Pratiksha Saxena
9871758015
Monika Kashyap

Programming in R: Basics & Text Analytics

[redacted]

Dr Vivek Kumar Singh
Department of Computer Science
Banaras Hindu University, Varanasi, India
&
Ashraf Uddin, Rajesh Piryani & Sumit Banshal
Department of Computer Science
South Asian University, New Delhi

About R

What is R?

- ✓R is a dialect of the S language.
- ✓R is a free software programming language
- ✓software environment for statistical computing and graphics
- ✓ widely used among statisticians and data miners for developing statistical software and data analysis

✓The source code for the R software environment is written primarily in C, Fortran, and R

→ It is programming language & env. commonly used in statistical computing, data analytics & scientific research
→ It is used to retrieve, clean, analyze, visualize and present data
→ expressive syntax & easy to use

History of R

✓1991: Created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand

✓1993: First announcement of R to the public.

✓1995: R was made as free software.

✓1997: The R Core Group is formed (containing some people associated with S-PLUS). The core group controls the source code for R.

✓2000: R version 1.0.0 is released.

✓2013: R version 3.1.2 has been released on 2014-10-31.

Statistical features of R

✓provides a wide variety of statistical and graphical techniques:

- ✓linear and nonlinear modelling
- ✓classical statistical tests
- ✓ time-series analysis,
- ✓classification, clustering
- ✓Others

✓easily extensible through functions and extensions

✓Many of R's standard functions are written in R itself

✓C, C++, and Fortran code can be linked and called at run time

✓Dynamic and interactive graphics are available through additional packages

Programming features of R

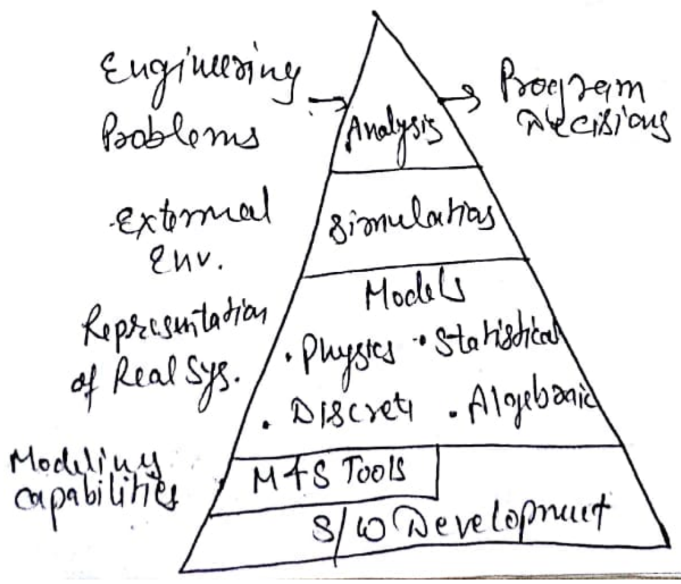
- ✓R is an interpreted language, users typically access it through a command-line interpreter.
- ✓Like other similar languages such as MATLAB, R supports matrix arithmetic
- ✓R supports procedural programming with functions
- ✓for some functions, object-oriented programming with generic functions

Features of R continued...

- ✓functionality is divided into modular packages
- ✓Graphics capabilities very sophisticated.
- ✓Useful for interactive work, but contains a powerful programming language for developing new tools
- ✓Very active and vibrant user community; R-help and (R-devel or R-patched mailing lists and Stack Overflow

problem discussing

purposal of new fxnality



Design of the R System

- ✓The R system is divided into 2 conceptual parts:
  - ✓The "base" R system that you download from CRAN
  - ✓Everything else.
- ✓R functionality is divided into a number of packages
  - ✓The "base" R system contains, among other things, the base package which is required to run R and contains the most fundamental functions.
  - ✓The other packages contained in the "base" system include utils, stats, datasets, graphics, grDevices, grid, methods, tools, parallel, compiler, splines, tcltk, stats4.
  - ✓There are also other packages: tm, stringr, boot, class, cluster, codetools, foreign, KernSmooth, lattice, mgcv, nlme, rpart, survival, MASS, spatial, nnet, Matrix.

It is a N/w to ftp and web servers around the world that store identical, up-to-date, versions of code and doc. for R.

Design of the R System continued...

- ✓And there are many other packages available:
  - ✓There are about 4000 packages on CRAN that have been developed by users and programmers around the world.

Start Working in R

✓Download & Install R: http://www.r-project.org/

✓Download & Install R studio: http://www.rstudio.com/products/rstudio/download/, Wikipedia

✓Materials:

- ✓Chambers (2008). *Software for Data Analysis, Springer. (your textbook)*
- ✓Chambers (1998). *Programming with Data, Springer.*
- ✓Venables & Ripley (2002). *Modern Applied Statistics with S, Springer.*
- ✓Venables & Ripley (2000). *S Programming, Springer.*
- ✓Pinheiro & Bates (2000). *Mixed-Effects Models in S and S-PLUS, Springer.*
- ✓Murrell (2005). *R Graphics, Chapman & Hall/CRC Press.*
- ✓Springer has a series of books called *Use R!.*
- ✓A longer list of books is at http://www.r-project.org/doc/bib/R-books.html

✓Course on R: https://www.coursera.org/course/rprog

Data Types and Basic Operations

Objects

R has five basic or "atomic" classes of objects:

1. Character
2. numeric (real numbers)
3. Integer
4. Complex
5. logical (True/False)

The most basic object is a vector

✓A vector can only contain objects of the same class

✓BUT: The one exception is a list, which is represented as a vector but can contain objects of different classes.

Data Types and Basic Operations continued...

Numbers

✓Numbers in R a generally treated as numeric

✓a special number Inf which represents infinity; e.g. 1 / 0; Inf can be used in ordinary calculations; e.g. 1 / Inf is 0

✓The value NaN represents an undefined value ("not a number"); e.g. 0 / 0; NaN can also be thought of as a missing value

Data Types and Basic Operations continued...

Attributes

R objects can have attributes

- ✓names, dimnames
- ✓dimensions (e.g. matrices, arrays)
- ✓Class
- ✓Length

```
> msg<- "hello"
>class(msg)
[1]"character"
> length(x)
[1] 1
```

Qu what is assignment in R?

Ans. objects obtain values in R by assigning (x gets a value)

[<- or =] | → x <- 6 or x = 6
y <- "a" or y = "a"

Data Types and Basic Operations continued...

< assignment
comments.
: create sequences

Entering Input

The <- symbol is the assignment operator.

```
> x <- 1
> print(x)
[1] 1
> x
[1] 1
```

The grammar of the language determines whether an expression is complete or not.

```
> x <- ## Incomplete expression
```

The # character indicates a comment. Anything to the right of the # (including the # itself) is ignored.

Data Types and Basic Operations continued...

Printing

```
> x <- 1:20
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
[16] 16 17 18 19 20
```

The : operator is used to create integer sequences.

→ operations in R

operators	fxn
+ - * / %% ^	Arithmetic
>= <= == !=	Relational
! +	logical
~	Module formule
<- ->	Assignment
$	list indexing (the 'element name' operator)
:	create a sequence

objects
1 Vector
2 Matrices
3 Lists
4 Data types

Complex - Data Types and Basic Operations continued...

Creating Vectors

[contain similar types of data)

The c() function can be used to create vectors of objects.

typeof()
for checking type of data

```
> x <- c(0.5, 0.6) ## numeric
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
> x <- 9:29 ## integer
> x <- c(1+0i, 2+4i) ## complex
```

Using the vector() function

```
> x <- vector("numeric", length = 10)
> x
[1] 0 0 0 0 0 0 0 0 0 0
```

Data Types and Basic Operations continued...

Mixing Objects

What about the following?

```
> y <- c(1.7, "a") ## character
> y <- c(TRUE, 2) ## numeric
> y <- c("a", TRUE) ## character
```

When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.

Case sensitive

Data Types and Basic Operations continued...

Matrices (Homogeneous data)

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)

```
> m <- matrix(nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
> dim(m)
[1] 2 3
> attributes(m)$dim
[1] 2 3
```

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

```
> m <- matrix(c(1,2,3,4,
5,6), nrow = 3)

> m
[,1] [,2]
1 4
2 5
3 6
```

→ Row major
→ Column major
→ rowSums()
→ colSums()
→ rbind()
→ cbind()

Data Types and Basic Operations continued...

Matrices: Matrix sum & multiplication

```
> m<-matrix(data=c(1,0,0,4,4,3), nrow=2,ncol=3)
> n<-matrix(data=c(1,2,3,4,5,6), nrow=2,ncol=3)
> m+n
[,1] [,2] [,3]
[1,] 2 3 9
[2,] 2 8 9
> m*n
[,1] [,2] [,3]
[1,] 1 0 20
[2,] 0 16 18
> m%*%n
Error in m %*% n : non-conformable arguments
>n<-matrix(data=c(1,2,3,4), nrow=2,ncol=2)
>n %*% m
[,1] [,2] [,3]
[1,] 1 12 13
[2,] 2 16 20
```

$\begin{bmatrix} 1 & 0 & 4 \\ 0 & 4 & 3 \end{bmatrix}$ $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$

$\begin{bmatrix} 2 & 3 & 9 \\ 2 & 8 & 9 \end{bmatrix}$

$\begin{bmatrix} 1 & 0 & 20 \\ 0 & 16 & 18 \end{bmatrix}$

100 123
443 456

Scanned by CamScanner

Data Types and Basic Operations continued...

Lists [diff. length and types of data)

Lists are a special type of vector that can contain elements of different classes. Lists are a very important data type in R and you should get to know them well. similar to C Struct.

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1
[[2]]
[1] "a"
[[3]]
[1] TRUE
[[4]]
[1] 1+4i
```

[[ ]] for single element

[ ] for group

→ Add, delete

→ indexing [2, 3]

→

→ size of list:- length()

→ Recursive list [Lists within lists]

Data Types and Basic Operations continued...

Data Frames [same length yet not necessarily of the same Type.]

✓Data frames are used to store tabular data

✓They are represented as a special type of list where every element of the list has to have the same length

✓Each element of the list can be thought of as a column and the length of each element of the list is the number of rows

✓Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class

→ in Rectangular Matrix.

Data Types and Basic Operations continued...

Data Frames

```
> x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
> x
      foo    bar
1     1      TRUE
2     2      TRUE
3     3      FALSE
4     4      FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

→ indexing ( )
→ filtering ( )
→ Apply ( )

→ convert a list to a data frame.

Data frames (as csvfile)

```
> data<-read.csv("G:/records.csv")
>cd<-data[data$PY==2000,]
>cd<-data[data$PY==2012,]
```

Reading and Writing Data continued...

Reading Data

There are a few principal functions reading data into R.

- ✓read.table, read.csv, for reading tabular data
- ✓readLines, for reading lines of a text file

Reading and Writing Data continued...

Writing Data

There are analogous functions for writing data to files

- ✓write.table
- ✓writeLines
- ✓save

Reading and Writing Data continued...

15
21
38
74

Reading Lines of a Text File

```
# read a csv file
con <- file("txt/sport2.txt", "r")
x <- readLines(con, 3)
```

```
## This might take time
>con <- url("http://www.jhsph.edu", "r")
>x <- readLines(con)
> head(x)
[1] "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">"
[2] ""
[3] "<html>"
[4] "<head>"
[5] "\t<meta http-equiv=\"Content-Type\" content=\"text/html;charset=utf-8
```

Functions

✓Functions are created using the function() directive and are stored as R objects just like anything else.

```
f <- function(<arguments>) {
## Do something interesting
}
```

✓Functions can be passed as arguments to other functions

✓Functions can be nested

✓The return value of a function is the last expression in the function body to be evaluated.

Functions continued...

Defining a Function

```
average<-function(array=numeric(1)){
  sum<-0
  for(i in 1: length(array)){
    sum<-sum+array[i]
  }
  value<-sum/length(array)
  value
}
```

```
> m<-c(10,11,2)
> average(m)
[1] 7.666667

> average(10)
[1] 10

> average()
[1] 0
```

Implementation: Largest number in a matrix

```
> m <- matrix(data=c(1,6,23,45,78,12,11,7,4), nrow=3, ncol =3)
> max(m)
[1] 78
> which.max(m)
[1] 5
```

position

Implementation: Sort an array of numbers

```
>d <- c(1,6,23,45,78,12,11,7,4)
>sort(d)
[1] 1 4 6 7 11 12 23 45 78
>sort(d, decreasing = T)
[1] 78 45 23 12 11 7 6 4 1
```

Implementation: Find a word if occurs in a file

```
txt <- readLines("txt/sport1.txt")
library(stringr)
ind <- which(str_detect(txt,"cricket"))
txt[ind]
```

Implementation: Create TDM from docs

```
library(tm)
crp <- Corpus(DirSource("txt")) # change directory path if required
crp <- tm_map(crp,tolower)
crp <- tm_map(crp,removeWords,stopwords("english"))
crp <- tm_map(crp,removePunctuation)
crp <- tm_map(crp,removeNumbers)
crp <- tm_map(crp,stripWhitespace)
crp <- tm_map(crp,PlainTextDocument)
dtm <- TermDocumentMatrix(crp, control = list(weighting = weightTfIdf))

dim(dtm)
dtm$dimnames$Terms
m <- as.matrix(dtm)
```

Implementation: POS Tagging

```
## packages NLP, openNLP
library("tm")
library("NLP")
library("openNLP")
## Some text.

data("acq")

s <- as.String(acq[[10]])
## Need sentence and word token annotations.
sent_token_annotator <- Maxent_Sent_Token_Annotator()
word_token_annotator <- Maxent_Word_Token_Annotator()
a2 <- annotate(s, list(sent_token_annotator,
word_token_annotator))
pos_tag_annotator <- Maxent_POS_Tag_Annotator()
#pos_tag_annotator
a3 <- annotate(s, pos_tag_annotator, a2)
a3w <- subset(a3, type == "word")
tags <- sapply(a3w$features, "[[", "POS")
show(sprintf("%s/%s", s[a3w], tags))
```

Implementation: Word Frequency

```
text.files<-list.files(path="txt",full.names = T)
words<-character()
for(fp in text.files){
  data<-readLines(con = fp) #read text file line by line
  # extract words from each line
  for(line in 1: length(data)){
    if(data[line]!=""){
      list<-unlist(strsplit(data[line]," "))
      list<-list[list!=""] #remove the empty strings
      words<-c(words,list)
    }
  }
}
word_freq <- sort(table(words),decreasing = T)
head(word_freq,20)
```

Implementation: Extract emails from url

```
# extract email from web page
html_text <- readLines("http://www.viveksingh.in/")
txt <- paste(html_text,collapse = "")
str_extract_all(txt, "[a-z0-9]+\\@[a-z0-9]+\\.[a-z0-9]+")
```

Implementation: Text Classification

- ✓Training data set
- ✓Test data set
- ✓Data set (Training +Test data set)

- ✓**Example:** Sports, News, Opinion/ Reviews

- ✓Two basic steps
  - ✓Representation of text documents (TDM)
  - ✓Supervised/ Unsupervised algorithm

Implementation: Text Classification

```
options(stringsAsFactors = F)

libs <- c("tm", "plyr", "class")
lapply(libs, require, character.only = TRUE)

#read data
d <- read.csv("data.csv") # download sample data from link at the
bottom
textVector <- do.call(paste, d[ , 1:3]) #data columns
taggedValues <- d$Tagged #tag column
```

Implementation: Text Classification continue...

```
generateTDM <- function(cls, size, dataVector, tags){
  # cls = classes ex. 1:4 for 4 class classification
  # size = number of samples in each class
  # dataVector = data as a character vector
  # tags = original tag/class
  s <- (cls - 1)*size + 1
  e <- cls*size

  crps <- Corpus(VectorSource(dataVector[s : e]))
  crps <- tm_map(crps,tolower)
  crps <- tm_map(crps,removeWords,stopwords("english"))
  crps <- tm_map(crps,removePunctuation)
  crps <- tm_map(crps,removeNumbers)
  crps <- tm_map(crps,stripWhitespace)
  crps <- tm_map(crps,PlainTextDocument)

  dtm <- TermDocumentMatrix(crps, control = list(weighting =
weightTfIdf))
  colnames(dtm) <- s:e
  dtm <- removeSparseTerms(dtm, 0.9)
  result <- list(name =  tags[s], tdm = dtm)
}
# generate indpendent TDM for each class document set
tdm<-lapply(1:4, generateTDM, 20, textVector, taggedValues)
```

Implementation: Text Classification continue...

```
bindClassToTDM<-function(tdm){
  mat <- t(data.matrix(tdm[["tdm"]]))
  df <- as.data.frame(mat, StringAsFactors = FALSE)
  df <- cbind(df,rep(tdm[["name"]], nrow(df)))
  colnames(df)[ncol(df)] <- "taggedClass"
  return (df)
}
```

Implementation: Text Classification continue...

```
bindClassToTDM<-function(tdm){
  mat <- t(data.matrix(tdm[["tdm"]]))
  df <- as.data.frame(mat, StringAsFactors = FALSE)
  df <- cbind(df,rep(tdm[["name"]], nrow(df)))
  colnames(df)[ncol(df)] <- "taggedClass"
  return (df)
}
# bind a tag column to each TDM
candTDM <- lapply(tdm, bindClassToTDM)

# cobine all TDMs into one TDM
tdmStack <- do.call(rbind.fill, candTDM)
tdmStack[is.na(tdmStack)] <- 0
```

Scanned by CamScanner

Implementation: Text Classification continue...

```
# Naive Bayes classifier using N folds
library("e1071")
df <- tdmStack[ , !colnames(tdmStack) %in% "taggedClass"]
cl <- as.factor(tdmStack$taggedClass)
nfold <- 5
C <- length(unique(cl))
N <- nrow(df)/C
n <- N/nfold
predNB <- predSVM <- cl
for(i in 1: nfold){
  show(i)
  s <- (i - 1)*n + 1
  e <- i*n
  testInd <- NULL
  for(j in 1: C){
    ind <- ((j-1)*N + s) : ((j-1)*N + e)
    testInd <- c(testInd, ind)
  }
  # Naive Nayes
  modelNB <- naiveBayes(x = df[-testInd,], y = cl[-testInd])
  predNB[testInd] <- predict(modelNB, df[testInd,])
}

#confusion matrix
table(predNB, cl)

#accuracy
show(sum(diag(table(predNB, cl)))/length(cl))
```

Implementation: Sentiment Classification

```
options(stringsAsFactors=FALSE)
#loading libraries
library("tm")
library(e1071)
library("RWeka")

#reading data from csv file
mr=read.csv("data_sa.csv")
#reading data using vector sorce from read data
corpus=Corpus(VectorSource(mr$Review))
#removing punction marks from corpus
corpus=tm_map(corpus, removePunctuation)
# removing Numbers from corpus
corpus=tm_map(corpus, removeNumbers)
#changing the case of text to lower
corpus=tm_map(corpus, tolower)
#creating documentTermMatrix
corpus <- tm_map(corpus,PlainTextDocument)
dtm=DocumentTermMatrix(corpus, control = list(weighting = weightTfIdf))
```

Implementation: Sentiment Classification

```
#5th and 6th document is taken as test document
test=c(5,6);
#number of document
ndocs=length(test);
#prediction result
predictions=NULL
#building model for dataset
model=naiveBayes(as.matrix(dtm[-test,]), as.factor(ms$Class[-test]))
#Predicting for test document
predictions=predict(model, as.matrix(dtm[test,]))
#prediction result
predictions
```

Implementation: Text Classification

Making TDM (Term Document Matrix):

- ✓Making Corpus
- ✓Clean Corpus (removing punctuation, stop words, white space, lower case)

Data and codes: http://bit.ly/basicRtutorial

Examples:

http://ashrafsau.blogspot.com/2017/01/text-classification-using-naive-bayes.html

R

→ op Bayes,

→ R initialization + Batch Multi

→ R - DS - Matrics
list
data f.
classes

Thank You

Thursday, 11-12:00

1 2 3 4 5.

6.7

6 14 18 28 30

8.

12

Implementation: Text Classification

Scanned by CamScanner

Interactive Mode is the simplest way to work on a system. You log in, run commands which execute immediately and log off when you have finished.

→ You can use either command line or a graphical env. These jobs run directly on the limited no. of login nodes on each cluster.

→ Short tasks
→ Tasks that require frequent user interaction
→ Graphical intensive tasks

Batch Processing:-

→ More complex because work has to carefully plann
→ jobs submitted to job scheduller.
→ Uses queue for waiting
→ Longer running processes.
→ Parallel processes
→ Running large no. of short jobs simultaneously
→ Tasks that can be left running for significant amount of time without any interaction.

Batch Mode :- Much of the time, R is used interactively : a user sits in front of a computer and types instructions in R language at the command line. ~~You can prepare~~ The instructions are executed, the result is displayed on screen and then R waits for the next command.

But R can also be used non-interactively, you can prepare a sequence of commands in advance as a script file and have R execute those commands in batch mode, without ever waiting for human intervention.

→ Batch Mode is used for simulation or analysis.
→ It is execution of series of programs ('jobs') with human interaction.
→ Can run non-interactively, so all data (input) is preselected through scripts or command line parameters.
→ input file from → "infile"
  output → "outfile"] Can also pass arguments.

The cat command takes input from Keyboard and redirect it to a file:-

```
$ cat > hello_world.R
# hello world example
a <- c("Hello, world!")
print (a)
```

output →

```
$ R --vanilla --slave < hello_world.R
```

This above called non-interactively.

⇒ For Redirect the file use —

```
$ R --vanilla --slave < hello_world.R > result.txt
```