DIT UNIVERSITY DEHRADUN

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Lab Manual for the Academic Year 2019-20

Subject : Machine learning using R Lab
Subject code : CS368
Course coordinator : Dr. Ranjeet K. Ranjan
HOD : Prof. Vishal Bharti

Table of Contents

GENERAL INSTRUCTIONS FOR LABORATORY CLASSES:-

DO'S

(1) Without Prior permission do not enter into the Laboratory.

(2) While entering into the LAB students should carry ID cards.

(3) The Students should come with proper uniform.

(4) Students should come with the record note book into the laboratory.

(5) Students should maintain silence inside the laboratory.

(6) After completing the laboratory exercise, make sure to shut-down the system properly.

DONT'S

(1) Students bringing the bags inside the laboratory.

(2) Students wearing slippers/shoes insides the laboratory.

(3) Students using the computers in an improper way.

(4) Students bringing pen drive or other secondary storage device inside the laboratory.

(5) Students using mobile phones inside the laboratory.

(6) Students making noise inside the laboratory.

System Requirements

Minimum Hardware Requirements:-

Intel i3 and higher

RAM: 4GB

Hard Disk: 40 GB

Operating System: -

Windows XP/Windows 7/Windows 8/Linux

Software Requirement: -

R/R Studios

Lab Objectives

1. To provide students with a maiden concept of R programming in Data Science domain.

2. To provide students with an in-depth knowledge of working with different packages of R used for Different Data Science and Machine Learning Algorithms.

3. To enable students to write Programs in R for Data Analysis using Machine Learning Algorithms.

Practical – 1

Objective:- : R Environment Set-up

- Download set-up files
- Installation
- Environment Set-up
- Demo Program

Installation Guide: Practical 1 PPT

Demo Programs:

```
> abs(5*2*8.3-sqrt(16))
[1] 79
> 12-17*2/3-9
[1] -8.333333
> abs(12-17*2/3-9)
[1] 8.333333
>
> exp(10000)
[1] Inf
> exp(0.5)
[1] 1.648721
> ans1= 23+10*10-100
> ans1
[1] 23
> ans1
[1] 23
> ans2= 23*10+10*10-100
> ans2
[1] 230
> ans3= ans2/ans1
> ans3
[1] 10
> data1 c(2, 1, 4, 3, 5, 1, 6, 7, 10)
Error: unexpected symbol in "data1 c"
> data1= c(2, 1, 4, 3, 5, 1, 6, 7, 10)
> data1
[1]  2  1  4  3  5  1  6  7 10
> data2= c(data1, 8,7,6,5,4,3,2,1)
```

```
> data2
 [1] 2 1 4 3 5 1 6 7 10 8 7 6 5 4 3 2 1
> data3= c(data1, data2, data1, data2, data1)
```

Practical – 2

Objective: **Exp 2: R Objects (Data Types and Objects in R)**

a. Object, Vector, List, Factor
b. Matrix, Array, Data Frame
c. Manipulating Objects, R constructs

Objects :

Number

```
> x <- 1
> y <- 2.5
> class(x)
[1] "numeric"
> class(y)
[1] "numeric"
> class(x+y)
[1] "numeric"
```

Logical value

```
> m <- x > y     # Is x larger than y?
> n <- x < y     # Is x smaller than y?
> m
[1] FALSE
> n
```

```
[1] TRUE
> class(m)
[1] "logical"
> class(NA)       # NA is another logical value: 'Not Available'/Missing Values
[1] "logical"
```

Here are some logical operators you may want to try.

```
> m & n          # AND
[1] FALSE
> m | n          # OR
[1] TRUE
> !m             # Negation
[1] TRUE
```

Character(string)

```
> a <- "1"; b <- "2.5"       # Are they different from x and y we used earlier?
> a;b
[1] "1"
[1] "2.5"
> a+b                        # a+b=3.5?
Error in a + b : non-numeric argument to binary operator
> class(a)
[1] "character"
> class(as.numeric(a))       # but you can coerce this character into a number
[1] "numeric"
> class(as.character(x))     # vice resa
[1] "character"
```

Vector: A vector is a sequence of data elements of the same basic type.

```
> o <- c(1,2,5.3,6,-2,4)                          # Numeric vector
> p <- c("one","two","three","four","five","six")    # Character vector
> q <- c(TRUE,TRUE,FALSE,TRUE,FALSE,TRUE)          # Logical vector
> o;p;q
```

```
[1]  1.0  2.0  5.3  6.0 -2.0  4.0
[1] "one"   "two"   "three" "four"  "five"  "six"
[1]  TRUE  TRUE FALSE  TRUE FALSE
```

We talked about component extraction briefly in our first tutorial. Here are some other fun ways of doing that.

```
> o[q]                                  # Logical vector can be used to extract vector
components
[1] 1 2 6 4
> names(o) <- p                             # Give each component a name
> o
  one   two three  four  five   six
  1.0   2.0   5.3   6.0  -2.0   4.0
> o["three"]                              # Extract your components by "calling" their
names
three
  5.3
```

Matrix: A matrix is a collection of data elements arranged in a two-dimensional rectangular layout. Same as vector, the components in a matrix must be of the same basic type. The following is an example of a matrix with 4 rows and 3 columns.

```
> t <- matrix(
+     1:12,               # the data components (Don't type "+"!)
+     nrow=4,               # number of rows
+     ncol=3,             # number of columns
+     byrow = FALSE)       # fill matrix by columns
> t                     # print the matrix
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Similar to vectors, matrices also use [] to reference elements.

```
> t[2,3]                 # component at 2nd row and 3rd column
[1] 10
> t[,3]                  # 3rd column of matrix
[1]  9 10 11 12
> t[4,]                  # 4th row of matrix
[1]  4  8 12
> t[2:4,1:3]               # rows 2,3,4 of columns 1,2,3
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    3    7   11
[3,]    4    8   12
```

Data Frame: A data frame is more general than a matrix, in that different columns can have different basic data types. Data frame is the most common data type we are going to use in this class.

```
> d <- c(1,2,3,4)
> e <- c("red", "white", "red", NA)
> f <- c(TRUE,TRUE,TRUE,FALSE)
> mydata <- data.frame(d,e,f)
> names(mydata) <- c("ID","Color","Passed")      # variable names
> mydata
  ID Color Passed
1  1   red   TRUE
```

```
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

Extracting components from data frames is somehow similar to what we did for matrices, but after assigning names to each column (variable), it becomes more flexible.

```
> mydata$ID                  # try mydata["ID"] or mydata[1]
[1] 1 2 3 4
> mydata$ID[3]                # try mydata[3,"ID"] or mydata[3,1]
[1] 3
> mydata[1:2,]               # first two records
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
```

List: A list is a generic vector containing other objects. There is no restriction on data types or length of the components. Usually, we work with lists that have named components.

```
> l <-list(vec=p, mat=t, fra=mydata, count=3)                # a list with a vector, a matrix, a data frame defined earlier and a scalar
> l
$vec
[1] "one"   "two"   "three" "four"  "five"  "six"

$mat
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

```
$fra
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE

$count
[1] 3
> l$vec                                          # extract components from list
[1] "one"   "two"   "three" "four"  "five"  "six"
> l$mat[2,3]
[1] 10
> l$fra$Color
[1] red   white red   <NA>
Levels: red white
```

Object: In R, all types of data are treated as objects. However, objects are not simply collections of data. They are particular instances (instantiations) of particular classes. Operations, or functions, are defined for specific classes. Let's try working on something such as a point pattern.

```
# This time I will not show R outputs with codes. Just type or paste these lines into
R and see what you get.
x <- rnorm(50, 10, 3)              # creates 50 random x values from a normal
distribution
y <- rnorm(50, 10, 4)              # creates 50 random y values
mypoints <- as.data.frame(cbind(x,y)) # makes a data frame
class(mypoints)
mypoints
summary(mypoints)
plot(mypoints)                  # Gee, it looks like a point pattern...
box <- bbox(mypoints)              # Type in library(splancs) first. Bounding Box -
did this work? Why not?
```

It seems that most functions above work well with this data frame but "bbox" does not. See help(bbox). It didn't work because "bbox" doesn't work on objects of class data.frame. "bbox" operates on objects of class points (or a matrix of x and y values). Therefore you need to change the class accordingly. The following four approaches all work (try each one separately):

```
box <- bbox(cbind(x,y))
box <- bbox(as.matrix(mypoints))
box <- bbox(as.points(x,y))
box <- bbox(as.points(mypoints))
```

Practical – 3

Objective- R Data Handling

- Downloading Dataset
- Import/Export Dataset Files
- Summarization of dataset

STEP1: Download a dataset

STEP2: Read Datafile (.CSV)

STEP3: Write content of the file to another file with new name.

Reading data in a file:

For reading and writing in files, R uses the working directory. To find this directory, the command getwd() (get working directory) can be used, and the working directory can be changed with setwd("C:/data") or setwd("/home/ paradis/R").

The function read.table has for effect to create a data frame, and so is the main way to read data in tabular form. For instance, if one has a file named data.dat, the command:

```
> mydata <- read.table("data.dat")
```

will create a data frame named mydata, and each variable will be named, by default, V1, V2, . . . and can be accessed individually by mydata$V1, mydata$V2,..., or by mydata["V1"], mydata["V2"], ...,

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".", row.names, col.names, as.is =
FALSE, na.strings = "NA", colClasses = NA, nrows = -1,
skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip =
TRUE, comment.char = "#")
```

There are several options whose default values (i.e. those used by R if they are omitted by the user) are detailed in the following table:

`file`	the name of the file (within `""` or a variable of mode character), possibly with its path (the symbol \ is not allowed and must be replaced by /, even under Windows), or a remote access to a file of type URL (http://...)
`header`	a logical (`FALSE` or `TRUE`) indicating if the file contains the names of the variables on its first line
`sep`	the field separator used in the file, for instance `sep="\t"` if it is a tabulation
`quote`	the characters used to cite the variables of mode character
`dec`	the character used for the decimal point
`row.names`	a vector with the names of the lines which can be either a vector of mode character, or the number (or the name) of a variable of the file (by default: `1`, `2`, `3`, ...)
`col.names`	a vector with the names of the variables (by default: `V1`, `V2`, `V3`, ...)
`as.is`	controls the conversion of character variables as factors (if `FALSE`) or keeps them as characters (`TRUE`); `as.is` can be a logical, numeric or character vector specifying the variables to be kept as character
`na.strings`	the value given to missing data (converted as `NA`)
`colClasses`	a vector of mode character giving the classes to attribute to the columns
`nrows`	the maximum number of lines to read (negative values are ignored)
`skip`	the number of lines to be skipped before reading the data
`check.names`	if `TRUE`, checks that the variable names are valid for R
`fill`	if `TRUE` and all lines do not have the same number of variables, "blanks" are added
`strip.white`	(conditional to `sep`) if `TRUE`, deletes extra spaces before and after the character variables
`blank.lines.skip`	if `TRUE`, ignores "blank" lines
`comment.char`	a character defining comments in the data file, the rest of the line after this character is ignored (to disable this argument, use `comment.char = ""`)

The variants of read.table are useful since they have different default values:

read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".", fill = TRUE, ...)

read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",", fill = TRUE, ...)

read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".", fill = TRUE, ...)

read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec=",", fill = TRUE, ...)

The function read.fwf can be used to read in a file some data in fixed width format:

read.fwf(file, widths, header = FALSE, sep = "\t", as.is = FALSE, skip = 0, row.names, col.names, n = -1, buffersize = 2000, ...)

The options are the same than for read.table() ex- cept widths which specifies the width of the fields (buffersize is the maximum number of lines read si- multaneously). For example, if a file named data.txt has the data indicated on the right, one can read the data with the following command:

> mydata <- read.fwf("data.txt", widths=c(1, 4, 3))

> mydata

```
  V1   V2  V3
1  A 1.50 1.2
2  A 1.55 1.3
3  B 1.60 1.4
4  B 1.65 1.5
5  C 1.70 1.6
6  C 1.75 1.7
```

Output:

Saving data: The function write.table writes in a file an object, typically a data frame but this could well be another kind of object (vector, matrix, . . .). The arguments and options are:

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 ""

Practical – 4

Objective- **Data Pre-processing using R**

- Missing Value
- Outlier Handling
- Formatting data

Practical – 5

Objective- **Descriptive Statistics using R**

a. variance, standard deviation, shape – skewness, kurtosis, percentiles, five point summary
b. boxplots, histograms, bar plot, pie chart, scatter plot, two way tables,
c. covariance, correlation analysis, Chi-Square test for two way tables

Practical – 6

Objective- **Supervised Learning- Regressions using R**

- Linear Regression with one variable
- Linear Regression with multiple variable
- Polynomial regression

Practical – 7

Objective- **Supervised Learning- Classifications using R**

a. Logistic Regression
b. Decision Tree
c. k-Nearest Neighbors
d. Support Vector Machine

Practical – 8

Objective- **Unsupervised Learning- Clustering using R**

a. K-means clustering
b. Hierarchical clustering

Practical – 9

Objective- **Deep Learning using R**

a. Implementation of ANN
b. Implementation of CNN
c. Implementation of RNN

Practical – 10

Objective- **Transfer Learning using R**

a. Importing pre-trained models
b. Implementation of pre-trained models with a new learning model