
31.683 5.377
[[2]]
Call:
FUN(formula = X[[2]])
Coefficients:
(Intercept) poly(x, 2)1 poly(x, 2)2
4.9330 1.2181 -0.6037
```

Now the functions we have used, sapply () is a flexible variant of lapply(),

which can take a vector or matrix as the main argument.

Writing a program in R

Basically the R program in writing in a file, which is saved in ASCII format with the extension '.R'. It is a typical way in programming to use the several programs for typical situations. In our first program we will plot data of three different species of birds, so our data must be in three different files. In order to deal with this problem, we will proceed step by step and see different ways of handling this problem.

```
layout(matrix(1:3, 3, 1)) # partition the graphics
data <- read.table("Swal.dat") # read the data
plot(data$V1, data$V2, type="l")
title("swallow") # add a title
data <- read.table("Wren.dat")
plot(data$V1, data$V2, type="l")
title("wren")
data <- read.table("Dunn.dat")
plot(data$V1, data$V2, type="l")
title("dunnock")
```

The character that we use in above given code, # is used to add comments in a program and R moves to next line. Now the problem with the above code we have written is that it might get long if we want to add more species so we have to make this program generic.

```
layout(matrix(1:3, 3, 1)) # partition the graphics
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
for(i in 1:length(species)) {
data <- read.table(file[i]) # read the data
plot(data$V1, data$V2, type="l")
title(species[i]) # add a title
}
```

Now our program is more efficient, we can add more species since the vector containing the species list and file name is at the beginning of the code.

The above program will execute perfectly if data files .dat are located in R directory, if code does not work user might change the directory or specify

the path of the data files. If the program is written to the file Mybirds.R, it will be called by typing: > source("Mybirds.R")

Writing your own Function

In the above R codes, we have clearly seen that most of the R work is done with the functions, which arguments are given in parenthesis. In this topic we as user write our own function which will have the same exact properties like other functions in R. The basic aim of writing our own function is to make an efficient, rational and flexible use of R. Now let's quickly come back to example of reading data by plotting a graph.

```
myfun <- function(S, F)
{
data <- read.table(F)
plot(data$V1, data$V2, type="l")
title(S)
}
```

To execute this function, it must be loaded in memory and it can be done in several ways. The lines of function can be directly typed from the keyboard or copy and paste from the editor. If a function is saved in the file it can be loaded by using this command source () like another program. If user wants to load the program each time when it starts R, then the user can save the program in workspace. RData which will be loaded in memory if it is in the working directory. Another possibility is to configure the file '.Rprofile' or 'Rprofile'. Finally, we can create packages as well, but it cannot be discussed here.

```
layout(matrix(1:3, 3, 1))
myfun("swallow", "Swal.dat")
myfun("wren", "Wrenn.dat")
myfun("dunnock", "Dunn.dat")
We may also use sapply() leading to a fourth version of our program:
layout(matrix(1:3, 3, 1))
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
sapply(species, myfun, file)
```

In R it is not necessary to declare variables used within the functions, when the function will execute there is a rule in R called lexical scoping, R decides itself whether the object is local to the function or global. To have

a better understanding of this let us explain this with the help of the example given below:

```
> foo <- function() print(x)
> x <- 1
> foo()
[1] 1
> x <- 1
> foo2 <- function() { x <- 2; print(x) }
> foo2()
[1] 2
> x
[1] 1
```

Now let's do the numerical analysis of Ricker's model. The model can be explained with the help of the mathematical formula:

```
Nt+1 = Nt exp  r  1 − Nt K
ricker <- function(nzero, r, K=1, time=100, from=0, to=time)
{
N <- numeric(time+1)
N[1] <- nzero
for (i in 1:time) N[i+1] <- N[i]*exp(r*(1 - N[i]/K))
Time <- 0:time
plot(Time, N, type="l", xlim=c(from, to))
}
```

Why didn't my command(s) do what I expected?

This is the common problem that we face in every programming language. Basically, R will do what you said, not what you just meant. So here are some tips to remember if you really want the commands to work exactly as you want them to do so:

- Before executing any command in your code look for the examples of that command and follow them.
- If you have too many commands to write simply break down the commands in to smaller part and check each part is working propely or not.
- Experiment with the test data

- If you want to achieve better results, look for the on-line documentation of the command before using it.
- Look at your data maybe your data is not what you have thought
- Look at the structure of the code.

A common problem is variable definition in the workspace and the variable define in local space may have the same names. This can be illustrated with the help of R code:

```
> data(trees); str(trees)
`data.frame': 31 obs. of 3 variables:
$ Girth : num 8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 ...
$ Height: num 70 65 63 72 81 83 66 75 80 75 ...
$ Volume: num 10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 ...
> (Volume <- sample(1:31))
[1] 6 7 3 21 22 1 24 9 11 14 19 13 20 8 5 30 29 31
[19] 18 10 23 15 12 27 16 2 26 17 28 4 25
> attach(trees)
> cor(Volume, Girth)
[1] 0.30725
```

Now this is not the expected value.

It is very low, and we know that a tree's volume should be fairly well correlated with its girth. What is wrong? Listing the workspace gives a clue:

```
> ls()
[1] "Volume" "trees"
> cor(trees$Volume, Girth)
[1] 0.96712
> rm(Volume)
> cor(Volume, Girth)
[1] 0.96712
```

Opening the Black Box

Now is the time to get more into this R language. In this section we will test a variety of techniques.

The Dataset

In this dataset section we will use wine dataset form UCI repository. Our Goal is to predict best quality of wine of which there are seven values in total (integers 3-9). We will turn our dataset in binary classification task to predict whether the quality of wine is good or not, which is arbitrarily chosen as 6 or higher. There is one thing that we need to remember that there are very few 3s or 9s so we really have to work with 5 values only, but aside from quality of wine there it also contains predictors such as sugar, alcohol content, and acidity or other features of wine.

The original wine dataset from UCI repository is divided into white and red sets. We have combined these sets and add additional variables; color and its numeric version white indicating white or red, and good, indicating scores greater than or equal to 6 (denoted as 'Good').

```
wine = read.csv("http://www.nd.edu/~mclark19/learn/data/goodwine.csv")
summary(wine)
```

R Implementation

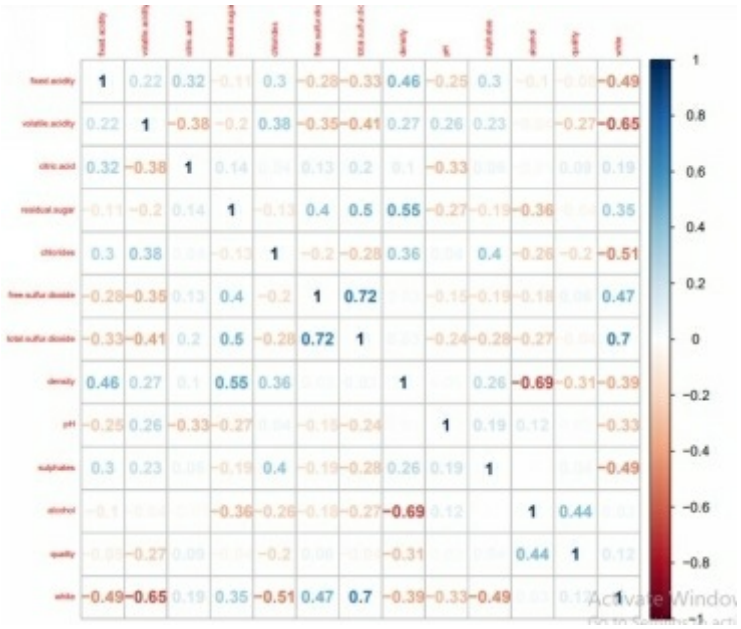
Before doing this, we will have to install and load the **caret** package in R. This package is very helpful in implementing of validation, data partitioning, performance assessment of the given dataset, and predictions about the data once it's trained. Once the package is installed it is a very genuine approach to use your computer resources as much possible as you can because some of these procedures takes more time, and more with the type of data you have. If you have more data it will take more time and it you have less data it will take less time. For example, if your computer is quad core that means your processor has four cores that independently acting as CPUs. You can set up R for parallel processing with the code given below, this will allow the newly installed package caret to assign task to three cores simultaneously.

```
library(doSNOW)
registerDoSNOW(makeCluster(3, type = "SOCK"))
```

Feature Selection & the Data Partition

To leave a holdout sample this data set is large enough, and this data set allows us to search for the best given modelling approach initially, over a grid of tuning parameters specific to the technique. In this we don't want the test performance to be contaminated with the tuning process to iterate previous discussion. With the best model at t tuning parameter(s), performance will be asses with the prediction on holdout set. To deal with notably collinearity in the data I have made some decisions, which can severely hamper some methods. To deal with let's have a look at the simple correlation matrix:

```
library(corrplot)
corrplot(cor(wine[, -c(13, 15)]), method = "number", tl.cex = 0.5)
```

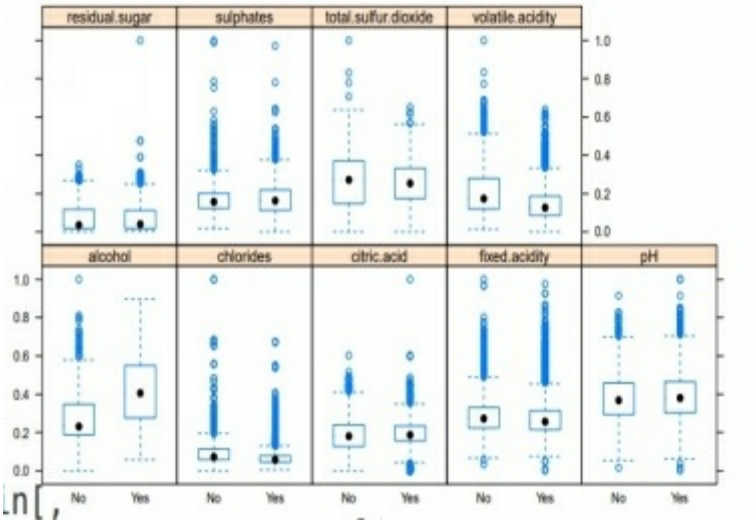


Regression analysis is running all over the data to examine the r-squared for each predictor in a model as they were dependent variable predicted by other input variable. Density was the highest at over 96%, and further investigation reveals that color and either sulfur dioxide is captured by other variables, well these will not be included in the following models.

The purpose of using **caret** package is that it has his own partitioning

function which we can use to separate our data into training and test data. The total number of the observations are 6497 out of them I will put 80% into the training set. The function **createDataPartition** will produce indices that will use as training set. Once this is done we will normalize the continuous variable to [0, 1]. Training of data set will be done as training process so that all the subsets under consideration are scaled properly, but for the set of tests set we will do it now.

```
library(caret)

set.seed(1234) #so that the indices will be the same when re-run trainIndices =
createDataPartition(wine$good, p = 0.8, list = F)

wine_train = wine[trainIndices, -c(6, 8, 12:14)] #remove quality and color, as well as density and
others

wine_test = wine[!1:nrow(wine) %in% trainIndices, -c(6, 8, 12:14)]

# prep_test = preProcess(wine_test[,-10], method='range') wine_test =

# data.frame(predict(prep_test, wine_test[,-10]), good=wine_test[ ,10])


wine_trainplot = predict(preProcess(wine_train[, -10], method = "range"), wine_train[, -10])


featurePlot(wine_trainplot, wine_train$good, "box")
```



K-nearest Neighbors

To choose something like Euclidean distance as a metric, then each point in the matrix says about the value of how far observation is from some other point, and on a set of variables respective values are given. This approach is commonly used because it will exploit this information for predictive purpose. To illustrate this, we will take a classification example, suppose k=5 neighbors. Let say we have a given observation x(i), now

based on predictive variables find 5 closet k neighbor in terms of Euclidean distance. To obtain the continuous outcome we might take, mean of those neighbors as a prediction.

In this algorithm **caret** package provide different methods for validation some of them are listed as k-fold, bootstrap, leave-one-out and others. For this we will use 10-fold cross validation.

```
#set.seed(1234)
cv_opts = trainControl(method="cv", number=10)
knn_opts = data.frame(.k=c(seq(3, 11, 2), 25, 51, 101)) #odd to avoid ties
results_knn = train(good~., data=wine_train, method="knn",
preProcess="range", trControl=cv_opts,
tuneGrid = knn_opts)
results_knn
```

Strengths

- Our approach is Intuitive
- Robust to outliers on the predictors

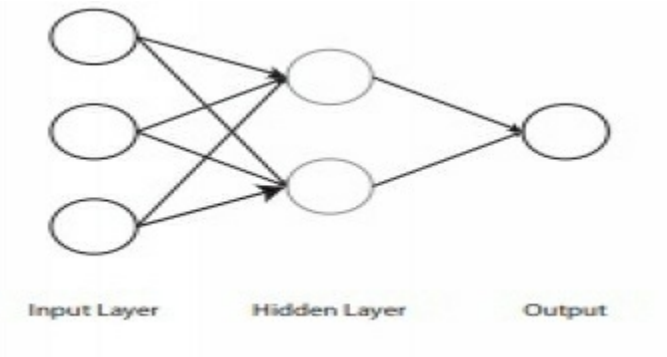
Weaknesses

- Susceptible to irrelevant features.
- Susceptible to correlated inputs.
- Ability to handle data of mixed types.
- Big data. Though approaches are available that help in this regard.

Neural Networks

Neural networks have been implemented for so long as a core concept of artificial intelligence and even as a machine learning algorithm and it works well in this case as well. In most of the case, these networks are considered as a nonlinear regression. Basically, visually we can seem them as a bunch of layers of inputs and outputs. The inputs that are created as weighted combination and pass through some method and this method will generate a new layer of input. The new layer is passed through another function to create one more layer or we can predict the output, which will be the final layer. All layers that are between input layer and output layer

are names as hidden layer. Now in the case neural networks if there is no hidden layer then this network is considered as a standard regression problem.



The main problem with neural networks is to determine how many hidden layers are there and how many hidden units in a layer. The neural network will face a variance problem if the network is very complex and less generalized, this will also happen if there is less relevant information in the training data. The executing of neural network might slow down your pc if your pc doesn't have multiple processors. You can replace the method with "nnet" and shorten the tuneLength to 3 which will be faster without much loss of accuracy.

```
results_nnet = train(good~., data=wine_train, method="avNNet", trControl=cv_opts,
preProcess="range", tuneLength=5, trace=F, maxit=1000)
results_nnet
```

With the help of this we can clearly see that our best model has 9 hidden layer nodes and decay parameter of 0. Now the first thought that came to your mind is how many hidden units you want to examine in terms of the amount of data you have, and in our case, we have a very decent amount. In our case we have to start with a broad range of the values for the number of inputs and then we have to narrow our focus, but we some of the weight decay, we must be able to avoid overfitting. The accuracy is increased up-to 1.5% by using up-to 50 hidden units.

```
preds_nnet = predict(results_nnet, wine_test[,-10])
confusionMatrix(preds_nnet, wine_test[,10], positive='Good')
```

Strengths

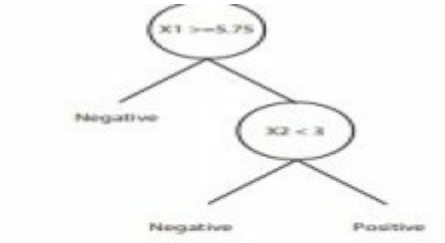
- Good prediction generally
- Incorporating the predictive power of different combinations of inputs
- Some tolerance to correlated inputs

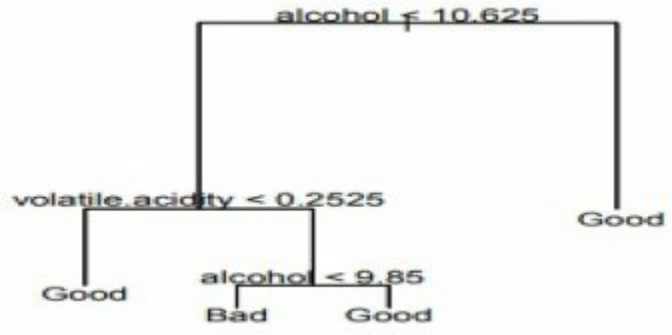
Weaknesses

- Susceptible to irrelevant features
- Not robust to outliers
- Big data with complex models

Trees and Forests

There is a different approach to prediction that are provided by classification and regression trees. Suppose we have two input variable one is single input and other is binary independent. We have done this because we have to find a point where if we partition the data at that point we will get a best classification accuracy and for that we have to search all the values of the input. For example, there is a single variable whose range might be from 1 to 10, we find that a cut at 5.75 is the best classification result if all the observation that we have are greater than, or equal to 5.75 are considered as best positive result and rest as negative. So basically, the purpose for explaining this is that this approach is easy to learn or grasp and that is the main reason tree approaches are appealing.





Now on a range from 1 to 10 add a second input. With the help of this input we might get a better classification results, if we are only looking upon the data regarding those greater and equal to 5.75 which we have already classified as positive result even if they are less than 3 on the second input variable.

The example tree given above is based on wine training data set. The Tree is interpreted as follows: wine is classified as good if the alcohol content is greater than 10.63%, if alcohol content is less than 10.63% but its volatile acidity is also less than .25 in that case wine will also be classified as good, and for the other observations, we have if the percentage is least more than 9.85% it is also classified as good, other than this all the remaining observations are classified as bad wine.

```
set.seed(1234)
rf_opts = data.frame(.mtry=c(2:6))
results_rf = train(good~., data=wine_train, method="rf",
preProcess='range',trControl=cv_opts, tuneGrid=rf_opts,
n.tree=1000)
results_rf
## 5199 samples
## 9 predictors
## 2 classes: 'Bad', 'Good'
##
## Pre-processing: re-scaling to [0, 1]
## Resampling: Cross-Validation (10 fold)
##
## Summary of sample sizes: 4679, 4679, 4680, 4679, 4679, 4679, ...
##
## Resampling results across tuning parameters:
##
## mtry Accuracy Kappa Accuracy SD Kappa SD
## 2 0.8 0.6 0.02 0.04
## 3 0.8 0.6 0.02 0.04
## 4 0.8 0.6 0.02 0.04
## 5 0.8 0.6 0.02 0.04
## 6 0.8 0.6 0.02 0.04
##
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

The initial results look promising with mtry = 2 producing the best initial result. Now for application to the test set.

```
preds_rf = predict(results_rf, wine_test[,-10])
confusionMatrix(preds_rf, wine_test[,10], positive='Good')
## Confusion Matrix and Statistics
##
## Reference
## Prediction Bad Good
## Bad 335 91
## Good 141 731
##
## Accuracy : 0.821
## 95% CI : (0.799, 0.842)
## No Information Rate : 0.633
## P-Value [Acc > NIR] : <2e-16
##
## Kappa : 0.606
## Mcnemar's Test P-Value : 0.0013
##
## Sensitivity : 0.889
## Specificity : 0.704
## Pos Pred Value : 0.838
## Neg Pred Value : 0.786
## Prevalence : 0.633
## Detection Rate : 0.563
## Detection Prevalence : 0.672
##
## 'Positive' Class : Good
##
library(rpart)
x <- cbind(x_train,y_train)
# grow tree
fit <- rpart(y_train ~ ., data = x,method="class")
summary(fit)
#Predict Output
```

```
predicted= predict(fit,x_test)
```

Our best result so far with the accuracy of 82.31%.

```
library(randomForest)
x <- cbind(x_train,y_train)
# Fitting model
fit <- randomForest(Species ~ ., x,ntree=500)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Support Vector Machines

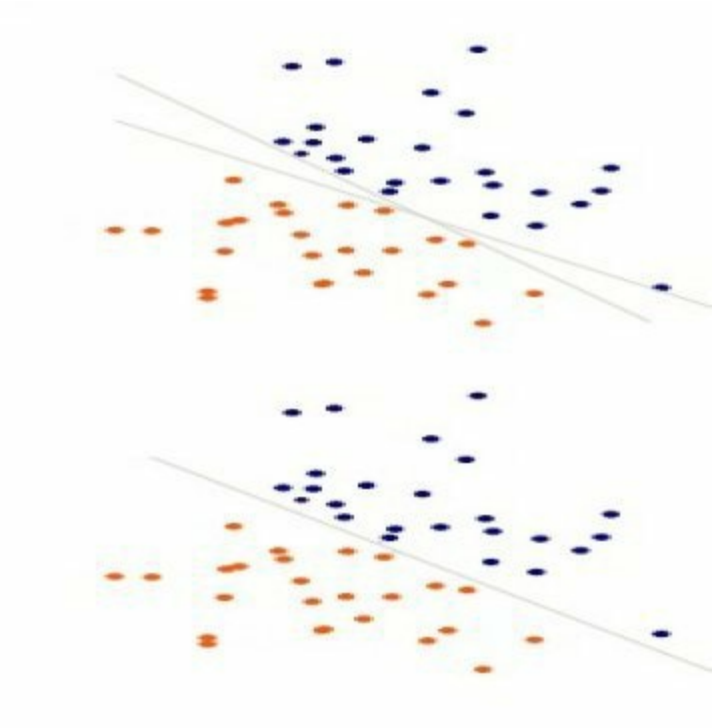
In this section discussion on Support vector machine will be our last example, because it is perhaps the least intuitive. Basically, the working of SVM is as follows, it seeks to map input space via kernel function to a higher dimension, meanwhile between this process it transformed feature space, find a hyperplane that will result in maximal separation of the data.

To have a better understanding of this procedure there is the figure given above which have two inputs x and y. No separation between classes is shown by cursory intersection. However, to find a clearer separation in the following figure we must have to map data to higher dimensions. But at this point there is a point to remember that there are a number of choices in regard to kernel function that performs the mapping, but in our case in higher dimension, a decision boundary is chosen in order to get maximum distance between the classes.

```
set.seed(1234)
results_svm = train(good~., data=wine_train, method="svmLinear",
preProcess="range", trControl=cv_opts, tuneLength=5)
results_svm
## 5199 samples
## 9 predictors
## 2 classes: 'Bad', 'Good'
##
```

```
## Pre-processing: re-scaling to [0, 1]
## Resampling: Cross-Validation (10 fold)
##
## Summary of sample sizes: 4679, 4679, 4680, 4679, 4679, 4679, ...
##
## Resampling results across tuning parameters:
##
## C Accuracy Kappa Accuracy SD Kappa SD
## 0.2 0.7 0.4 0.02 0.05
## 0.5 0.7 0.4 0.02 0.05
## 1 0.7 0.4 0.02 0.05
## 2 0.7 0.4 0.02 0.04
## 4 0.7 0.4 0.02 0.05
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.5.
class1
class2
class1
class2
preds_svm = predict(results_svm, wine_test[,-10])
confusionMatrix(preds_svm, wine_test[,10], positive='Good')
## Confusion Matrix and Statistics
##
## Reference
## Prediction Bad Good
## Bad 268 123
## Good 208 699
##
## Accuracy : 0.745
## 95% CI : (0.72, 0.769)
## No Information Rate : 0.633
## P-Value [Acc > NIR] : < 2e-16
##
## Kappa : 0.43
## Mcnemar's Test P-Value : 3.89e-06
##
```

```
## Sensitivity : 0.850
## Specificity : 0.563
## Pos Pred Value : 0.771
## Neg Pred Value : 0.685
## Prevalence : 0.633
## Detection Rate : 0.539
## Detection Prevalence : 0.699
##
## 'Positive' Class : Good
```



```
> set.seed(3233)
> svm_Radial <- train(V14 ~., data = training, method = "svmRadial",
  trControl=trctrl,
  preProcess = c("center", "scale"),
  tuneLength = 10)
> svm_Radial
```

Support Vector Machines with Radial Basis Function Kernel

210 samples

13 predictor

2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)

Resampling: Cross-Validated (10 fold, repeated 3 times)

Summary of sample sizes: 189, 189, 189, 189, 189, 189, ...

Resampling results across tuning parameters:

Tuning parameter 'sigma' was held constant at a value of 0.04744793

Accuracy was used to select the optimal model using the largest value.

The final values used for the model were sigma = 0.04744793 and C = 0.25.

```
> plot(svm_Radial)
> grid_radial <- expand.grid(sigma = c(0,0.01, 0.02, 0.025, 0.03, 0.04,
0.05, 0.06, 0.07,0.08, 0.09, 0.1, 0.25, 0.5, 0.75,0.9),
C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75,
1, 1.5, 2,5))> set.seed(3233)
> svm_Radial_Grid <- train(V14 ~., data = training, method = "svmRadial",
                  trControl=trctrl,
                  preProcess = c("center", "scale"),
                  tuneGrid = grid_radial,
                  tuneLength = 10)

> svm_Radial_Grid
```

Support Vector Machines with Radial Basis Function Kernel

210 samples

13 predictor

2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)

Resampling: Cross-Validated (10 fold, repeated 3 times)

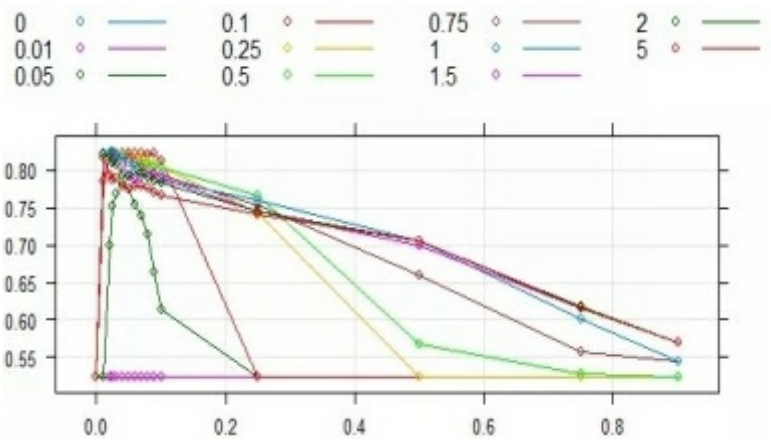
Summary of sample sizes: 189, 189, 189, 189, 189, 189, ...

Resampling results across tuning parameters:

Accuracy was used to select the optimal model using the largest value.

The final values used for the model were sigma = 0.025 and C = 1.

```
>plot(svm_Radial_Grid)
```



This is an example script of SVM perform on iris data set.

```
library(e1071)
x <- cbind(x_train,y_train)
# Fitting model
fit <-svm(y_train ~ ., data = x)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Tool you Already Have

Before getting started you should keep one thing is your mind that standard techniques are still available, but in this book, we might tweak them or do something different with them. So before getting started machine learning in R you have a better knowledge of statistics that is all required.

Standard Linear Model

The knowledge of the statistics is important because this will cover the main topic, linear regression in detail. Linear regression can be served as the starting point while learning machine learning. We can also describe it in the terms of matrix notations as follows:

$y = N(\mu, \sigma 2)$

$\mu = X\beta$

In above given equation y is referred as normally distributed vector responses with mean u and constant variance σ 2. X in this above equation is typically model matrix, it is a matrix of predictor variables and first column in this describes vector of ones for the intercept, while on the other hand beta is vector of coefficients corresponding to the intercept and predictors in the model. So, in applied courses less focus is given to the tool whether it will or not be the best tool for the job or even applicable in the form it is presented. Because of this problem many applied researchers are still hammering screws with it, even after the discovery of the statistical methods during the past quarter century, has rendered obsolete many current introductory statistical texts that are written for disciplines. So, the concepts that you once gain while learning the standard linear model are generalized and few modifications are made, but still maintain the basic design, can render it still very effective in situations where it is appropriate.

Logistic Regression

After the complete examination of the data, if the response is in categorical form we used logistic regression, most of the time with binary outcome in

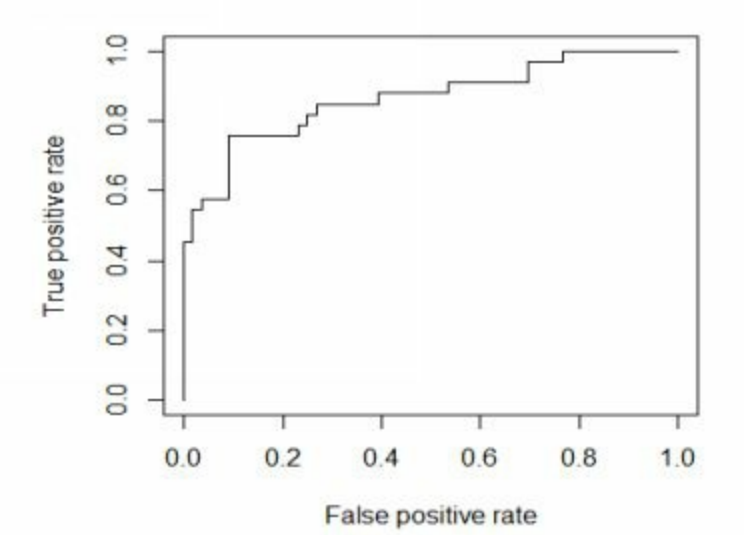
which some event occurs and some event or not. Even at this point once could use standard linear model, but you could end up with nonsensical predictions that fall outside the 0-1 range regarding the probability of the event occurring, to go along with other shortcomings. Using logistic regression over linear probability model no more effort is required. It is also important to keep logistics regression in mind when we discuss other classification techniques.

Model fitting

We split the data into two chunks: training and testing set. The training set will be used to fit our model which we will be testing over the testing set.

```
# Train test splitting
train <- data[1:800,]
test <- data[801:889,]

# Model fitting
model <- glm(Survived ~.,family=binomial(link='logit'),data=train)
summary(model)

# Analysis of deviance
anova(model,test="Chisq")

# McFadden R^2
library(pscl)
pR2(model)
# MEASURING THE PREDICTIVE ABILITY OF THE MODEL

# If prob > 0.5 then 1, else 0. Threshold can be set for better results
fitted.results <- predict(model,newdata=subset(test,select=c(2,3,4,5,6,7,8)),type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)

misClasificError <- mean(fitted.results != test$Survived)
print(paste('Accuracy',1-misClasificError))

# Confusion matrix
library(caret)
confusionMatrix(data=fitted.results, reference=test$Survived)
```

```
library(ROCR)
# ROC and AUC
p <- predict(model, newdata=subset(test,select=c(2,3,4,5,6,7,8)), type="response")
pr <- prediction(p, test$Survived)
# TPR = sensitivity, FPR=specificity
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)

auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```



Unsupervised Learning

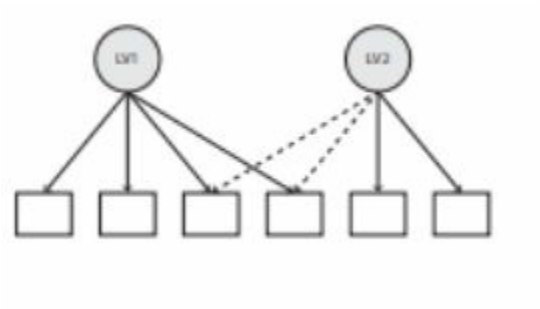
In this section we will discuss other machine learning techniques that will help us to have to better understanding of machine learning. By unsupervised learning, in machine learning we mean that we are dealing with the data that is unlabeled. In this type of scenario, we have our own typical sets of features in which we are interested in, we will not focus on mapping the data. In this type of technique, we are more interested in discovery of the structure within the data we have.

Clustering

Most of the methods that we use in unsupervised learning are commonly taught in different disciplines as various forms of the cluster analysis. The core concept of clustering is that we are dealing with unknown class structure rather than seeing how the different inputs relate to known class structure. The most commonly used clustering techniques are K-mean, hierarchical and model based.

Latent Variable Models

The aim of using these variables is to reduce the dimensionality of the inputs to make sets of information more manageable. The basic approach of using these variables is that we are thinking that much of our data can be seen, having only a few sources of variability, which are often called latent variables or factors. Now it will only take the familiar forms such as principal components. And factor analysis, but on the other hand it will also take independent component analysis and partial least squares techniques. There is an important point to be remembered that these can also be a part of supervised learning.

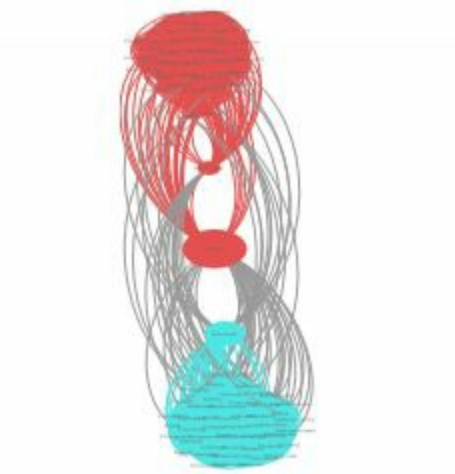


Imputation

This is the best approach which we can use when we are missing the data, which in term means that to impute the missing values. While many machine learning experts are familiar with this problem and know how to deal with it, so it may not be obvious that ML technique can also be used to deal with this problem. To have a better understanding of this concept, let us consider the simple example: Consider Netflix, Amazon and other ecommerce websites that suggest your different products based on products that you already liked or interested in. So, in this scenario the suggested products contain missing values for the user which are imputed based on their available data and other consumer similar to them who have rated the product in the question. Nowadays these recommender systems are widely used.

Graphical Structure

There are lots of other ways to understand structure among the observations and features. Among most of the approaches the famous one is a popular network analysis, in this type of network we get the similarities among different observations and after that we examine the structure of the data points visually, for this we placed the observations close together those which are similar in nature. In simple words we are not interested in the structure because we are focusing on modeling the relationships and making the predictions from the correlations of inputs.



Example graph of the social network of senators based on data and filter at the following link.

Textual Analysis

Most of the time when we deal with the machine learning problem, most of the times data is not in its typical forms, it is in textual content. In this situation we will have to do more work while preparing the data, by preparing data in this case we mean that there is rarely a text data before we start analyzing our data. The eventual goals may include using the word usage in the prediction of an outcome, perhaps modeling the use of select terms, or examining the structure of the term usage graphically as in a network model. Further, we can also apply machine learning methods to the sounds to discern the speech characteristics and other useful information from the sound.

Classification using R

The following example considers that we have a file Vehicles.txt that store.

```
# Set the working directory
setwd("C:/Data")


# Read a tab-delimited  data file
vehicle <- read.table(
   file = "Vehicles.txt",
   header = TRUE,
   sep = "\t",
   quote = "\"")
```

```
# Have a look at the first row of your file
head(vehicle)

# Loading the dplyr library
library(dplyr)

# Select a subset of columns
temp.subset <- select(
    .data = vehicle,
    Transmission,
    Cylinders,
    Fuel.Economy)

# Inspect the results
head(temp.subset)

# Filter a subset of rows
temp.subset <- filter(
    .data = temp.subset,
    Transmission == "Automatic")

# Inspecting the results
head(temp.subset)

# Compute a new column
temp.subset <- mutate(
    .data = temp.subset,
    Consumption = Fuel.Economy * 0.425)

# Inspect the results
head(temp.subset)

# Group by a column
temp.subset <- group_by(
    .data = temp.subset,
    Cylinders)
```

```
# Inspect the results
head(temp.subset)

# Aggregate based on groups
temp.subset <- summarize(
     .data = temp.subset,
    Avg.Consumption = mean(Consumption))

# Inspect the results
head(temp.subset)

# Arrange the rows in descending order
temp.subset <- arrange(
    .data = temp.subset,
    desc(Avg.Consumption))

# Inspect the results
head(temp.subset)

# Convert to data frame
efficiency <- as.data.frame(temp.subset)

# Inspect the results
print(efficiency)

# Chain methods together
efficiency <- vehicle %>%
    select(Fuel.Economy, Cylinders, Transmission) %>%
    filter(Transmission == "Automatic") %>%
    mutate(Consumption = Fuel.Economy * 0.425) %>%
    group_by(Cylinders) %>%
    summarize(Avg.Consumption = mean(Consumption)) %>%
    arrange(desc(Avg.Consumption)) %>%
    as.data.frame()

# Inspect the results
print(efficiency)
```

```
# Save the results to a CSV file
write.csv(
    x = efficiency,
    file = "Fuel Efficiency.csv",
    row.names = FALSE)
```

Frequently Asked Questions

Help! I got an error, what did I do wrong?

Whenever you any error while doing in code always read the error very carefully. Error often says exactly what is wrong with the piece of the code. For example, let see this code given below:

```
> x <- rnorm(100)
> summary(X)
Error in summary(X) : Object "X" not found
```

Now it is very clear from the error if you read it carefully that there is no object named X found in a workspace nor there is any object in data frames, so when R try to execute the code it will not find any object X. So, in this case the solution is very simple the variable X is upper case not lower-case x.

```
> summary(x)
Min. 1st Qu. Median Mean 3rd Qu. Max.
-2.750 -0.561 -0.070 -0.014 0.745 2.350
```

Now there are other error of directory if your command involves the external file other than from R directory, then you must have to include this file in R directory to give the full path name of the file in R command

```
> dlv <- read.csv("dlv.csv")
```

Error in file(file, "r") : unable to open connection

In addition: Warning message:

```
cannot open file 'dlv.csv'
```

The “unable to open connection” message means that the file could

```
not be found.
```

Check the current working directory with the getwd method, and see

if the file is there with the list.files method (or, you could look

for the file in Windows Explorer):

```
> getwd()
[1] "/Users/rossiter/ds/DavisGeostats"
> list.files(pattern="dlv.csv")
character(0)
```

So, the file is in another directory one way is to give the full path name of the file and another way is to copy the file in R Directory.

```
> dlv <- read.csv("/Users/rossiter/ds/DLV/dlv.csv")
> dim(dlv)
[1] 88 33
```

Another way is to change the working directory with the setwd method:

```
> setwd("/Users/rossiter/ds/DLV")
```

```
> getwd()
[1] "/Users/rossiter/ds/DLV"
> dlv <- read.csv("dlv.csv")
> dim(dlv)
[1] 88 33
```

Now another common error that we do in R, is to make an attempt to use a method that is in unloaded package.

For example, suppose we try to run a resistant regression with the

lqs method:

```
> lqs(lead ~ om, data=meuse)
Error: couldn't find function "lqs"
```

Now the error is because of two problems either there is no such method in R, or the method you are trying to use is in unloaded package, so to solve this problem we have a solution:

```
> help.search("lqs")
> library(MASS)
> lqs(lead ~ om, data=meuse)
```

```
Coefficients:
(Intercept) om
-19.9 15.4
Scale estimates 46.9 45.9
```

If there is any command that produces the error **compound** simply break down the commands into smaller pieces. If you still find an error about your command, go and check the full documentation of the command, in documentation they may have explains why an error is produced. For example, if we try to compute the non-parametric correlation between lead and organic matter in the Meuse data set, we get an error:

```
> library(gstat); data(meuse)
> cor(meuse$lead, meuse$om, method="spearman")
Error in cor(lead, om) : missing observations in cov/cor
> ?cor
```

We can check for these with the is.na method:

```
> sum(is.na(lead)); sum(is.na(om))
[1] 0
[1] 2
```

There are two missing values of organic matter (but none of lead).

Then we must decide how to deal with them; one way is to compute

the correlation without the missing cases:

```
> cor(lead, om, method="spearman", use="complete")
[1] 0.59759
```

Still if you find any error go to stack overflow type all the problem from beginning that you have with the piece of code you have, there you will find plenty of answer from the online community members. If not, there are so many Facebook pages and groups for developers, go post your problem and get the desired solution.

Conclusion

I hoped that this book shed some light on the topics that might be unfamiliar to some applied researchers. The field of machine learning has rapidly evolved over the past few decades with probability and statistics as its core part. In data analysis regression is an important problem, and this is not extensively dealt with Machine learning community. So, in this book we have discussed a system that is capable of learning regression rules. Regression models that are built by R have a benefit of achieving better prediction accuracy, when compared to other model approaches. In this book we have also discussed that our model compares to other machine learning algorithm, implemented in R outperforms them on some data sets. However, in most of the cases due to the large size of the data sets many differences are not statistically significant. We have also discussed how to improve the performance and efficiency of R code using Rccp package. In Future our aim is to explore new techniques that will enable our systems to deal with large scale domains. To deal with large scale domain methods like sampling and iterative regression modeling will help to overcome these problems.

Thank you !

Thank you for buying this book! It is intended to help you understanding machine learning using R. If you enjoyed this book and felt that it added value to your life, we ask that you please take the time to review it.

Your honest feedback would be greatly appreciated. It really does make a difference.

We are a very small publishing company and our survival depends on your reviews. Please, take a minute to write us your review.

Useful References

1. Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). Statistical Science, 16(3):199–231. Mathematical Reviews number (MathSciNet): MR1874152.

2. Domingos, P. (2012). A few useful things to know about machine learning. Commun. ACM, 55(10).

3. Harrell, F. E. (2001). Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis. Springer.

4. Hastie, T., Tibshirani, R., and Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer, 2nd ed. 2009. corr. 3rd printing 5th printing. Edition.

5. Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. The MIT Press. Wood, S. N. (2006). Generalized additive models: an introduction with R, volume 66. CRC Press.

6. Aha,D. Kibler,D. (1991): Instance-Based Learning Algorithms. In Machine Learning, vol. 6 - 1. Kluwer Academic Publishers.

7. Breiman, L., Friedman,J.H., Olshen,R.A. & Stone,C.J. (1984): Classification and Regression Trees, Wadsworth Int. Group, Belmont, California, USA.

8. Clark, P., Niblett, T. : Induction in noisy domains, in Proc. of the 2th European Working Session on Learning , Bratko,I. and Lavrac,N. (eds.), Sigma Press, Wilmslow, 1987.

9. Dillon,W., Goldstein,M. (1984) : Multivariate Analysis methods and applications. John Wiley & Sons.

10. Friedman,J. (1991): Mutivariate Adaptative Regression Splines. In Annals of Statistics , 19:1.

11. Friedman,J. Stuetzle,W. (1981): Projection Pursuit Regression. In J. American Statistics Association 76.

12. Karalic, A.. (1991): The Bayesian Approach to Tree-Structured Regression. In Proceedings of ITI-91 , Cavtat, Croatia, 1991.

13. Karalic, A..(1992): Employing Linear Regression in Regression Tree Leaves. In Proceedings of ECAI-92 , Wiley & Sons, 1992.

14. McClelland,J. Rumelhart,D. (1988): Explorations in Parallel Distributed Processing. Cambridge, Ma. : MIT Press.

15. Michalski, R.S. , Mozetic, I. , Hong, J., Lavrac, N. : The multi-purpose incremental learning system AQ15 and its testing application to three medical domains, in Proceedings of AAAI-86, 1986.

16. Michie,D., Spiegelhalter,D.J., Taylor,C. (1994) : Machine Learning, Neural and Statistical Classification. Ellis Horwood series in Artificial Intelligence. Ellis Horwood.

17. Quinlan, J.R. (1992): Learning with Continuos Classes. In Proceedings of the 5th Australian Joint Conference on Artificial Intelligence. Singapore: World Scientific, 1992.

18. Quinlan, J.R. (1993): Combining Instance-Based and Model-Based Learning. In Proceedings of 10th IML, Utgoff,P. (ed.). Morgan Kaufmann Publishers.

19. Rissanen,J. (1983) : A universal prior for integers and estimation by minimum description length. In Annals of Statistics 11, 2.

20. Torgo,L. (1993a) : Controlled Redundancy in Incremental Rule Learning. In Proceedings of ECML-93, Brazdil,P. (ed.). Lecture Notes in Artificial Intelligence - 667. Springer-Verlag.

21. Torgo,L. (1993b) : Rule Combination in Inductive Learning. In Proceedings of ECML-93, Brazdil,P. (ed.). Lecture Notes in Artificial Intelligence - 667. Springer-Verlag.

22. Torgo,L. (1995) : Applying Propositional Learning to Time Series Prediction. In ECML-95 workshop on Statistics, Machine Learning and Knowledge Discovery in Databases. Kodratoff, Y. et al. (eds.).

23. Urbancic,T., Bratko,I. (1994): Reconstructing Human Skill with Machine Learning. In Proceedings of the European Conference in Artificial Intelligence (ECAI-94), Cohn, A.G. (ed.). John Wiley & Sons.

24. Weiss,S.M., Indurkhya,N. (1993): Rule-Based Regression. In Proceedings of IJCAI-93, Bajesy,R. (ed.). Morgan Kaufmann Publishers.

25. Altman RB, Bergman CM, Blake J, et al. Text mining for biology–the way forward: opinions from leading scientists. Genome Biology. 2008;9(Suppl 2):S7. [PMC free article] [PubMed]

26. Aphinyanaphongs Y, Aliferis C. Text categorization models for identifying unproven cancer treatments on the web. Stud Health Technol Inform. 2007;129:968–72. [PubMed]

27. Balk EM, Moorthy D, Obadan NO, et al. Diagnosis and Treatment of Obstructive Sleep Apnea in Adults. Comparative Effectiveness Review No. 32. Rockville, MD: Agency for Healthcare Research and Quality; 2011. (Prepared by Tufts Evidence-based Practice Center under Contract No. 290-2007-100551). AHRQ Publication No. 11-EHC052-EF. www.effectivehealthcare.ahrq.gov/reports/final.cfm.

28. Cohen AM, Ambert K, McDonagh M. Cross-topic learning for work prioritization in

systematic review creation and update. J Am Med Inform Assoc. 2009;16(5):690–704. [PMC free article] [PubMed]

29. Cohen AM, Ambert K, McDonagh M. A Prospective Evaluation of an Automated Classification System to Support Evidence-based Medicine and Systematic Review. AMIA Annual Symposium proceedings. 2010:121–5. [PMC free article] [PubMed]

30. Cohen AM, Hersh WR, Peterson K, et al. Reducing workload in systematic review preparation using automated citation classification. J Am Med Inform Assoc. 2006;13(2):206–19. [PMC free article] [PubMed]

31. Dalal SR, Shekelle PG, Hempel S, et al. A Pilot Study Using Machine Learning and Domain Knowledge To Facilitate Comparative Effectiveness Review Updating, Methods Research Report. Rockville, MD: Agency for Healthcare Research and Quality; Sep, 2012. Prepared by the Southern California Evidence-based Practice Center under Contract No. 290-2007-10062-I). AHRQ Publication No. 12-EHC069-EF.

32. Danz M, Rubenstein L, Hempel S, et al. Identifying quality improvement intervention evaluations –is consensus achievable? Qual Saf Health Care. 2010;19(4):279–83. [PubMed]

33. Deerwester S, Dumais S, Furnas G, et al. Indexing by Latent Semantic Analysis. J Am Soc Inf Sci. 1990;41(6):391–407.

34. EPOC: Cochrane Effective Practice and Organisation of Care Group. Welcome to the EPOC Ottawa Website. [Accessed April 2012]. http://epoc.cochrane.org.

35. Friedman J, Hastie T, Tibshirani R. Regularization paths for generalized linear models via coordinate descent. J Stat Softw. 2010;33(1):1–22. [PMC free article] [PubMed]

36. Genkin A, Lewis DD, Madigan D. Large-scale Bayesian logistic regression for text categorization. Techometric. 2007;49(3):201–304.

37. Glanville JM, Lefebvre C, Miles JN, et al. How to identify randomized controlled trials in MEDLINE: ten years on. J Med Libr Assoc. 2006;94:130–6. [PMC free article] [PubMed]

38. Hastie T, Tibshirani R, Friedman J. New York: Springer Verlag; The elements of statistical learning: data mining, inference, and prediction. 2009

39. Hempel S, Rubenstein L, Shanman R, et al. Identifying quality improvement intervention publications. Implement Sci. 2011;6:85. [PMC free article] [PubMed]

40. Jenkins M. Evaluation of methodological search filters—a review. Health Info Libr J. 2004;21:148–63. [PubMed]

41. O'Neil S, Rubenstein L, Danz M, et al. Identifying continuous quality improvement publications: what makes an improvement intervention 'CQI'? BMJ Qual Saf. 2011 Dec;20(12):1011–9. Epub 2011 Jul 4. [PMC free article] [PubMed]

42. Porter M. An algorithm for suffix stripping. Program. 1980;14(3):130–7.

43. Rubenstein LV, Hempel S, Farmer M, et al. Finding order in heterogeneity: Types of quality improvement intervention publications. Qual Saf Health Care. 2008;17(6):403–8. [PubMed]

44. Shetty KD, Dalal SR. Using information mining of the medical literature to improve drug safety. J Am Med Inform Assoc. 2011;18:668–74. [PMC free article] [PubMed]

45. Tibshirani R. Regression shrinkage and selection by Lasso. (Series B).J R Stat Soc Ser C Appl Stat. 1996;58:267–88.

46. Vapnik VN. The Nature of statistical learning theory. New York: Springer-Verlag; 1995.

47. Wallace BC, Small K, Brodley C, et al. Active learning for biomedical citation screening. KDD '2010. Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining; Washington, DC: ACM; 2010.

48. Wallace BC, Trikalinos TA, Lau J, et al. Semi-automated screening of biomedical citations for systematic reviews. BMC Bioinformatics. 2010;11:55. [PMC free article] [PubMed]

49. Wei Y, Clyne M, Dolan SM, et al. GAPscreener: an automatic tool for screening human genetic association literature in PubMed using the support vector machine technique. BMC Bioinformatics. 2008;9:205

50. R. Agrawal, T. Imielinski, and A. Swami (1993) Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207--216, Washington D.C.

51. Christian Borgelt and Rudolf Kruse (2002) Induction of Association Rules: Apriori Implementation. *15th Conference on Computational Statistics* (COMPSTAT 2002, Berlin, Germany) Physica Verlag, Heidelberg, Germany.

52. Christian Borgelt (2003) Efficient Implementations of Apriori and Eclat. *Workshop of Frequent Item Set Mining Implementations* (FIMI 2003, Melbourne, FL, USA).

53. Knuth, D. E. 1992. Literate programming. CSLI lecture notes 27. Stanford, CA: Center for the Study of Language and Information 101

54. Kopka, H. & Daly, P. W. 2004. Guide to LATEX. Boston: Addison-Wesley, 4th edition 101

55. Lamport, L. 1994. LATEX: a document preparation system : user's guide and reference manual. Reading, MA: Addison-Wesley, 2nd edition 101

56. Leisch, F. 2002. Sweave, part I: Mixing R and LATEX. R News 2(3):28–31 URL http://CRAN.R-project.org/doc/Rnews/ 101, 104

57. Leisch, F. 2006. Sweave User's Manual. Vienna (A): TU Wein, 2.7.1 edition URL http://www.stat.uni-muenchen.de/~leisch/Sweave/ 101, 104

58. Metcalfe, A. V. & Cowpertwait, P. S. 2009. Introductory Time Series with R. Use R! Springer. DOI: 10.1007/978-0-387-88698-5 106

59. Mevik, B.-H. 2006. The pls package. R News 6(3):12–17 62

60. Murrell, P. 2005. R Graphics. Boca Raton, FL: Chapman & Hall/CRC. ISBN 1-584-88486-X 69 Nason, G. P. 2008. Wavelet methods in statistics with R. Use R! New York ; London: Springer Paradis, E. 2002. R for Beginners. Montpellier (F): University of Montpellier URL http://cran.r-project.org/doc/contrib/rdebuts_en.pdf 106

61. Paradis, E. 2002. R para Principiantes. Montpellier (F): University of Montpellier URL http://cran.r-project.org/doc/contrib/rdebuts_es.pdf 106

62. Paradis, E. 2002. R pour les débutants. Montpellier (F): University of Montpellier

63. 2012. Technical Note: Literate Data Analysis. International Institute for Geo-information Science & Earth Observation (ITC), 1.3 edition, 29 pp. URL http://www.itc.nl/personal/rossiter/teach/R/LDA.pdf 101, 104

64. Rossiter, D. G. 2012. Tutorial: Using the R Environment for Statistical Computing: An example with the Mercer & Hall wheat yield dataset. Enschede (NL): International Institute for Geo-information Science & Earth Observation (ITC), 2.51 edition URL http://www.itc.nl/personal/rossiter/teach/R/R_mhw. pdf 1, 107

65. Rossiter, D. G. & Loza, A. 2012. Technical Note: Analyzing land cover change with R. Enschede (NL): International Institute for Geo-information Science & Earth Observation (ITC), 2.32 edition, 67 pp. URL http://www.itc.nl/personal/rossiter/teach/R/R_LCC. pdf 107

66. Sarkar, D. 2002. Lattice. R News 2(2):19–23 URL http://CRAN.R-project.org/doc/Rnews/ 17, 77

67. Sarkar, D. 2008. Lattice : multivariate data visualization with R. Use R! New York: Springer URL http://lmdvr.r-forge.r-project.org/ 106

68. Shumway, R. H. & Stoffer, D. S. 2006. Time Series Analysis and Its Applications, with R examples. Springer Texts in Statistics. Springer, 2nd edition URL http://www.stat.pitt.edu/stoffer/tsa2/index.html 107

69. Skidmore, A. K. 1999. Accuracy assessment of spatial information. In Stein, A.; Meer, F. v. d.; & Gorte, B. G. F. (eds.), Spatial statistics for remote sensing, pp. 197–209. Dordrecht: Kluwer Academic 25, 92