**Q1. What is R?**

R is a programming language and software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. R was designed 20 years ago to allow academic statisticians and others with sophisticated programming skills to perform complex data [statistical analysis](http://whatis.techtarget.com/definition/statistical-analysis) and display the results in any of a multitude of visual graphics.

Q2 Eligibility to do R?

No requirements are needed to learn R. If a person has some programming experience then he/she would be able to pick up the R easily but that is not mandatory. R is a functional paradigm language.

Q3 Where to use?

**R is used in following types of applications.**

Predictive modelling

Predictive modelling aims to generate the most accurate estimates of some quantity or event. It uses concepts of artificial intelligence, statistics and probability for estimation. More accurately it is a process of creating, testing and validating a model to best predict the probability of an outcome.

R has lots of packages inbuilt to perform these task.

Time series analysis

There are lot of time series algorithms implemented in R which perfrom time based prediction. Algorithms are Single exponential smoothing, Double exponential smoothing and Triple exponential smoothing. Example is they help us to estimate the impact of weather, season in forecasting or evaluating a factor like sales or consumption.

Regression analysis

They predicts continuous variables based on other variables in the dataset. Algorithms are Linear Regression, Exponential Regression, Geometric Regression, Logarithmic Regression etc. Practical example is they help us to estimate the factors which increase or decrease the sales and by how much.

Clustering :

It clusters the similar observations or data into groups. For example it will find out what type of items are frequently bought togather.

Decision tree Analysis

They classify and predict the discrete variables based on other variables in the dataset. For example it will help a bank to classify a new customer as good payer or bad payer.

Other programming languages like C, C++, Java, .Net or Python can be used to manipulate R objects directly.

R has stronger object oriented programming facilities than most statistical computing language.

**R and SAS**

R is a freeware and is contributed by thousands of developers across the globle. R can pretty much do everything SAS can do in terms of Statistical analysis and some pretty cool things SAS cant do like buildig a predictive model, ARIMA, decision trees etc. It is one of the best Statistical analysis tool in the world. The latest cutting edge techniques are released in R first. To date R has almost got 15,000 packages in the CRAN.

SAS is a high level programming but R is true programming language hence gives more flexibility and power than SAS to the programmer.

This is the html version of the file <https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec02.pdf>.  
**Google** automatically generates html versions of documents as we crawl the web.

|  |
| --- |
| **Page 1** |

R for Machine Learning

Allison Chang

**1 Introduction**

It is common for today’s scientific and business industries to collect large amounts of data, and the ability to

analyze the data and learn from it is critical to making informed decisions. Familiarity with software such as R

allows users to visualize data, run statistical tests, and apply machine learning algorithms. Even if you already

know other software, there are still good reasons to learn R:

1. **R is free.** If your future employer does not already have R installed, you can always download it for free,

unlike other proprietary software packages that require expensive licenses. No matter where you travel, you

can have access to R on your computer.

2. **R gives you access to cutting-edge technology.** Top researchers develop statistical learning methods

in R, and new algorithms are constantly added to the list of packages you can download.

3. **R is a useful skill.** Employers that value analytics recognize R as useful and important. If for no other

reason, learning R is worthwhile to help boost your résumé.

Note that R is a programming language, and there is no intuitive graphical user interface with buttons you can

click to run different methods. However, with some practice, this kind of environment makes it easy to quickly

code scripts and functions for various statistical purposes. To get the most out of this tutorial, follow the examples

by typing them out in R on your own computer. A line that begins with > is input at the command prompt. We

do not include the output in most cases, but you should try out the commands yourself and see what happens.

If you type something at the command line and decide not to execute, press the down arrow to clear the line;

pressing the up arrow gives you the previous executed command.

**1.1 Getting Started**

The R Project website is <http://www.r-project.org/>. In the menu on the left, click on CRAN under “Download,

Packages.” Choose a location close to you. At MIT, you can go with University of Toronto under Canada. This

leads you to instructions on how to download R for Linux, Mac, or Windows.

Once you open R, to figure out your current directory, type getwd(). To change directory, use setwd (note

that the “C:” notation is for Windows and would be different on a Mac):

> setwd("C:\\Datasets")

**1.2 Installing and loading packages**

Functions in R are grouped into packages, a number of which are automatically loaded when you start R. These

include “base,” “utils,” “graphics,” and “stats.” Many of the most essential and frequently used functions come

in these packages. However, you may need to download additional packages to obtain other useful functions. For

example, an important classification method called Support Vector Machines is contained in a package called

1

|  |
| --- |
| **Page 2** |

“e1071.” To install this package, click “Packages” in the top menu, then “Install package(s)...” When asked to

select a CRAN mirror, choose a location close to you, such as “Canada (ON).” Finally select “e1071.” To load

the package, type library(e1071) at the command prompt. **Note that you need to install a package only**

**once, but that if you want to use it, you need to load it each time you start R.**

**1.3 Running code**

You could use R by simply typing everything at the command prompt, but this does not easily allow you to save,

repeat, or share your code. Instead, go to “File” in the top menu and click on “New script.” This opens up a new

window that you can save as a .R file. To execute the code you type into this window, highlight the lines you

wish to run, and press Ctrl-R on a PC or Command-Enter on a Mac. If you want to run an entire script, make

sure the script window is on top of all others, go to “Edit,” and click “Run all.” Any lines that are run appear in

red at the command prompt.

**1.4 Help in R**

The functions in R are generally well-documented. To find documentation for a particular function, type ?

followed directly by the function name at the command prompt. For example, if you need help on the “sum”

function, type ?sum. The help window that pops up typically contains details on both the input and output for

the function of interest. **If you are getting errors or unexpected output, it is likely that your input is**

**insufficient or invalid, so use the documentation to figure out the proper way to call the function.**

If you want to run a certain algorithm but do not know the name of the function in R, doing a Google search

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

**2 Datasets**

When you test any machine learning algorithm, you should use a variety of datasets. R conveniently comes with

its own datasets, and you can view a list of their names by typing data() at the command prompt. For instance,

you may see a dataset called “cars.” Load the data by typing data(cars), and view the data by typing cars.

Another useful source of available data is the UCI Machine Learning Repository, which contains a couple

hundred datasets, mostly from a variety of real applications in science and business. The repository is located at

<http://archive.ics.uci.edu/ml/datasets.html>. These data are often used by machine learning researchers

to develop and compare algorithms. We have downloaded a number of datasets for your use, and you can find

the text files

[.](http://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/datasets)

These include:

**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

2

in the [Datasets](http://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/datasets) section

|  |
| --- |
| **Page 3** |

You are encouraged to download your own datasets from the UCI site or other sources, and to use R to study the

data. Note that for all except the Housing and Mushroom datasets, there is an additional class attribute column

that is not included in the column counts. Also note that if you download the Mushroom dataset from the UCI

site, it has 22 categorical features; in our version, these have been transformed into 119 binary features.

**3 Basic Functions**

In this section, we cover how to create data tables, and analyze and plot data. We demonstrate by example how

to use various functions. To see the value(s) of any variable, vector, or matrix at any time, simply enter its name

in the command line; you are encouraged to do this often until you feel comfortable with how each data structure

is being stored. To see all the objects in your workspace, type ls(). Also note that the arrow operator <- sets

the left-hand side equal to the right-hand side, and that a comment begins with #.

**3.1 Creating data**

To create a variable x and set it equal to 1, type x <- 1. Now suppose we want to generate the vector [1, 2, 3, 4, 5],

and call the vector v. There are a couple different ways to accomplish this:

> 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)

These can be row vectors or column vectors. To generate a vector v0 of six zeros, use either of the following.

Clearly the second choice is better if you are generating a long vector.

> v0 <- c(0,0,0,0,0,0)

> v0 <- seq(from=0,to=0,length.out=6)

We can combine vectors into matrices using cbind and rbind. For instance, if v1, v2, v3, and v4 are vectors of

the same length, we can combine them into matrices, using them either as columns or as 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)

Another way to create the second matrix is to use the matrix function to reshape a vector into a matrix of the

right dimensions.

> v <- seq(from=1,to=20,by=1)

> matrix(v, nrow=4, ncol=5)

Notice that this is not exactly right—we need to specify that we want to fill in the matrix by row.

> matrix(v, nrow=4, ncol=5, byrow=TRUE)

It is often helpful to name the columns and rows of a matrix using colnames and rownames. In the following,

first we save the matrix as matrix20, and then we name the columns and rows.

3

|  |
| --- |
| **Page 4** |

> matrix20 <- matrix(v, nrow=4, ncol=5, byrow=TRUE)

> colnames(matrix20) <- c("Col1","Col2","Col3","Col4","Col5")

> rownames(matrix20) <- c("Row1","Row2","Row3","Row4")

You can type colnames(matrix20)/rownames(matrix20) at any point to see the column/row names for matrix20.

To access a particular element in a vector or matrix, index it by number or by name with square braces:

> 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]

# element in third row and first column of matrix20

You can find the length of a vector or number of rows or columns in a matrix using length, nrow, and ncol.

> length(v1)

> nrow(matrix20)

> ncol(matrix20)

Since you will be working with external datasets, you will need functions to read in data tables from text files.

For instance, suppose you wanted 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 is the location (full path) of the file. If the first row of data contains column names, then the

second argument should be header = TRUE, and otherwise it is header = FALSE. The third argument contains

the delimiter. If the data are separated by spaces or tabs, then the argument is sep = " " and sep = "\t"

respectively. The default delimiter (if you do not include this argument at all) is “white space” (one or more

spaces, tabs, etc.). Alternatively, you can use setwd to change directory and use only the file name in the

read.table function. If the delimiter is a comma, you can also use read.csv and leave off the sep argument:

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

Use write.table to write a table to a file. Type ?write.table to see details about this function. If you need to

write text to a file, use the cat function.

A note about matrices versus data frames: A data frame is similar to a matrix, except it can also include

non-numeric attributes. For example, there may be a column of characters. Some functions require the data

passed in to be in the form of a data frame, which would be stated in the documentation. You can always coerce

a matrix into a data frame using as.data.frame. See Section 3.6 for an example.

A note about factors: A factor is essentially a vector of categorical variables, encoded using integers. For

instance, if each example in our dataset has a binary class attribute, say 0 or 1, then that attribute can be

represented as a factor. Certain functions require one of their arguments to be a factor. Use as.factor to encode

a vector as a factor. See Sections 4.5 and 4.9 for examples.

**3.2 Sampling from probability distributions**

There are a number of functions for sampling from probability distributions. For example, the following commands

generate random vectors of the user-specified length n from distributions (normal, exponential, poisson, uniform,

binomial) with user-specified parameters. There are other distributions as well.

4

|  |
| --- |
| **Page 5** |

> 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)

Suppose you have a vector v of numbers. To randomly sample, say, 25 of the numbers, use the sample function:

> sample(v, size=25, replace=FALSE)

If you want to sample with replacement, set the replace argument to TRUE.

If you want to generate the same random vector each time you call one of the random functions listed above,

pick a “seed” for the random number generator using set.seed, for example set.seed(100).

**3.3 Analyzing data**

To compute the mean, variance, standard deviation, minimum, maximum, and sum of a set of numbers, use mean,

var, sd, min, max, and sum. There are also rowSum and colSum to find the row and column sums for a matrix.

To find the component-wise absolute value and square root of a set of numbers, use abs and sqrt. Correlation

and covariance for two vectors are computed with cor and cov respectively.

Like other programming languages, you can write if statements, and for and while loops. For instance, here

is a simple loop that prints out even numbers between 1 and 10 (%% is the modulo operation):

> for (i in 1:10){

+

if (i %% 2 == 0){

+

cat(paste(i, "is even.\n", sep=" "))

# use paste to concatenate strings

+

}

+ }

The 1:10 part of the for loop can be specified as a vector. For instance, if you wanted to loop over indices 1, 2,

3, 5, 6, and 7, you could type for (i in c(1:3,5:7)).

To pick out the indices of elements in a vector that satisfy a certain property, use which, for example:

> which(v >= 0)

# indices of nonnegative elements of v

> v[which(v >= 0)]

# nonnegative elements of v

**3.4 Plotting data**

We use the Haberman’s Survival data (read into data frame dataset) to demonstrate plotting functions. Each

row of data represents a patient who had surgery for breast cancer. The three features are: the age of the patient

at the time of surgery, the year of the surgery, and the number of positive axillary nodes detected. Here we plot:

1. Scatterplot of the first and third features,

2. Histogram of the second feature,

3. Boxplot of the first feature.

To put all three plots in a 1 × 3 matrix, use par(mfrow=c(1,3)). To put each plot in its own window, use

win.graph() to create new windows.

> 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")

5

|  |
| --- |
| **Page 6** |

**Scatterplot**

**Histogram**

**Boxplot**

0 1

0 2

0 3

0 4

0 5

0

Number of Nodes

Count

0 1

02

03

04

05

06

0

30

40

50

60

70

80

30

40

50

60

70

80

58

60

62

64

66

68

Age

Year

Age

Figure 1: Plotting examples.

The pch argument in the plot function can be varied to change the marker. Use points and lines to add extra

points and lines to an existing plot. You can save the plots in a number of different formats; make sure the plot

window is on top, and go to “File” then “Save as.”

**3.5 Formulas**

Certain functions have a “formula” as one of their arguments. Usually this is a way to express the form of a

model. Here is a simple example. Suppose you have a response variable y and independent variables x1, x2, and

x3. To express that y depends linearly on x1, x2, and x3, you would use the formula y ∼ x1 + x2 + x3, where

y, x1, x2, and x3 are also column names in your data matrix. See Section 3.6 for an example. Type ?formula

for details on how to capture nonlinear models.

**3.6 Linear regression**

One of the most common modeling approaches in statistical learning is linear regression. In R, use the lm function

to generate these models. The general form of a linear regression model is

Y = β0 + β1X1 + β2X2 + ··· + βkXk + ε,

where ε is normally distributed with mean 0 and some variance σ

2

.

Let y be a vector of dependent variables, and x1 and x2 be vectors of independent variables. We want to find

the coefficients of the linear regression model Y = β0 + β1X1 + β2X2 + ε. The following commands generate the

linear regression model and give a summary of it.

> lm\_model <- lm(y ∼ x1 + x2, data=as.data.frame(cbind(y,x1,x2)))

> summary(lm\_model)

The vector of coefficients for the model is contained in lm model$coefficients.

**4 Machine Learning Algorithms**

We give the functions corresponding to the algorithms covered in class. **Look over the documentation for**

**each function on your own as only the most basic details are given in this tutorial.**

6

|  |
| --- |
| **Page 7** |

**4.1 Prediction**

For most of the following algorithms (as well as linear regression), we would in practice first generate the model

using training data, and then predict values for test data. To make predictions, we use the predict function. To

see documentation, go to the help page for the algorithm, and then scroll through the Contents menu on the left

side to find the corresponding predict function. Or simply type ?predict.name, where name is the function

corresponding to the algorithm. Typically, the first argument is the variable in which you saved the model, and

the second argument is a matrix or data frame of test data. Note that when you call the function, you can just

type predict instead of predict.name. For instance, if we were to predict for the linear regression model above,

and x1 test and x2 test are vectors containing test data, we can use the command

> predicted\_values <- predict(lm\_model, newdata=as.data.frame(cbind(x1\_test, x2\_test)))

**4.2 Apriori**

To run the Apriori algorithm, first install the arules package and load it. See Section 1.2 for installing and

loading new packages. Here is an example of how to run the Apriori algorithm using the Mushroom dataset.

Note that the dataset must be a binary incidence matrix; the column names should correspond to the “items”

that make up the “transactions.” The following commands print out a summary of the results and a list of the

generated rules.

> 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)

You can modify the parameter settings depending on your desired support and confidence thresholds.

**4.3 Logistic Regression**

You do not need to install an extra package for logistic regression. Using the same notation as in Section 3.6, the

command is:

> glm\_mod <-glm(y ∼ x1+x2, family=binomial(link="logit"), data=as.data.frame(cbind(y,x1,x2)))

**4.4** K**-Means Clustering**

You do not need an extra package. If X is the data matrix and m is the number of clusters, then the command is:

> kmeans\_model <- kmeans(x=X, centers=m)

**4.5** k**-Nearest Neighbor Classification**

Install and load the class package. Let X train and X test be matrices of the training and test data respectively,

and labels be a binary vector of class attributes for the training examples. For k equal to K, the command is:

> knn\_model <- knn(train=X\_train, test=X\_test, cl=as.factor(labels), k=K)

Then knn model is a factor vector of class attributes for the test set.

7

|  |
| --- |
| **Page 8** |

**4.6 Na¨ıve Bayes**

Install and load the e1071 package. Using the same notation as in Section 3.6, the command is:

> nB\_model <- naiveBayes(y ∼ x1 + x2, data=as.data.frame(cbind(y,x1,x2)))

**4.7 Decision Trees (CART)**

CART is implemented in the rpart package. Again using the formula, the command is

> cart\_model <- rpart(y ∼ x1 + x2, data=as.data.frame(cbind(y,x1,x2)), method="class")

You can use plot.rpart and text.rpart to plot the decision tree.

**4.8 AdaBoost**

There are a number of different boosting functions in R. We show here one implementation that uses decision

trees as base classifiers. Thus the rpart package should be loaded. Also, the boosting function ada is in the ada

package. Let X be the matrix of features, and labels be a vector of 0-1 class labels. The command is

> boost\_model <- ada(x=X, y=labels)

**4.9 Support Vector Machines (SVM)**

The SVM algorithm is in the e1071 package. Let X be the matrix of features, and labels be a vector of 0-1 class

labels. Let the regularization parameter be C. Use the following commands to generate the SVM model and view

details:

> svm\_model <- svm(x=X, y=as.factor(labels), kernel ="radial", cost=C)

> summary(svm\_model)

There are a variety of parameters such as the kernel type and value of the C parameter, so check the documentation

on how to alter them.

8

|  |
| --- |
| **Page 9** |

MIT OpenCourseWare

[http://ocw.mit.edu](http://ocw.mit.edu/)

15.097 Prediction: Machine Learning and Statistics

Spring 2012

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.