Data Analysis in Software Engineering using R

Daniel Rodriguez and Javier Dolado

2022-08-04

Contents

Welcome **5**

I Introduction to the R Language 7

1 Introduction to R **9**

1.1 Installation . . . 9
1.2 R and RStudio . . . 10
1.3 Basic Data Types . . . 15
1.4 Vectors . . . 18
1.5 Arrays and Matrices . . . 22
1.6 Factors . . . 28
1.7 Lists . . . 30

2 Split data into training and test datasets **31**

2.1 The caret package . . . 34

3 Regression **35**

3.1 Linear Regression modeling . . . 35
3.2 Linear Regression Diagnostics . . . 39
3.3 Multiple Linear Regression . . . 46
3.4 Linear regression in Software Effort estimation . . . 47
3.5 References . . . 51

II Unsupervised Models 53

4 Unsupervised or Descriptive modeling **55**

4.1 Clustering . . . 55
4.2 Association rules . . . 56

III Evaluation **59**

5 Evaluation of Models **61**
5.1 Building and Validating a Model 61
5.2 Evaluation of Classification Models 62
5.3 Other Metrics used in Software Engineering with Classification . 65
5.4 Graphical Evaluation 66
5.5 Numeric Prediction Evaluation 67

6 Measures of Evaluation in Software Engineering **69**
6.1 Effort estimation evaluation metrics 69
6.2 Evaluation of the Model in the Testing data 70
6.3 Building a Linear Model on the Telecom1 dataset 72
6.4 Building a Linear Model on the Telecom1 dataset with all observations 73
6.5 Standardised Accuracy Examples 74
6.6 Exact MARP0 77
6.7 Computing the bootstraped confidence interval of the mean for the Test observations of the China dataset: 78
6.8 Defect prediction evaluation metrics 80

IV Advanced Topics **81**

7 Feature Selection **83**
7.1 Instance Selection 83
7.2 Missing Data Imputation 84

8 Feature Selection Example **85**

9 Further Classification Models **87**
9.1 Multilabel classification 87
9.2 Semi-supervised Learning 87

10 Social Network Analysis in SE **89**

11 Text Mining Software Engineering Data **93**
11.1 Terminology 93
11.2 Example of classifying bugs from Bugzilla 94
11.3 Extracting data from Twitter 101

12 Time Series **103**
12.1 Web tutorials about Time Series: 106

V Bibliography **107**

Welcome

This **Data Analysis in Software Engineering (DASE)** book/notes will try teach you how to do data science with R in Software Engineering.

It is a work in progress.

Acknowledgments

Projects:

- PRESI: TIN2013-46928-C3
  - amuSE TIN2013-46928-C3-2-R
  - PERTEST TIN2013-46928-C3-1-R
- QARE: TIN2016-76956-C3
  - BadgePeople: TIN2016-76956-C3-3-R
  - TESTEAMOS: TIN2016-76956-C3-1-R
- Network SBSE (SEBASENet): TIN2015-71841-REDT
- TestBUS PID2019-105455GB-C32

Part I

Introduction to the R Language

Chapter 1

Introduction to R

The goal of the first part of this book is to get you up to speed with the basics of **R** as quickly as possible.

1.1 Installation

Install the latest preview version for getting all features.

Follow the procedures according to your operating system.

- Linux: You need to have `blas` and `gfortran` installed on your Linux, for installing the `coin` package.
- *Rgraphviz* requires installation from `source("http://bioconductor.org/biocLite.R")`, then `biocLite("Rgraphviz")`.
- Run the following lines for installing all needed packages (this may take some time):

```
## listofpackages <- c("arules","arulesViz", "bookdown", "ggplot2", "vioplot", "UsingR", "fpc", "

# newpackages <- listofpackages[!(listofpackages %in% installed.packages()[,"Package"])]
# if(length(newpackages)>0) install.packages(newpackages,dependencies = TRUE)

# # install from archive (RPG is no maintained anymore)
# if (!is.element("rgp", installed.packages()[,1]))
# { install.packages("https://cran.r-project.org/src/contrib/Archive/rgp/rgp_0.4-1.tar.gz",
#                                                         repos = NULL)
# }
## end of installing packages

# in Linux you may need to run several commands (in the terminal) and install additional librarie
# sudo R CMD javareconf
# sudo apt-get install build-essential
# sudo apt-get install libxml2-dev
# sudo apt-get install libpq
# sudo apt-get install libpq-dev
# sudo apt-get install -y libmariadb-client-lgpl-dev
# sudo apt-get install texlive-xetex
# sudo apt-get install r-cran-rmysql
```

1.2 R and RStudio

- R is a programming language for statistical computing and data analysis that supports a variety of programming styles. See R in Wikipedia
- R has multiple online resources and books.
- R coding style
- R-Bloggers
- Getting help in R
  - RStudio cheat sheet
  - Base R cheat sheet
  - Advanced R cheat sheet
  - Data Visualization cheat sheet
  - R Markdown cheatsheet
  - [R Markdown Basics] (http://rmarkdown.rstudio.com/authoring_basics.html)
  - Python with R and Reticulate Cheatsheet
  - caret
  - All cheatsheets and translations
  - `help("    ")` command
- R as a calculator. Console: It uses the command-line interface.
- This document is an RMarkdown document. See bookdown.org

Examples:

```
x <- c(1,2,3,4,5,6)    # Create ordered collection (vector)
y <- x^2               # Square the elements of x
print(y)               # print (vector) y
```

```
## [1]  1  4  9 16 25 36
```

```
mean(y)                # Calculate average (arithmetic mean) of (vector) y; result is s
```

```
## [1] 15.16667
```

```
var(y)                 # Calculate sample variance
```

```
## [1] 178.9667
```

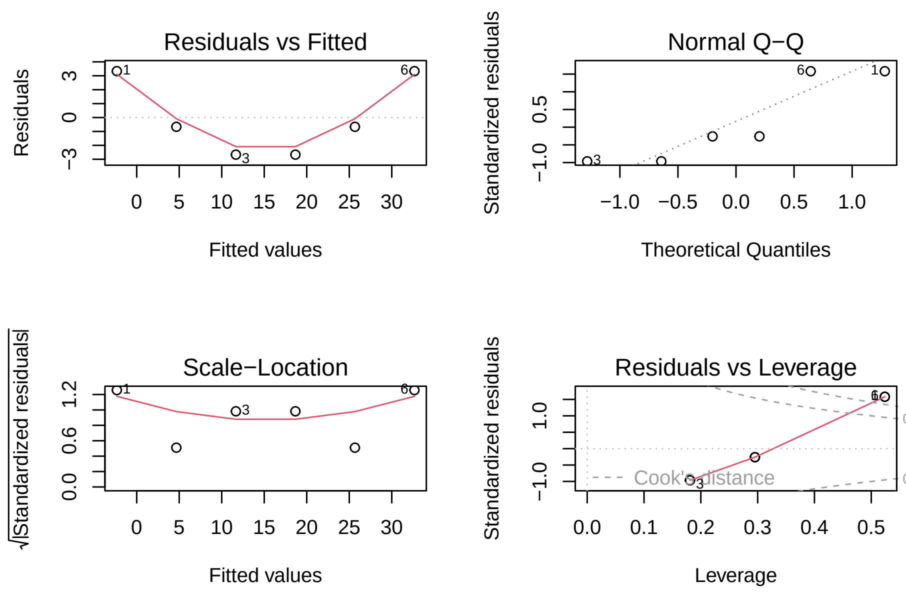
```
lm_1 <- lm(y ~ x)      # Fit a linear regression model "y = f(x)" or "y = B0 + (B1 * x)"
                       # store the results as lm_1
print(lm_1)            # Print the model from the (linear model object) lm_1
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##      -9.333        7.000
```

```
summary(lm_1)          # Compute and print statistics for the fit
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       1       2       3       4       5       6
##  3.3333 -0.6667 -2.6667 -2.6667 -0.6667  3.3333
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -9.3333     2.8441  -3.282 0.030453 *
## x             7.0000     0.7303   9.585 0.000662 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.055 on 4 degrees of freedom
## Multiple R-squared:  0.9583, Adjusted R-squared:  0.9478
## F-statistic: 91.88 on 1 and 4 DF,  p-value: 0.000662
```

```
                       # of the (linear model object) lm_1
par(mfrow=c(2, 2))     # Request 2x2 plot layout
plot(lm_1)             # Diagnostic plot of regression model
```



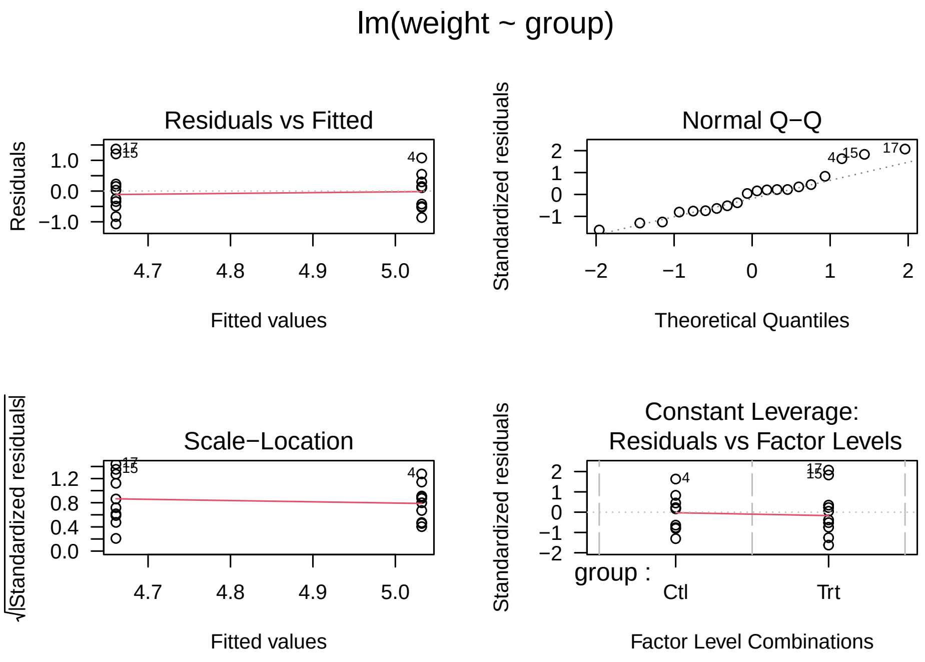
```
help(lm)
?lm
apropos("lm")
```

```
##  [1] ".colMeans"       ".lm.fit"         "colMeans"        "confint.lm"
##  [5] "contr.helmert"   "dummy.coef.lm"   "glm"             "glm.control"
##  [9] "glm.fit"         "KalmanForecast"  "KalmanLike"      "KalmanRun"
## [13] "KalmanSmooth"    "kappa.lm"        "lm"              "lm_1"
## [17] "lm.fit"          "lm.influence"    "lm.wfit"         "model.matrix.lm"
## [21] "nlm"             "nlminb"          "predict.glm"     "predict.lm"
## [25] "residuals.glm"   "residuals.lm"    "summary.glm"     "summary.lm"
```

```
example(lm)
```

```
##
## lm> require(graphics)
##
## lm> ## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## lm> ## Page 9: Plant Weight Data.
## lm> ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
##
## lm> trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
##
## lm> group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
##
## lm> weight <- c(ctl, trt)
##
## lm> lm.D9 <- lm(weight ~ group)
```

```
##
## lm> lm.D90 <- lm(weight ~ group - 1) # omitting intercept
##
## lm> ## No test:
## lm> ##D anova(lm.D9)
## lm> ##D summary(lm.D90)
## lm> ## End(No test)
## lm> opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
##
## lm> plot(lm.D9, las = 1)      # Residuals, Fitted, ...
```



```
##
## lm> par(opar)
##
## lm> ## Don't show:
## lm> ## model frame :
## lm> stopifnot(identical(lm(weight ~ group, method = "model.frame"),
## lm+                     model.frame(lm.D9)))
##
## lm> ## End(Don't show)
## lm> ### less simple examples in "See Also" above
## lm>
## lm>
## lm>
```

- R script. # A file with R commands `# comments` `source("filewithcommands.R")` `sink("recordmycommands.lis") savehistory()`

- From command line:
    - Rscript
    - Rscript file with `-e` (e.g. `Rscript -e 2+2`)
    - To exit R: `quit()`
- Variables. R is case sensitive

```
var1 <- 1:10
vAr1 <- 11:20
var1
```

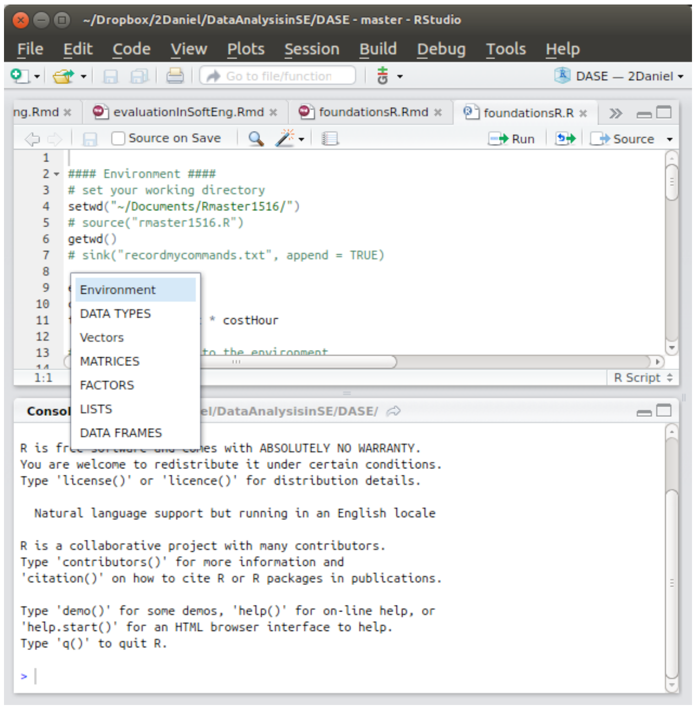
```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
vAr1
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20
```

- Operators
    - assign operator `<-`
    - sequence operator, for example: `mynums <- 0:20`
    - arithmetic operators: + - = / ^ %/% (integer division) %% (modulus operator)
- The Workspace. Objects.
    - `ls()` `objects()` `ls.str()` lists and describes the objects
    - `rm(x)` delete a variable. E.g., `rm(totalCost)`
    - `s.str()`
    - `objects()`
    - `str()` The structure function provides information about the variable
- RStudio, RCommander and RKWard are the well-known IDEs for R (more later).

- Four # (‘####’) create an *environment* in RStudio. An environment binds a set of names to a set of values. You can think of an environment as a bag of names.
    - Environment basics



```
#### Environment ####
# set your working directory
setwd("~/Documents/Rmaster1516/")
# source("rmaster1516.R")
getwd()
# sink("recordmycommands.txt", append = TRUE)
```

```
R is fr                es with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Working directories:

```r
# set your working directory
# setwd("~/workingDir/")
getwd()
```

```
## [1] "/home/drg/Projects/DASE"
```

```r
# record R commands:
# sink("recordmycommands.txt", append = TRUE)
```

1.3 Basic Data Types

- `class( )`
- logical: `TRUE`, `FALSE`
- numeric, integer:
  - `is.numeric( )`
  - `is.integer( )`
- `character`

Examples:

```
TRUE
```

```
## [1] TRUE
```

```
class(TRUE)
```

```
## [1] "logical"
```

```
FALSE
```

```
## [1] FALSE
```

```
NA  # missing
```

```
## [1] NA
```

```
class(NA)
```

```
## [1] "logical"
```

```
T
```

```
## [1] TRUE
```

```
F
```

```
## [1] FALSE
```

```
NaN
```

```
## [1] NaN
```

```
class(NaN)
```

```
## [1] "numeric"
```

```
# numeric data type
2
```

```
## [1] 2
```

```
class(2)
```

```
## [1] "numeric"
```

```
2.5
```

```
## [1] 2.5
```

```
2L  # integer
```

```
## [1] 2
```

```
class(2L)
```

```
## [1] "integer"
```

```
is.numeric(2)
```

```
## [1] TRUE
```

```
is.numeric(2L)
```

```
## [1] TRUE
```

```
is.integer(2)
```

```
## [1] FALSE
```

```
is.integer(2L)
```

```
## [1] TRUE
```

```
is.numeric(NaN)
```

```
## [1] TRUE
```

- data type coercion:
  - `as.numeric(  )`
  - `as.character(  )`
  - `as.integer(  )`

Examples:

```
truenum <- as.numeric(TRUE)
truenum
```

```
## [1] 1
```

```
class(truenum)
```

```
## [1] "numeric"
```

```
falsenum <- as.numeric(FALSE)
falsenum
```

```
## [1] 0
```

```
num2char <- as.character(55)
num2char
```

```
## [1] "55"
```

```
char2num  <- as.numeric("55.3")

char2int  <- as.integer("55.3")
```

1.3.1 Mising values

- `NA` stands for Not Available, which is not a number as well. It applies to missing values.
- `NaN` means 'Not a Number'

Examples:

```
NA + 1
```

```
## [1] NA
```

```
mean(c(5,NA,7))
```

```
## [1] NA
```

```
mean(c(5,NA,7), na.rm=TRUE)  # some functions allow to remove NAs
```

```
## [1] 6
```

1.4 Vectors

Examples:

```
phases <- c("reqs", "dev", "test1", "test2", "maint")
str(phases)
```

```
##  chr [1:5] "reqs" "dev" "test1" "test2" "maint"
```

```
is.vector(phases)
```

```
## [1] TRUE
```

```
thevalues <- c(15, 60, 30, 35, 22)
names(thevalues) <- phases
str(thevalues)
```

```
##  Named num [1:5] 15 60 30 35 22
##  - attr(*, "names")= chr [1:5] "reqs" "dev" "test1" "test2" ...
```

```
thevalues
```

```
##  reqs   dev test1 test2 maint
##    15    60    30    35    22
```

A single value is a vector! Example:

```
aphase <- 44
is.vector(aphase)
```

```
## [1] TRUE
```

```
length(aphase)
```

```
## [1] 1
```

```
length(thevalues)
```

```
## [1] 5
```

1.4.1 Coercion for vectors

```
thevalues1 <- c(15, 60, "30", 35, 22)
class(thevalues1)
```

```
## [1] "character"
```

```
thevalues1
```

```
## [1] "15" "60" "30" "35" "22"
```

```
# <-  is equivalent to   assign ( )

assign("costs", c(50, 100, 30))
```

1.4.2 Vector arithmetic

The operation is carried out in all the elements of the vector. For example:

```
assign("costs", c(50, 100, 30))
costs/3
```

```
## [1] 16.66667 33.33333 10.00000
```

```
costs - 5
```

```
## [1] 45 95 25
```

```
costs <- costs - 5

incomes <- c(200, 800, 10)
earnings <- incomes - costs
sum(earnings)
```

```
## [1] 845
```

```
# R recycles values in vectors!
vector1 <- c(1,2,3)
vector2 <- c(10,11,12,13,14,15,16)
vector1 + vector2
```

```
## Warning in vector1 + vector2: longer object length is not a multiple of shorter
## object length
## [1] 11 13 15 14 16 18 17
```

Subsetting vectors

```
### Subsetting vectors  []

phase1 <- phases[1]
phase1
```

```
## [1] "reqs"
```

```
phase3 <- phases[3]
phase3
```

```
## [1] "test1"
```

```
thevalues[phase1]
```

```
## reqs
##   15
```

```
thevalues["reqs"]
```

```
## reqs
##   15
```

```
testphases <- phases[c(3,4)]
thevalues[testphases]
```

```
## test1 test2
##    30    35
```

```
### Negative indexes

phases1 <- phases[-5]
phases
```

```
## [1] "reqs"  "dev"   "test1" "test2" "maint"
```

```
phases1
```

```
## [1] "reqs"  "dev"   "test1" "test2"
```

```
#phases2 <- phases[-testphases] ## error in argument
phases2 <- phases[-c(3,4)]
phases2
```

```
## [1] "reqs"  "dev"   "maint"
```

```
### subset using logical vector

phases3 <- phases[c(FALSE, TRUE, TRUE, FALSE)] #recicled first value
phases3
```

```
## [1] "dev"   "test1"
```

```
selectionv <- c(FALSE, TRUE, TRUE, FALSE)
phases3 <- phases[selectionv]
phases3
```

```
## [1] "dev"   "test1"
```

```
selectionvec2 <- c(TRUE, FALSE)

thevalues2 <- thevalues[selectionvec2]
thevalues2
```

```
##  reqs test1 maint
##    15    30    22
```

```
### Generating regular sequences with  `:` and `seq`

aseqofvalues <- 1:20

aseqofvalues2 <- seq(from=-3, to=3, by=0.5 )
aseqofvalues2
```

```
##  [1] -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0
```

```
aseqofvalues3 <- seq(0, 100, by=10)
aseqofvalues4 <- aseqofvalues3[c(2, 4, 6, 8)]
aseqofvalues4
```

```
## [1] 10 30 50 70
```

```
aseqofvalues4 <- aseqofvalues3[-c(2, 4, 6, 8)]
aseqofvalues4
```

```
## [1]   0  20  40  60  80  90 100
```

```
aseqofvalues3[c(1,2)] <- c(666,888)
aseqofvalues3
```

```
##  [1] 666 888  20  30  40  50  60  70  80  90 100
```

```
### Logical values in vectors TRUE/FALSE

aseqofvalues3 > 50
```

```
##  [1]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
aseqofvalues5 <- aseqofvalues3[aseqofvalues3 > 50]
aseqofvalues5
```

```
## [1] 666 888  60  70  80  90 100
```

```
aseqofvalues6 <- aseqofvalues3[!(aseqofvalues3 > 50)]
aseqofvalues6
```

```
## [1] 20 30 40 50
```

```
### Comparison functions

aseqofvalues7 <- aseqofvalues3[aseqofvalues3 == 50]
aseqofvalues7
```

```
## [1] 50
```

```
aseqofvalues8 <- aseqofvalues3[aseqofvalues3 == 22]
aseqofvalues8
```

```
## numeric(0)
```

```
aseqofvalues9 <- aseqofvalues3[aseqofvalues3 != 50]
aseqofvalues9
```

```
##  [1] 666 888  20  30  40  60  70  80  90 100
```

```
logicalcond <- aseqofvalues3 >= 50
aseqofvalues10 <- aseqofvalues3[logicalcond]
aseqofvalues10
```

```
## [1] 666 888  50  60  70  80  90 100
```

```
### Remove Missing Values (NAs)

aseqofvalues3[c(1,2)] <- c(NA,NA)
aseqofvalues3
```

```
##  [1]  NA  NA  20  30  40  50  60  70  80  90 100
```

```
aseqofvalues3 <- aseqofvalues3[!is.na(aseqofvalues3)]
aseqofvalues3
```

```
## [1]  20  30  40  50  60  70  80  90 100
```

1.5 Arrays and Matrices

Matrices are actually long vectors.

```
mymat <- matrix(1:12, nrow =2)
mymat
```

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

```
mymat <- matrix(1:12, ncol =3)
mymat
```

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

```
mymat <- matrix(1:12, nrow=2, byrow = TRUE)
mymat
```

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

```
mymat <- matrix(1:12, nrow=3, ncol=4)
mymat
```

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

```
mymat <- matrix(1:12, nrow=3, ncol=4, byrow=TRUE)
mymat
```

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

```
### recycling
mymat <- matrix(1:5, nrow=3, ncol=4, byrow=TRUE)
```

```
## Warning in matrix(1:5, nrow = 3, ncol = 4, byrow = TRUE): data length [5] is not
## a sub-multiple or multiple of the number of rows [3]
```

```
mymat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    1    2    3
## [3,]    4    5    1    2
```

```
### rbind  cbind

cbind(1:3, 1:3)
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    3    3
```

```
rbind(1:3, 1:3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    1    2    3
```

```
mymat <- matrix(1)

mymat <- matrix(1:8, nrow=2, ncol=4, byrow=TRUE)
mymat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
```

```
rbind(mymat, 9:12)
```

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

```
mymat <- cbind(mymat, c(5,9))
mymat
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    5    6    7    8    9
```

```
mymat  <- matrix(1:8, byrow = TRUE, nrow=2)
mymat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
```

```
rownames(mymat) <- c("row1", "row2")
mymat
```

```
##      [,1] [,2] [,3] [,4]
## row1    1    2    3    4
## row2    5    6    7    8
```

```
colnames(mymat) <- c("col1", "col2", "col3", "col4")
mymat
```

```
##      col1 col2 col3 col4
## row1    1    2    3    4
## row2    5    6    7    8
```

```
mymat2 <- matrix(1:12, byrow=TRUE, nrow=3, dimnames=list(c("row1", "row2", "row3"),
                                                          c("col1", "col2", "col3", "col4")))
mymat2
```

```
##      col1 col2 col3 col4
## row1    1    2    3    4
## row2    5    6    7    8
## row3    9   10   11   12
```

```
### Coercion in Arrays

matnum <- matrix(1:8, ncol = 2)
matnum
```

```
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    6
## [3,]    3    7
## [4,]    4    8
```

```
matchar <- matrix(LETTERS[1:6], nrow = 4, ncol = 3)
matchar
```

```
##      [,1] [,2] [,3]
## [1,] "A"  "E"  "C"
## [2,] "B"  "F"  "D"
## [3,] "C"  "A"  "E"
## [4,] "D"  "B"  "F"
```

```
matchars <- cbind(matnum, matchar)
matchars
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "1"  "5"  "A"  "E"  "C"
## [2,] "2"  "6"  "B"  "F"  "D"
## [3,] "3"  "7"  "C"  "A"  "E"
## [4,] "4"  "8"  "D"  "B"  "F"
```

```
### Subsetting

mymat3 <- matrix(sample(-8:15, 12), nrow=3)  #sample 12 numbers between -8 and 15
```

```
mymat3
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0    5   -3   12
## [2,]    6   15    4   -6
## [3,]    9   -4    3   10
```

```
mymat3[2,3]
```

```
## [1] 4
```

```
mymat3[1,4]
```

```
## [1] 12
```

```
mymat3[3,]
```

```
## [1]  9 -4  3 10
```

```
mymat3[,4]
```

```
## [1] 12 -6 10
```

```
mymat3[5] # counts elements by column
```

```
## [1] 15
```

```
mymat3[9]
```

```
## [1] 3
```

```
## Subsetting multiple elements

mymat3[2, c(1,3)]
```

```
## [1] 6 4
```

```
mymat3[c(2,3), c(1,3,4)]
```

```
##      [,1] [,2] [,3]
## [1,]    6    4   -6
## [2,]    9    3   10
```

```
rownames(mymat3) <- c("r1", "r2", "r3")
colnames(mymat3) <- c("c1", "c2", "c3", "c4")
mymat3["r2", c("c1", "c3")]
```

```
## c1 c3
##  6  4
```

```
### Subset by logical vector
mymat3[c(FALSE, TRUE, FALSE),
       c(TRUE, FALSE, TRUE, FALSE)]
```

```
## c1 c3
##  6  4
```

```
mymat3[c(FALSE, TRUE, TRUE),
       c(TRUE, FALSE, TRUE, TRUE)]
```

```
##    c1 c3 c4
## r2  6  4 -6
## r3  9  3 10
```

```
### matrix arithmetic

row1 <- c(220, 137)
row2 <- c(345, 987)
row3 <- c(111, 777)

mymat4 <- rbind(row1, row2, row3)
rownames(mymat4) <- c("row_1", "row_2", "row_3")
colnames(mymat4) <- c("col_1", "col_2")
mymat4
```

```
##       col_1 col_2
## row_1   220   137
## row_2   345   987
## row_3   111   777
```

```
mymat4/10
```

```
##       col_1 col_2
## row_1  22.0  13.7
## row_2  34.5  98.7
## row_3  11.1  77.7
```

```
mymat4 -100
```

```
##       col_1 col_2
## row_1   120    37
## row_2   245   887
## row_3    11   677
```

```
mymat5 <- rbind(c(50,50), c(10,10), c(100,100))
mymat5
```

```
##      [,1] [,2]
## [1,]   50   50
## [2,]   10   10
## [3,]  100  100
```

```
mymat4 - mymat5
```

```
##       col_1 col_2
## row_1   170    87
## row_2   335   977
## row_3    11   677
```

```
mymat4 * (mymat5/100)
```

```
##       col_1 col_2
## row_1 110.0  68.5
## row_2  34.5  98.7
## row_3 111.0 777.0
```

```
### index matrices

m1 <- array(1:20, dim=c(4,5))
m1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

```
index <- array(c(1:3, 3:1), dim=c(3,2))
index
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    2
## [3,]    3    1
```

```
#use the "index matrix" as the index for the other matrix
m1[index] <-0
m1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    0   13   17
## [2,]    2    0   10   14   18
## [3,]    0    7   11   15   19
## [4,]    4    8   12   16   20
```

1.6 Factors

- Factors are variables in R which take on a limited number of different values; such variables are often referred to as ‘categorical variables’ and 'enumerated type'.

- Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed.
- The function `factor` is used to encode a vector as a factor

```
personnel <- c("Analyst1", "ManagerL2", "Analyst1", "Analyst2",
               "Boss", "ManagerL1", "ManagerL2", "Programmer1",
               "Programmer2", "Programmer3", "Designer1","Designer2",
               "OtherStaff")  # staff in a company

personnel_factors <- factor(personnel)
personnel_factors  #sorted alphabetically
```

```
##  [1] Analyst1    ManagerL2   Analyst1    Analyst2    Boss        ManagerL1
##  [7] ManagerL2   Programmer1 Programmer2 Programmer3 Designer1   Designer2
## [13] OtherStaff
## 11 Levels: Analyst1 Analyst2 Boss Designer1 Designer2 ManagerL1 ... Programmer3
```

```
str(personnel_factors)
```

```
##  Factor w/ 11 levels "Analyst1","Analyst2",..: 1 7 1 2 3 6 7 9 10 11 ...
```

```
personnel2 <- factor(personnel,
                     levels = c("Boss", "ManagerL1", "ManagerL2",
                                "Analyst1", "Analyst2",  "Designer1",
                                "Designer2", "Programmer1", "Programmer2",
                                "Programmer3", "OtherStaff"))
                                #do not duplicate the same factors
personnel2
```

```
##  [1] Analyst1    ManagerL2   Analyst1    Analyst2    Boss        ManagerL1
##  [7] ManagerL2   Programmer1 Programmer2 Programmer3 Designer1   Designer2
## [13] OtherStaff
## 11 Levels: Boss ManagerL1 ManagerL2 Analyst1 Analyst2 Designer1 ... OtherStaff
```

```
str(personnel2)
```

```
##  Factor w/ 11 levels "Boss","ManagerL1",..: 4 3 4 5 1 2 3 8 9 10 ...
```

```
# a factor's levels will always be character values.

levels(personnel2) <- c("B", "M1", "M2", "A1", "A2",
                        "D1", "D2", "P1", "P2", "P3", "OS")
personnel2
```

```
##  [1] A1 M2 A1 A2 B  M1 M2 P1 P2 P3 D1 D2 OS
## Levels: B M1 M2 A1 A2 D1 D2 P1 P2 P3 OS
```

```
personnel3 <- factor(personnel,
                     levels = c("Boss", "ManagerL1", "ManagerL2",
                                "Analyst1", "Analyst2",  "Designer1",
                                "Designer2", "Programmer1", "Programmer2",
                                "Programmer3", "OtherStaff"),
                     c("B", "M1", "M2", "A1", "A2", "D1", "D2",
                       "P1", "P2", "P3", "OS"))
personnel3
```

```
##  [1] A1 M2 A1 A2 B  M1 M2 P1 P2 P3 D1 D2 OS
## Levels: B M1 M2 A1 A2 D1 D2 P1 P2 P3 OS
```

```
### Nominal versus ordinal, ordered factors
personnel3[1] < personnel3[2]  # error, factors not ordered
```

```
## Warning in Ops.factor(personnel3[1], personnel3[2]): '<' not meaningful for
## factors
```

```
## [1] NA
```

```
tshirts <- c("M", "L", "S", "S", "L", "M", "L", "M")

tshirt_factor <- factor(tshirts, ordered = TRUE,
                        levels = c("S", "M", "L"))
tshirt_factor
```

```
## [1] M L S S L M L M
## Levels: S < M < L
```

```
tshirt_factor[1] < tshirt_factor[2]
```

```
## [1] TRUE
```

1.7 Lists

Lists are the R objects which contain elements of different types: numbers, strings, vectors and other lists. A list can also contain a matrix or a function as one of their elements. A list is created using `list()` function.

Operators for subsetting lists include: - ‘[’ returns a list - ’[[’ returns the list element - ‘$’ returns the content of that element in the list]<-1

cbo <- jdt$cbo bugs <- jdt$bugs

Chapter 2

Split data into training and test datasets

jdt2 = data.frame(cbo, bugs) inTrain <- createDataPartition(y=jdt2$bugs,p=.8,list=FALSE)
jdtTrain <- jdt2[inTrain,] jdtTest <- jdt2[-inTrain,]

~~~
BLR models fault-proneness are as follows $fp(X) = \frac{e^{logit()}}{1 + e^{logit(X)}}$

where the simplest form for logit is $logit(X) = c_{0} + c_{1}X$


```r
# logit regression
# glmLogit <- train (bugs ~ ., data=jdt.train, method="glm", family=binomial(link = logit))

glmLogit <- glm (bugs ~ ., data=jdtTrain, family=binomial(link = logit))
summary(glmLogit)
~~~

```
##
## Call:
## glm(formula = bugs ~ ., family = binomial(link = logit), data = jdtTrain)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.654  -0.591  -0.515  -0.471   2.150
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.20649    0.13900  -15.87   <2e-16 ***
## cbo          0.06298    0.00765    8.23   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 807.98  on 797  degrees of freedom
## Residual deviance: 691.80  on 796  degrees of freedom
## AIC: 695.8
##
## Number of Fisher Scoring iterations: 5
```

Predict a single point:

```
newData = data.frame(cbo = 3)
predict(glmLogit, newData, type = "response")
```

```
##     1
## 0.117
```

Draw the results, modified from: http://www.shizukalab.com/toolkits/plotting-logistic-regression-in-r

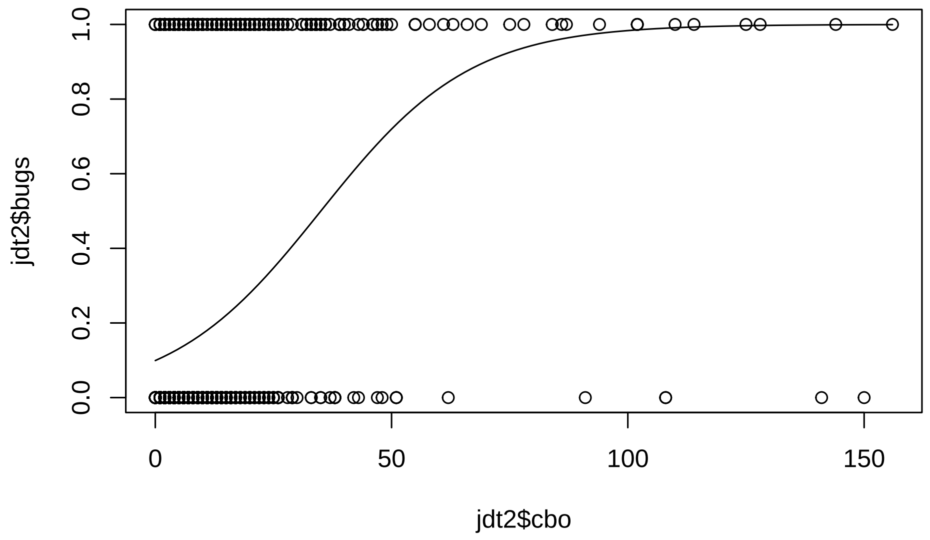
```
results <- predict(glmLogit, jdtTest, type = "response")

range(jdtTrain$cbo)
```

```
## [1]   0 156
```

```
range(results)
```

```
## [1] 0.0992 0.9993
```

```
plot(jdt2$cbo,jdt2$bugs)
curve(predict(glmLogit, data.frame(cbo=x), type = "response"),add=TRUE)
```



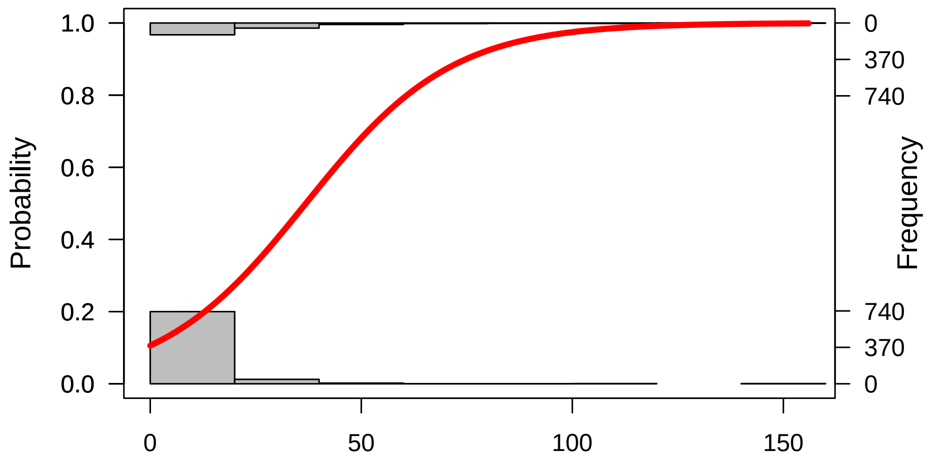
```r
# points(jdtTrain$cbo,fitted(glmLogit))
```

Another type of graph:

```r
library(popbio)
```

```
##
## Attaching package: 'popbio'

## The following object is masked from 'package:caret':
##
##     sensitivity
```

```r
logi.hist.plot(jdt2$cbo,jdt2$bugs,boxp=FALSE,type="hist",col="gray")
```



2.1 The caret package

There are hundreds of packages to perform classification task in R, but many of those can be used throught the ‘caret’ package which helps with many of the data mining process task as described next.

The caret packagehttp://topepo.github.io/caret/ provides a unified interface for modeling and prediction with around 150 different models with tools for:

- data splitting
- pre-processing
- feature selection
- model tuning using resampling
- variable importance estimation, etc.

Website: http://caret.r-forge.r-project.org

JSS Paper: www.jstatsoft.org/v28/i05/paper

Book: Applied Predictive Modeling

Chapter 3

Regression

3.1 Linear Regression modeling

- *Linear Regression* is one of the oldest and most known predictive methods. As its name says, the idea is to try to fit a linear equation between a dependent variable and an independent, or explanatory, variable. The idea is that the independent variable x is something the experimenter controls and the dependent variable y is something that the experimenter measures. The line is used to predict the value of y for a known value of x. The variable x is the predictor variable and y the response variable.

- *Multiple linear regression* uses 2 or more independent variables for building a model. See https://www.wikipedia.org/wiki/Linear_regression.

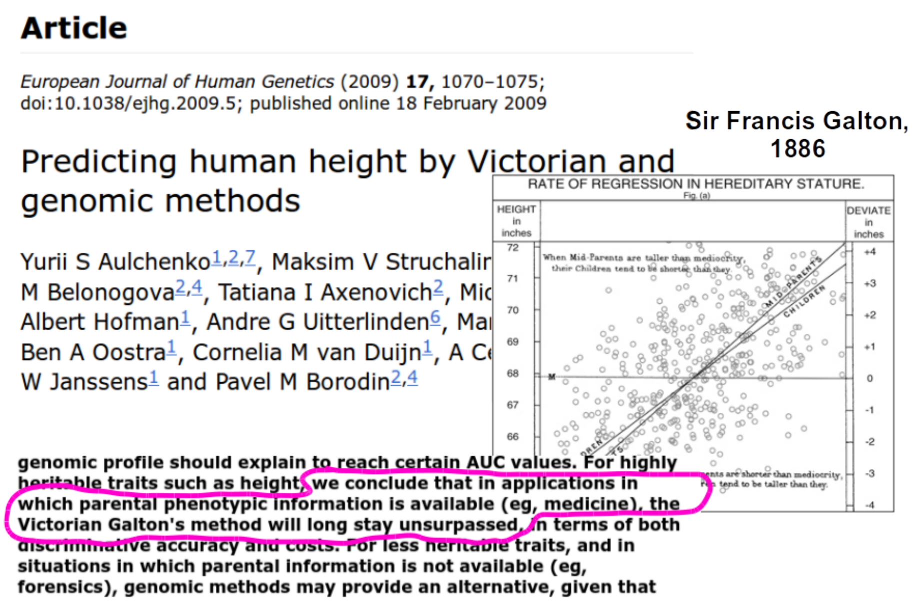
- First proposed many years ago but still very useful…

- The equation takes the form $\hat{y} = b_0 + b_1 * x$
- The method used to choose the values b_0 and b_1 is to minimize the sum of the squares of the residual errors.

3.1.1 Regression: Galton Data

Not related to Software Engineering but …

```
library(UsingR)
data(galton)
par(mfrow=c(1,2))
hist(galton$child,col="blue",breaks=100)
hist(galton$parent,col="blue",breaks=100)
```


Article

European Journal of Human Genetics (2009) **17,** 1070–1075;
doi:10.1038/ejhg.2009.5; published online 18 February 2009

Predicting human height by Victorian and genomic methods

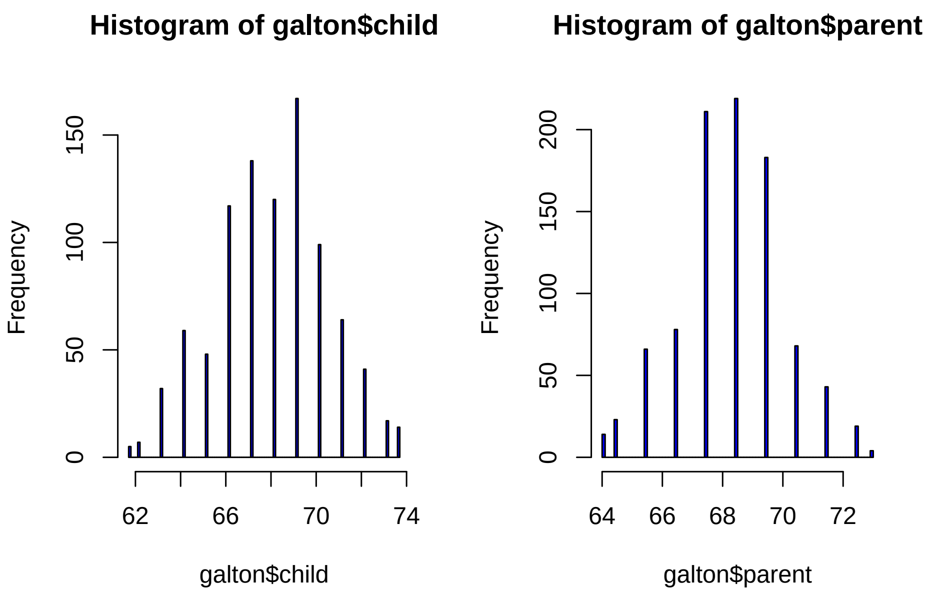
Yurii S Aulchenko[1,2,7], Maksim V Struchalir
M Belonogova[2,4], Tatiana I Axenovich[2], Mic
Albert Hofman[1], Andre G Uitterlinden[6], Mai
Ben A Oostra[1], Cornelia M van Duijn[1], A Ce
W Janssens[1] and Pavel M Borodin[2,4]

genomic profile should explain to reach certain AUC values. For highly heritable traits such as height, we conclude that in applications in which parental phenotypic information is available (eg, medicine), the Victorian Galton's method will long stay unsurpassed, in terms of both discriminative accuracy and costs. For less heritable traits, and in situations in which parental information is not available (eg, forensics), genomic methods may provide an alternative, given that

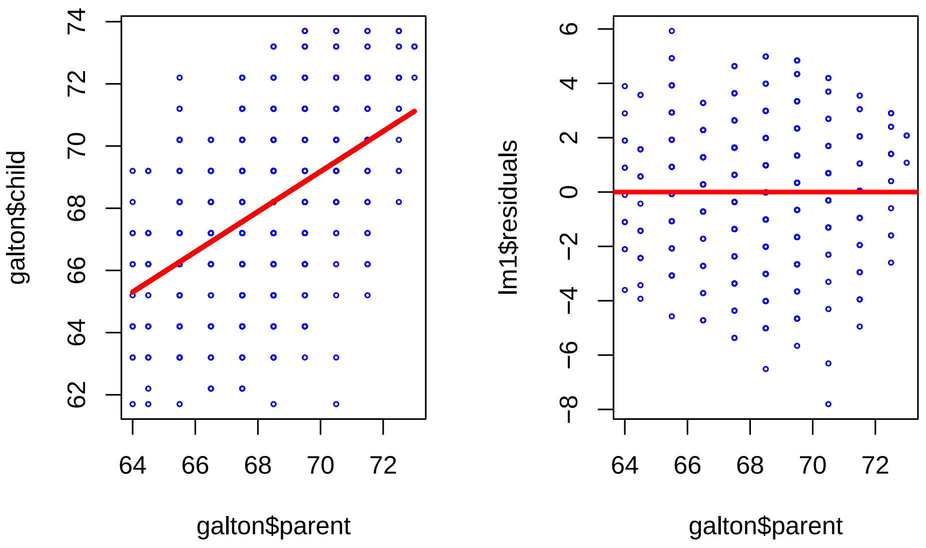
Sir Francis Galton, 1886

RATE OF REGRESSION IN HEREDITARY STATURE.
Fig. (a)
HEIGHT in inches
DEVIATE in inches
72
71
70
69
68
67
66
+4
+3
+2
+1
0
-1
-2
-3
-4
M
When Mid-Parents are taller than mediocrity, their Children tend to be shorter than they.
MID-PARENTS
CHILDREN
...ents are shorter than mediocrity, ...n tend to be taller than they.

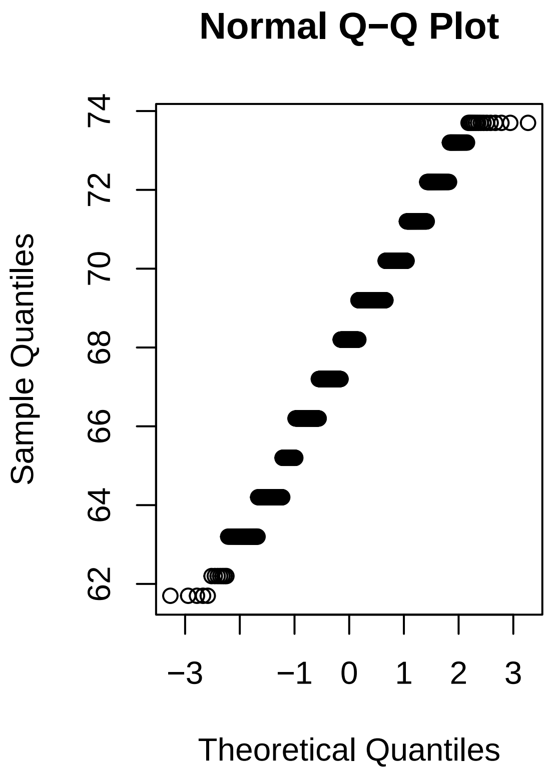
Figure 3.1: Galton Data



```
plot(galton$parent,galton$child,pch=1,col="blue", cex=0.4)
lm1 <- lm(galton$child ~ galton$parent)
lines(galton$parent,lm1$fitted,col="red",lwd=3)
plot(galton$parent,lm1$residuals,col="blue",pch=1, cex=0.4)
abline(c(0,0),col="red",lwd=3)
```



```
qqnorm(galton$child)
```



3.1.2 Simple Linear Regression

- Given two variables Y (response) and X (predictor), the assumption is that there is an approximate ($\approx$) *linear* relation between those variables.
- The mathematical model of the observed data is described as (for the case of simple linear regression):

$$Y \approx \beta_0 + \beta_1 X$$

- the parameter β_0 is named the *intercept* and β_1 is the *slope*
- Each observation can be modeled as

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i; \epsilon_i \sim N(0, \sigma^2)$$

- ϵ_i is the *error* - This means that the variable y is normally distributed:

$$y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$$

- The *predictions* or *estimations* of this model are obtained by a linear equation of the form $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$, that is, each new prediction is computed with

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

.
- The actual parameters β_0 and β_1 are unknown
- The parameters $\hat{\beta}_0$ and $\hat{\beta}_1$ of the linear equation can be estimated with different methods.

3.1.3 Least Squares

- One of the most used methods for computing $\hat{\beta}_0$ and $\hat{\beta}_1$ is the criterion of "least squares" minimization.
- The data is composed of n pairs of observations (x_i, y_i)
- Given an observation y_i and its corresponding estimation $\hat{y_i}$) the *residual* e_i is defined as

$$e_i = y_i - \hat{y_i}$$

- the Residual Sum of Squares is defined as

$$RSS = e_1^2 + \dots + e_i^2 + \dots + e_n^2$$

- the Least Squares Approach minimizes the RSS
- as result of that minimizitation, it can be obtained, by means of calculus, the estimation of $\hat{\beta}_0$ and $\hat{\beta}_1$ as

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

and

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where $\bar{y}$ and $\bar{x}$ are the sample means.

- the variance σ^2 is estimated by

$$\hat{\sigma}^2 = RSS/(n-2)$$

where n is the number of observations
- The *Residual Standard Error* is defined as

$$RSE = \sqrt{RSS/(n-2)}$$

- The equation

$$Y = \beta_0 + \beta_1 X + \epsilon$$

defines the linear model, i.e., the *population regression line*
- The *least squares line* is $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$
- *Confidence intervals* are computed using the *standard errors* of the intercept and the slope.
- The 95% confidence interval for the slope is computed as

$$[\hat{\beta}_1 - 2 \cdot SE(\hat{\beta}_1), \hat{\beta}_1 + SE(\hat{\beta}_1)]$$

- where

$$SE(\hat{\beta}_1) = \sqrt{\frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}}$$

3.1.4 Linear regression in R

The following are the basic commands in R:

- The basic function is `lm()`, that returns an object with the model.
- Other commands: `summary` prints out information about the regression, `coef` gives the coefficients for the linear model, `fitted` gives the predictd value of y for each value of x, `residuals` contains the differences between observed and fitted values.
- `predict` will generate predicted values of the response for the values of the explanatory variable.

3.2 Linear Regression Diagnostics

- Several plots help to evaluate the suitability of the linear regression
  - *Residuals vs fitted*: The residuals should be randomly distributed around the horizontal line representing a residual error of zero; that is, there should not be a distinct trend in the distribution of points.
  - *Standard Q-Q plot*: residual errors are normally distributed

  – *Square root of the standardized residuals vs the fitted values*: there should be no obvious trend. This plot is similar to the residuals versus fitted values plot, but it uses the square root of the standardized residuals.
  – *Leverage*: measures the importance of each point in determining the regression result. Smaller values means that removing the observation has little effect on the regression result.

3.2.1 Simulation example

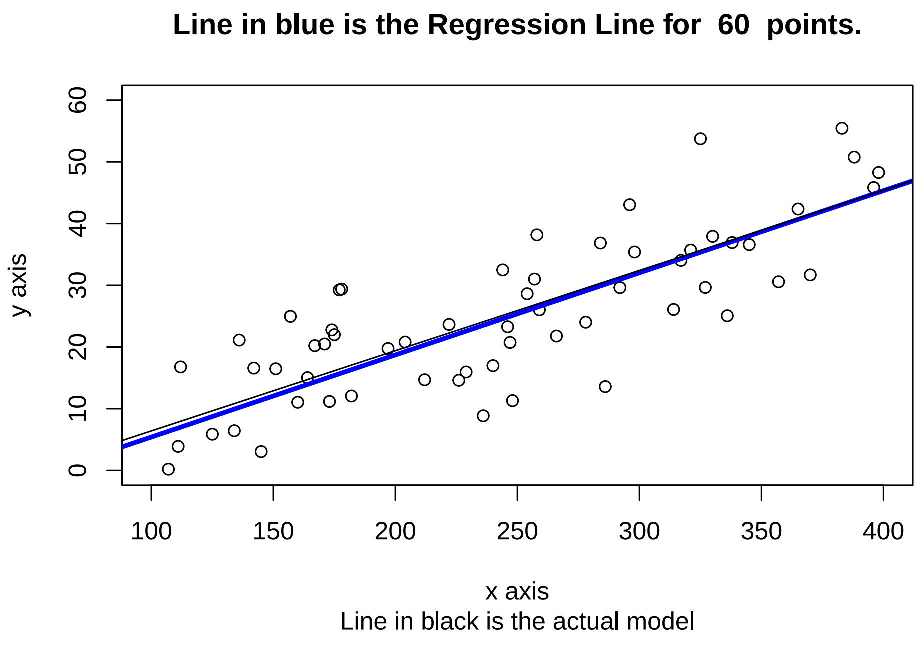
3.2.1.1 Simulate a dataset

```
set.seed(3456)
# equation is  y = -6.6 + 0.13 x +e
# range x 100,400
a <- -6.6
b <- 0.13
num_obs <- 60
xmin <- 100
xmax <- 400
x <- sample(seq(from=xmin, to = xmax, by =1), size= num_obs, replace=FALSE)

sderror <- 9 # sigma for the error term in the model
e <- rnorm(num_obs, 0, sderror)

y <- a + b * x + e


newlm <- lm(y~x)
summary(newlm)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -26.518  -5.645   0.363   5.695  18.392
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -7.9060     3.3922   -2.33    0.023 *
## x             0.1331     0.0132   10.05  2.6e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 8.48 on 58 degrees of freedom
## Multiple R-squared:  0.635,  Adjusted R-squared:  0.629
## F-statistic:  101 on 1 and 58 DF,  p-value: 2.57e-14
```

```
cfa1 <- coef(newlm)[1]
cfb2 <- coef(newlm)[2]
plot(x,y, xlab="x axis", ylab= "y axis", xlim = c(xmin, xmax), ylim = c(0,60), sub = "Line in bla
title(main = paste("Line in blue is the Regression Line for ", num_obs, " points."))

abline(a = cfa1, b = cfb2, col= "blue", lwd=3)
abline(a = a, b = b, col= "black", lwd=1) #original line
```



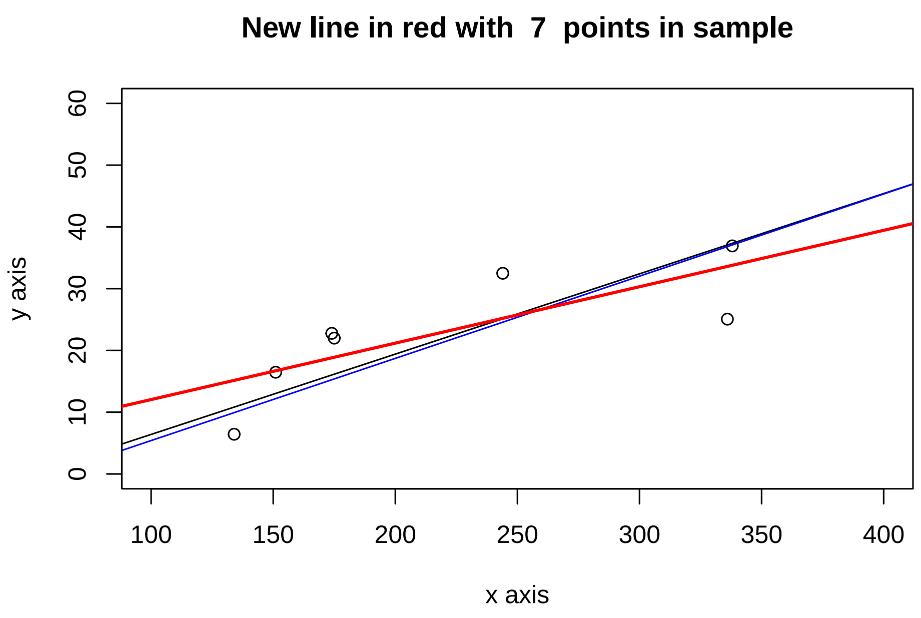
```
# sample from  the same  x     to compare least squares lines
# change the denominator in newsample to see how the least square lines changes accordingly.
newsample <- as.integer(num_obs/8) # number of pairs x,y

idxs_x1 <- sample(1:num_obs, size = newsample, replace = FALSE) #sample indexes
x1 <- x[idxs_x1]
e1 <- e[idxs_x1]
y1 <- a + b * x1 + e1
xy_obs <- data.frame(x1, y1)
names(xy_obs) <- c("x_obs", "y_obs")

newlm1 <- lm(y1~x1)
```

```
summary(newlm1)
```

3.2.1.1.1 Subset a set of points from the same sample

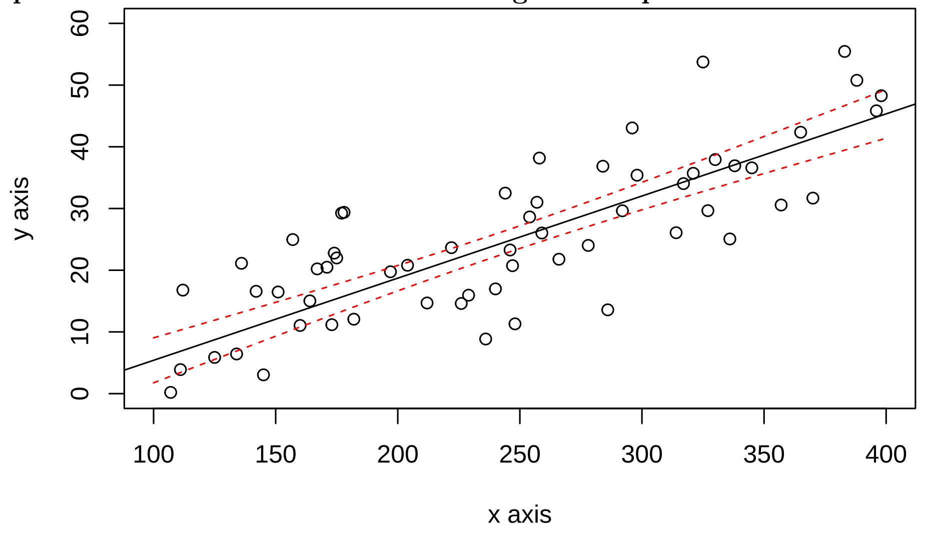
```
##
## Call:
## lm(formula = y1 ~ x1)
##
## Residuals:
##      1      2      3      4      5      6      7
##  3.968 -8.537  3.141 -8.723  7.294 -0.235  3.092
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.9107     7.7166    0.38    0.722
## x1            0.0913     0.0328    2.79    0.039 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.89 on 5 degrees of freedom
## Multiple R-squared:  0.609,  Adjusted R-squared:  0.53
## F-statistic: 7.77 on 1 and 5 DF,  p-value: 0.0385
```

```
cfa21 <- coef(newlm1)[1]
cfb22 <- coef(newlm1)[2]

plot(x1,y1, xlab="x axis", ylab= "y axis", xlim = c(xmin, xmax), ylim = c(0,60))
title(main = paste("New line in red with ", newsample, " points in sample"))

abline(a = a, b = b, col= "black", lwd=1)  # True line
abline(a = cfa1, b = cfb2, col= "blue", lwd=1)  #sample
abline(a = cfa21, b = cfb22, col= "red", lwd=2) #new line
```

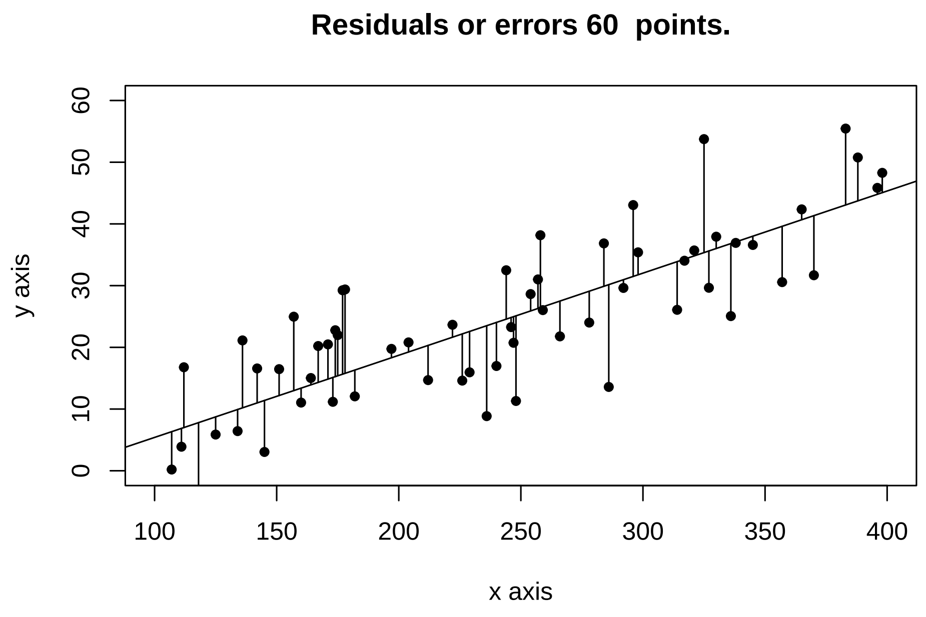


```
newx <- seq(xmin, xmax)
ypredicted <- predict(newlm, newdata=data.frame(x=newx), interval= "confidence", level= 0.90, se

plot(x,y, xlab="x axis", ylab= "y axis", xlim = c(xmin, xmax), ylim = c(0,60))
# points(x1, fitted(newlm1))
abline(newlm)

lines(newx,ypredicted$fit[,2],col="red",lty=2)
lines(newx,ypredicted$fit[,3],col="red",lty=2)
```

3.2.1.1.2 Compute a confidence interval on the original sample regression line



```
# Plot the residuals or errors
ypredicted_x <- predict(newlm, newdata=data.frame(x=x))
plot(x,y, xlab="x axis", ylab= "y axis", xlim = c(xmin, xmax), ylim = c(0,60), sub = "
title(main = paste("Residuals or errors", num_obs, " points."))
abline(newlm)
segments(x, y, x, ypredicted_x)
```



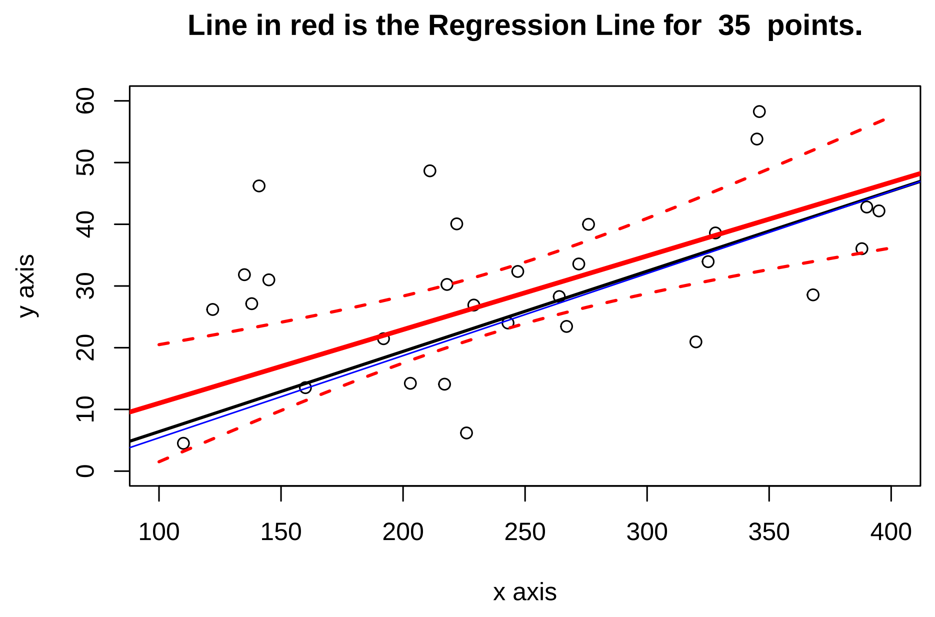
```
# equation is  y = -6.6 + 0.13 x +e
# range x 100,400
num_obs <- 35
xmin <- 100
xmax <- 400
x3 <- sample(seq(from=xmin, to = xmax, by =1), size= num_obs, replace=FALSE)
sderror <- 14 # sigma for the error term in the model
e3 <- rnorm(num_obs, 0, sderror)

y3 <- a + b * x3 + e3

newlm3 <- lm(y3~x3)
summary(newlm3)
```

3.2.1.1.3 Take another sample from the model and explore

```
##
## Call:
## lm(formula = y3 ~ x3)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -40.87  -9.20  -2.28  12.08  47.17
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.9284     8.7458   -0.11   0.9161
## x3            0.1193     0.0345    3.45   0.0015 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.2 on 33 degrees of freedom
## Multiple R-squared:  0.266,  Adjusted R-squared:  0.243
## F-statistic: 11.9 on 1 and 33 DF,  p-value: 0.00153
```

```
cfa31 <- coef(newlm3)[1]
cfb32 <- coef(newlm3)[2]
plot(x3,y3, xlab="x axis", ylab= "y axis", xlim = c(xmin, xmax), ylim = c(0,60))
title(main = paste("Line in red is the Regression Line for ", num_obs, " points."))
abline(a = cfa31, b = cfb32, col= "red", lwd=3)
abline(a = a, b = b, col= "black", lwd=2) #original line
abline(a = cfa1, b = cfb2, col= "blue", lwd=1) #first sample

# confidence intervals for the new sample
```

```
newx <- seq(xmin, xmax)
ypredicted <- predict(newlm3, newdata=data.frame(x3=newx), interval= "confidence", lev

lines(newx,ypredicted$fit[,2],col="red",lty=2, lwd=2)
lines(newx,ypredicted$fit[,3],col="red",lty=2, lwd=2)
```



3.2.2 Diagnostics fro assessing the regression line

3.2.2.1 Residual Standard Error

- It gives us an idea of the typical or average error of the model. It is the estimated standard deviation of the residuals.

3.2.2.2 R^2 statistic

- This is the proportion of variability in the data that is explained by the model. Best values are those close to 1.

3.3 Multiple Linear Regression

3.3.1 Partial Least Squares

- If several predictors are highly correlated, the least squares approach has high variability.
- PLS finds linear combinations of the predictors, that are called *components* or *latent* variables.

3.4 Linear regression in Software Effort estimation

Fitting a linear model to log-log - the predictive power equation is $y = e^{b_0} * x^{b_1}$, ignoring the bias corrections. Note: depending how the error term behaves we could try another general linear model (GLM) or other model that does not rely on the normality of the residuals (quantile regression, etc.) - First, we are fitting the model to the whole dataset. But it is not the right way to do it, because of overfitting.

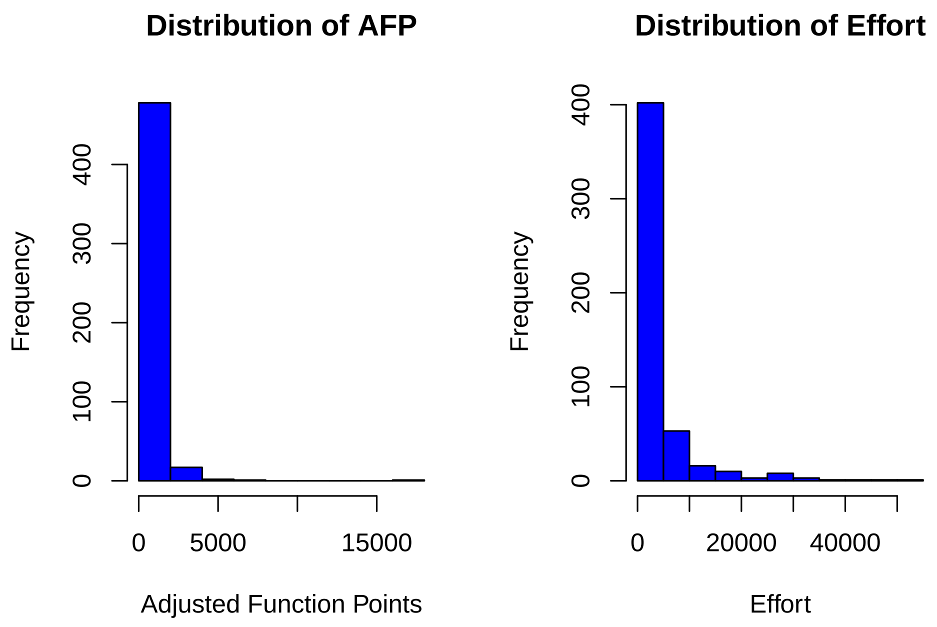
```
library(foreign)
china <- read.arff("./datasets/effortEstimation/china.arff")
china_size <- china$AFP
summary(china_size)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       9     100     215     487     438   17518
```

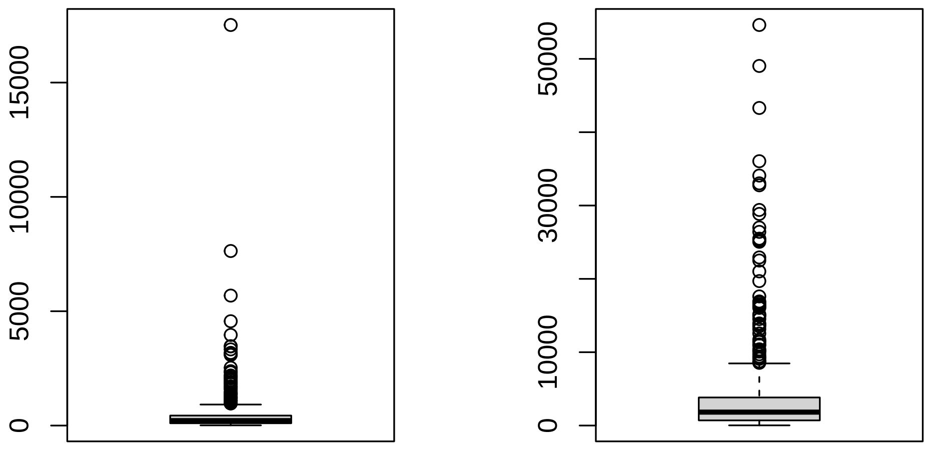
```
china_effort <- china$Effort
summary(china_effort)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      26     704    1829    3921    3826   54620
```

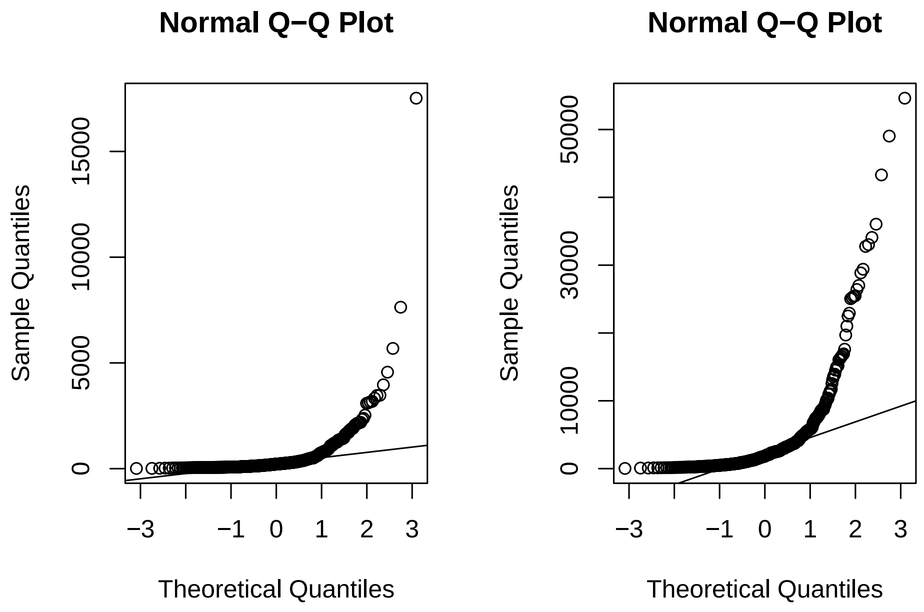
```
par(mfrow=c(1,2))
hist(china_size, col="blue", xlab="Adjusted Function Points", main="Distribution of AFP")
hist(china_effort, col="blue",xlab="Effort", main="Distribution of Effort")
```



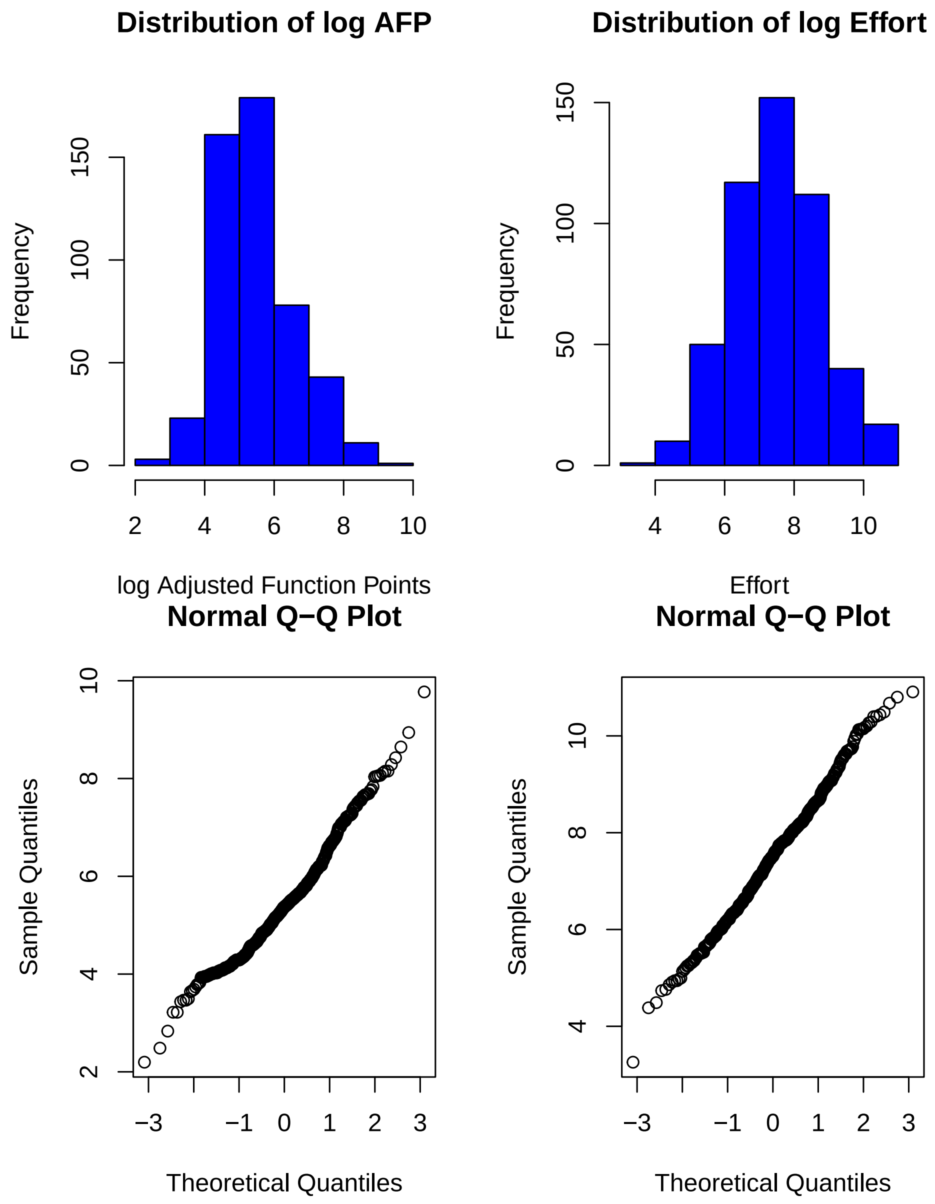
```
boxplot(china_size)
boxplot(china_effort)
```



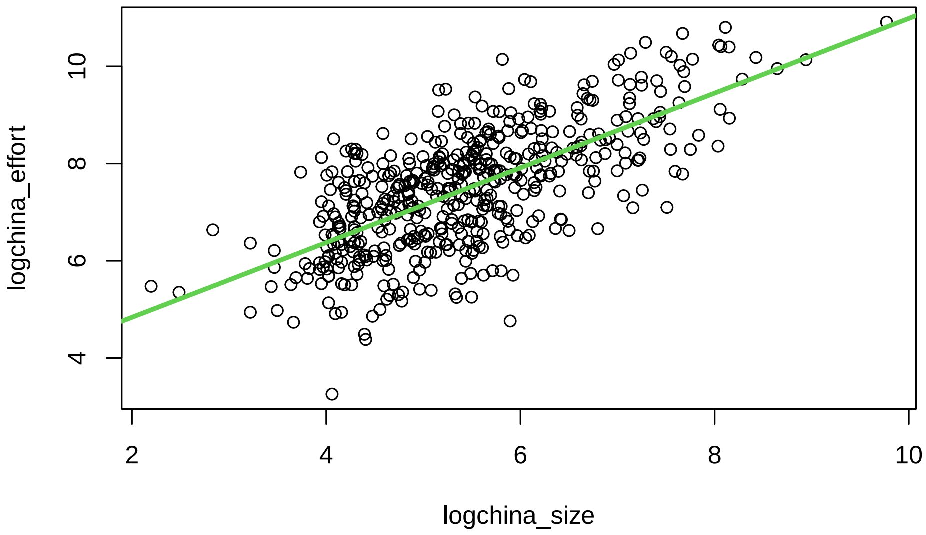
```
qqnorm(china_size)
qqline(china_size)
qqnorm(china_effort)
qqline(china_effort)
```



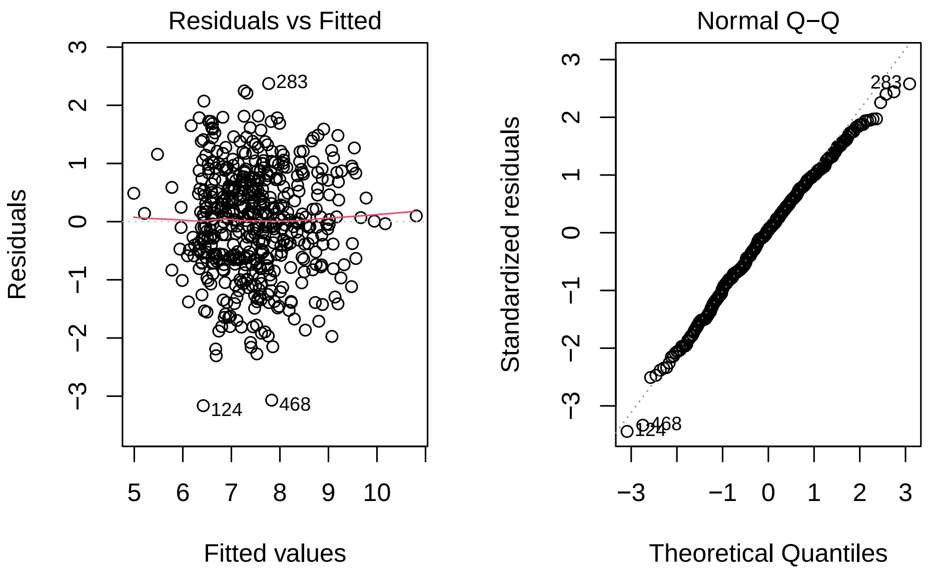
Applying the `log` function (it computes natural logarithms, base e)

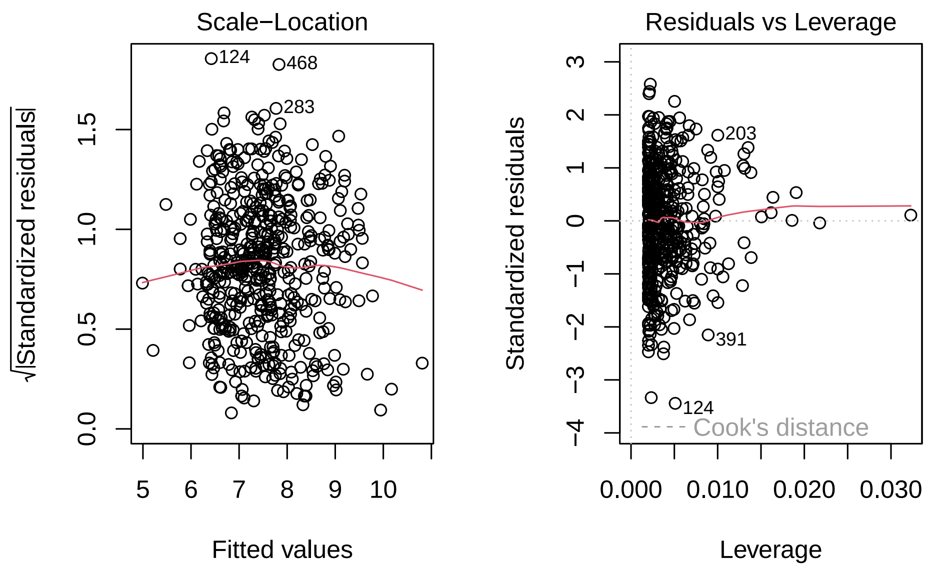


```
linmodel_logchina <- lm(logchina_effort ~ logchina_size)
par(mfrow=c(1,1))
plot(logchina_size, logchina_effort)
abline(linmodel_logchina, lwd=3, col=3)
```



```
par(mfrow=c(1,2))
plot(linmodel_logchina, ask = FALSE)
```





```
linmodel_logchina
```

```
##
## Call:
## lm(formula = logchina_effort ~ logchina_size)
##
## Coefficients:
##   (Intercept)  logchina_size
##         3.301          0.768
```

3.5 References

- The New Statistics with R, Andy Hector, 2015
- An Introduction to R, W.N. Venables and D.M. Smith and the R Development Core Team
- Practical Data Science with R, Nina Zumel and John Mount
- G. James et al, An Introduction to Statistical Learning with Applications in R, Springer, 2013

Part II

Unsupervised Models

Chapter 4

Unsupervised or Descriptive modeling

From the descriptive (unsupervised) point of view, patterns are found to predict future behaviour or estimate. This include association rules, clustering, or tree clustering which aim at grouping together objects (e.g., animals) into successively larger clusters, using some measure of similarity or distance. The dataset will be as the previous table without the C class attribute

Att_1		Att_n
a_{11}	...	a_{1n}
a_{21}	...	a_{2n}
...	...	...
a_{m1}	...	a_{mn}

4.1 Clustering

```
library(foreign)
library(fpc)

kc1 <- read.arff("./datasets/defectPred/D1/KC1.arff")

# Split into training and test datasets
set.seed(1)
ind <- sample(2, nrow(kc1), replace = TRUE, prob = c(0.7, 0.3))
kc1.train <- kc1[ind==1, ]
kc1.test <- kc1[ind==2, ]
```

```
# No class
kc1.train$Defective <- NULL

ds <- dbscan(kc1.train, eps = 0.42, MinPts = 5)

kc1.kmeans <- kmeans(kc1.train, 2)
```

4.1.1 k-Means

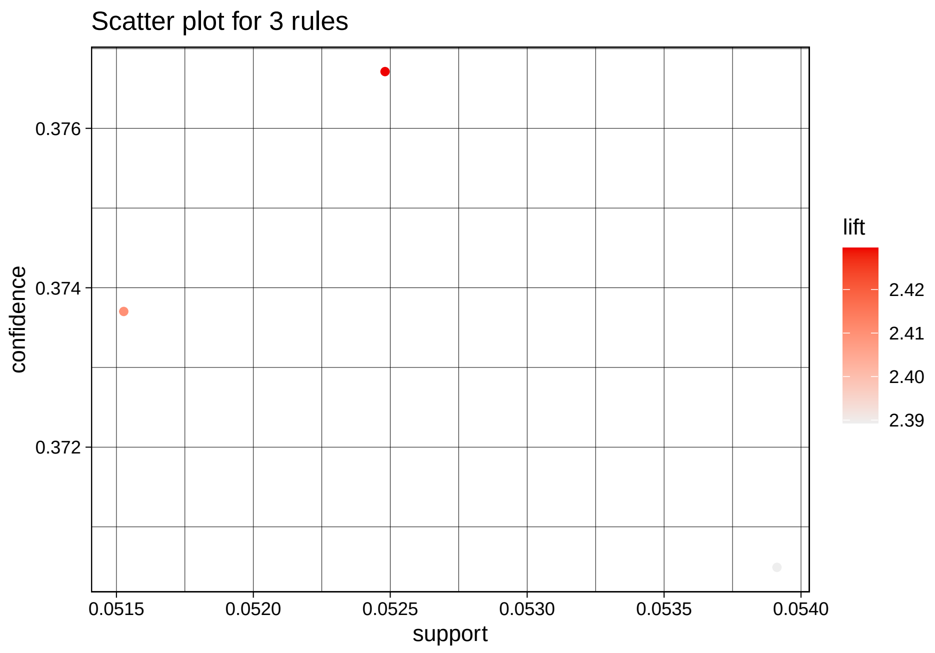
```
library(reshape, quietly=TRUE)
library(graphics)
kc1kmeans <- kmeans(sapply(na.omit(kc1.train), rescaler, "range"), 10)
#plot(kc1kmeans, col = kc1kmeans$cluster)
#points(kc1kmeans$centers, col = 1:5, pch = 8)
```

4.2 Association rules

```
library(arules)

# x <- as.numeric(kc1$LOC_TOTAL)
# str(x)
# summary(x)
# hist(x, breaks=30, main="LoC Total")
# xDisc <- discretize(x, categories=5)
# table(xDisc)

for(i in 1:21) kc1[,i] <- discretize(kc1[,i],  method = "interval", breaks = 5)

rules <- apriori(kc1,
   parameter = list(minlen=3, supp=0.05, conf=0.35),
   appearance = list(rhs=c("Defective=Y"),
   default="lhs"),
   control = list(verbose=F))

#rules <- apriori(kc1,
 #   parameter = list(minlen=2, supp=0.05, conf=0.3),
 #   appearance = list(rhs=c("Defective=Y", "Defective=N"),
 #   default="lhs"))

inspect(rules)
```

```
##     lhs                                   rhs           support confidence coverage li
## [1] {HALSTEAD_CONTENT=[38.6,77.2),
##      HALSTEAD_LEVEL=[0,0.4)}         => {Defective=Y}  0.0539     0.370    0.146 2.39   113
## [2] {LOC_CODE_AND_COMMENT=[0,2.4),
##      HALSTEAD_CONTENT=[38.6,77.2)}  => {Defective=Y}  0.0525     0.377    0.139 2.43   110
## [3] {LOC_CODE_AND_COMMENT=[0,2.4),
##      HALSTEAD_CONTENT=[38.6,77.2),
##      HALSTEAD_LEVEL=[0,0.4)}         => {Defective=Y}  0.0515     0.374    0.138 2.41   108
```

```
library(arulesViz)
plot(rules)
```



Part III

Evaluation

Chapter 5

Evaluation of Models

Once we obtain the model with the training data, we need to evaluate it with some new data (testing data).

> **No Free Lunch theorem** In the absence of any knowledge about the prediction problem, no model can be said to be uniformly better than any other

5.1 Building and Validating a Model

We cannot use the the same data for training and testing (it is like evaluating a student with the exercises previously solved in class, the student's marks will be "optimistic" and we do not know about student capability to generalise the learned concepts).

Therefore, we should, at a minimum, divide the dataset into *training* and *testing*, learn the model with the training data and test it with the rest of data as explained next.

5.1.1 Holdout approach

Holdout approach consists of dividing the dataset into *training* (typically approx. 2/3 of the data) and *testing* (approx 1/3 of the data). + Problems: Data can be skewed, missing classes, etc. if randomly divided. Stratification ensures that each class is represented with approximately equal proportions (e.g., if data contains approximately 45% of positive cases, the training and testing datasets should maintain similar proportion of positive cases).

Holdout estimate can be made more reliable by repeating the process with different subsamples (repeated holdout method).

The error rates on the different iterations are averaged (overall error rate).

- Usually, part of the data points are used for building the model and the remaining points are used for validating the model. There are several approaches to this process.
- *Validation Set approach*: it is the simplest method. It consists of randomly dividing the available set of observations into two parts, a *training set* and a *validation set* or hold-out set. Usually 2/3 of the data points are used for training and 1/3 is used for testing purposes.

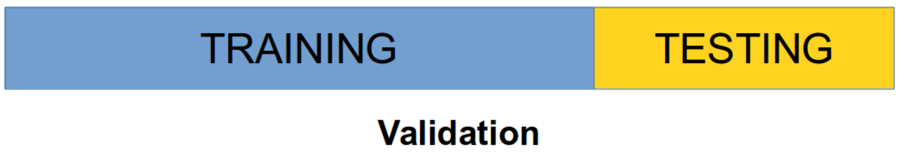


Figure 5.1: Hold out validation

5.1.2 Cross Validation (CV)

k-fold Cross-Validation involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. One fold is treated as a validation set and the method is trained on the remaining $k-1$ folds. This procedure is repeated k times. If k is equal to n we are in the previous method.

- 1st step: split dataset (D) into k subsets of approximately equal size $C_1, \dots, C_k$
- 2nd step: we construct a dataset $D_i = D - C_i$ used for training and test the accuracy of the classifier D_i on C_i subset for testing

Having done this for all k we estimate the accuracy of the method by averaging the accuracy over the k cross-validation trials

5.1.3 Leave-One-Out Cross-Validation (LOO-CV)

- *Leave-One-Out Cross-Validation* (LOO-CV): This is a special case of CV. Instead of creating two subsets for training and testing, a single observation is used for the validation set, and the remaining observations make up the training set. This approach is repeated n times (the total number of observations) and the estimate for the test mean squared error is the average of the n test estimates.

5.2 Evaluation of Classification Models

The confusion matrix (which can be extended to multiclass problems) is a table that presents the results of a classification algorithm. The following table shows the possible outcomes for binary classification problems:

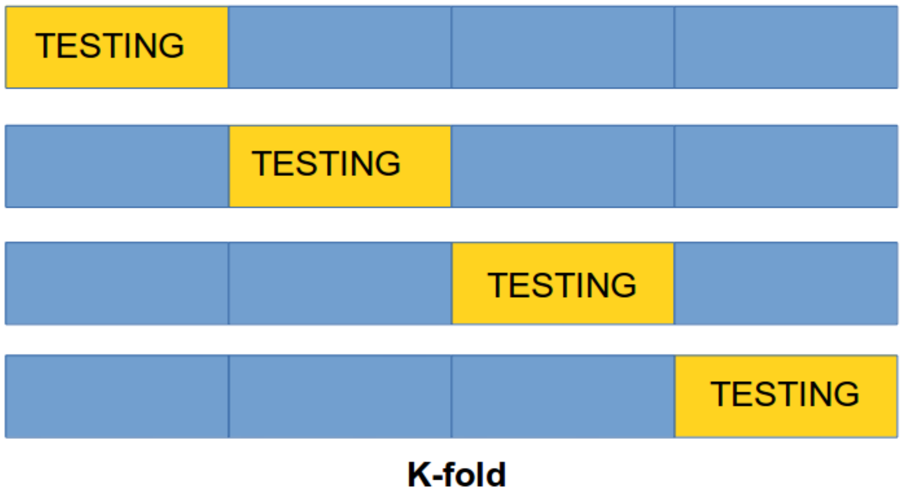


Figure 5.2: k-fold

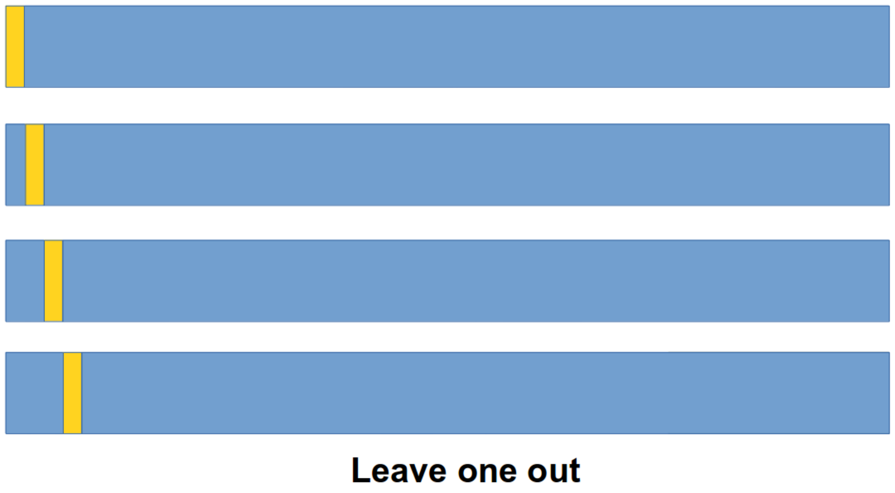


Figure 5.3: Leave One Out

	$ActPos$	$ActNeg$
$PredPos$	TP	FP
$PredNeg$	FN	TN

where *True Positives* (TP) and *True Negatives* (TN) are respectively the number of positive and negative instances correctly classified, *False Positives* (FP) is the number of negative instances misclassified as positive (also called Type I errors), and *False Negatives* (FN) is the number of positive instances misclassified as negative (Type II errors).

- Confusion Matrix in Wikipedia

From the confusion matrix, we can calculate:

- *True positive rate*, or *recall* ($TP_r = recall = TP/TP + FN$) is the proportion of positive cases correctly classified as belonging to the positive class.
- *False negative rate* ($FN_r = FN/TP + FN$) is the proportion of positive cases misclassified as belonging to the negative class.
- *False positive rate* ($FP_r = FP/FP + TN$) is the proportion of negative cases misclassified as belonging to the positive class.
- *True negative rate* ($TN_r = TN/FP + TN$) is the proportion of negative cases correctly classified as belonging to the negative class.

There is a trade-off between FP_r and FN_r as the objective is minimize both metrics (or conversely, maximize the true negative and positive rates). It is possible to combine both metrics into a single figure, predictive *accuracy*:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

to measure performance of classifiers (or the complementary value, the *error rate* which is defined as $1 - accuracy$)

- Precision, fraction of relevant instances among the retrieved instances,

$$\frac{TP}{TP + FP}$$

- Recall$ (*sensitivity* probability of detection, PD) is the fraction of relevant instances that have been retrieved over total relevant instances, $\frac{TP}{TP+FN}$
- *f-measure* is the harmonic mean of precision and recall, $2 \cdot \frac{precision \cdot recall}{precision + recall}$
- G-mean: $\sqrt{PD \times Precision}$

- G-mean2: $\sqrt{PD \times Specificity}$
- J coefficient, $j - coeff = sensitivity + specificity - 1 = PD - PF$
- A suitable and interesting performance metric for binary classification when data are imbalanced is the Matthew's Correlation Coefficient (MCC)~**?**:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

MCC can also be calculated from the confusion matrix. Its range goes from -1 to +1; the closer to one the better as it indicates perfect prediction whereas a value of 0 means that classification is not better than random prediction and negative values mean that predictions are worst than random.

5.2.1 Prediction in probabilistic classifiers

A probabilistic classifier estimates the probability of each of the posible class values given the attribute values of the instance $P(c|x)$. Then, given a new instance, x, the class value with the highest a posteriori probability will be assigned to that new instance (the *winner takes all* approach):

$\psi(x) = argmax_c(P(c|x))$

5.3 Other Metrics used in Software Engineering with Classification

In the domain of defect prediction and when two classes are considered, it is also customary to refer to the *probability of detection*, (pd) which corresponds to the True Positive rate (TP_{rate} or *Sensitivity*) as a measure of the goodness of the model, and *probability of false alarm* (pf) as performance measures~Menzies et al. (2007).

The objective is to find which techniques that maximise pd and minimise pf. As stated by Menzies et al., the balance between these two measures depends on the project characteristics (e.g. real-time systems vs. information management systems) it is formulated as the Euclidean distance from the sweet spot $pf = 0$ and $pd = 1$ to a pair of (pf, pd).

$$balance = 1 - \frac{\sqrt{(0 - pf^2) + (1 - pd^2)}}{\sqrt{2}}$$

It is normalized by the maximum possible distance across the ROC square $(\sqrt{2}, 2)$, subtracted this value from 1, and expressed it as a percentage.

5.4 Graphical Evaluation

5.4.1 Receiver Operating Characteristic (ROC)

The *Receiver Operating Characteristic* (*ROC*)(Fawcett, 2006) curve which provides a graphical visualisation of the results.

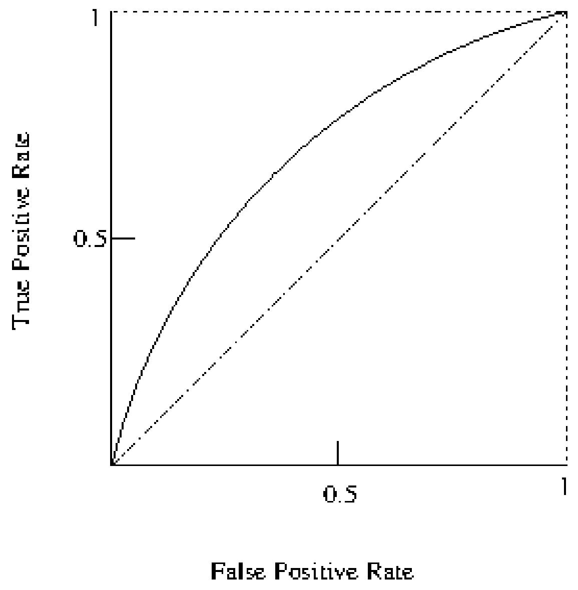


Figure 5.4: Receiver Operating Characteristic

The Area Under the ROC Curve (AUC) also provides a quality measure between positive and negative rates with a single value.

A simple way to approximate the AUC is with the following equation: $AUC = \frac{1+TP_r-FP_r}{2}$

5.4.2 Precision-Recall Curve (PRC)

Similarly to ROC, another widely used evaluation technique is the Precision-Recall Curve (PRC), which depicts a trade off between precision and recall and can also be summarised into a single value as the Area Under the Precision-Recall Curve (AUPRC)~**?**.

%AUPCR is more accurate than the ROC for testing performances when dealing with imbalanced datasets as well as optimising ROC values does not necessarily optimises AUPR values, i.e., a good classifier in AUC space may not be so good in PRC space. %The weighted average uses weights proportional to class frequencies in the data. %The weighted average is computed by weighting the measure of class (TP rate, precision, recall ...) by the proportion of instances there are in that class. Computing the average can be sometimes be misleading. For instance, if class 1 has 100 instances and you achieve a recall of 30%, and class 2 has 1 instance and you achieve recall of 100% (you predicted the only instance correctly), then when taking the average (65%) you will inflate the recall score because of the one instance you predicted correctly. Taking the weighted average will give you 30.7%, which is much more realistic measure of the performance of the classifier.

5.5 Numeric Prediction Evaluation

In the case of defect prediction, it matters the difference between the predicted value and the actual value. Common performance metrics used for numeric prediction are as follows, where $\hat{y_n}$ represents the predicted value and y_n the actual one.

Mean Square Error (MSE)

$MSE = \frac{(\hat{y_1}-y_1)^2+...+(\hat{y_n}-y_n)^2}{n} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i}-y_i)^2$

Root mean-squared error ($RMSE$)

$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(\hat{y}_t-y)^2}{n}}$

Mean Absolute Error (MAE)

$MAE = \frac{|\hat{y_1}-y_1|+...+|\hat{y_n}-y_n|}{n} = \sqrt{\frac{\sum_{t=1}^{n}|\hat{y}_t-y|}{n}}$

Relative Absolute Error (RAE)

$RAE = \frac{\sum_{i=1}^{N}|\hat{\theta}_i-\theta_i|}{\sum_{i=1}^{N}|\bar{\theta}-\theta_i|}$

Root Relative-Squared Error ($RRSE$)

$RRSE = \sqrt{\frac{\sum_{i=1}^{N}|\hat{\theta}_i-\theta_i|}{\sum_{i=1}^{N}|\bar{\theta}-\theta_i|}}$

where $\widehat{\theta}$ is a mean value of θ.

Relative-Squared r (RSE)

$\frac{(p_1-a_1)^2+...+(p_n-a_n)^2}{(a_1-\hat{a})^2+...+(a_n-\hat{a})^2}$

where ($\hat{a}$ is the mean value over the training data)

Relative Absolute Error (RAE)

Correlation Coefficient

Correlation coefficient between two random variables X and Y is defined as $\rho(X,Y) = \frac{\mathbf{Cov}(X,Y)}{\sqrt{\mathbf{Var}(X)\mathbf{Var}(Y)}}$. The sample correlation coefficient} r between two samples x_i and y_j is vvdefined as $r = S_{xy}/\sqrt{S_{xx}S_{yy}}$

Example: Is there any linear relationship between the effort estimates (p_i) and actual effort (a_i)?

$a \| 39, 43, 21, 64, 57, 47, 28, 75, 34, 52$

$p \| 65, 78, 52, 82, 92, 89, 73, 98, 56, 75$

```
p<-c(39,43,21,64,57,47,28,75,34,52)
a<-c(65,78,52,82,92,89,73,98,56,75)
#
cor(p,a)
```

```
## [1] 0.84
```

R^2

Chapter 6

Measures of Evaluation in Software Engineering

6.1 Effort estimation evaluation metrics

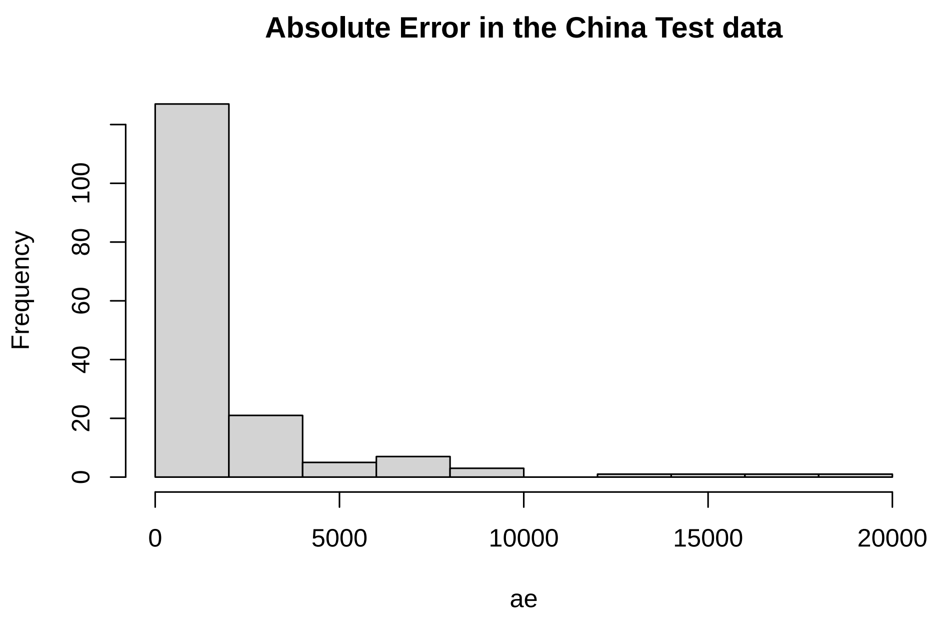
There are several measures typically used in software engineering. In particular for effort estimation, the following metrics are extensively used in addition or instead of statistical measures.

- *Mean of the Absolute Error (MAR)*: compute the absolute errors and take the mean
- *Geometric Mean of the Absolute Error (gMAR)*: more appropriate when the distribution is skewed
- *Mean Magnitude of the Relative Error (MMRE)*: this measure has been critisized many times as a biased measure ($\frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|/y_i}{n}$)
- *Median Magnitude of the Relative Error (MdMRE)*: using the median instead of the mean
- *Level of Prediction* ($Pred(l)$) defined as the percentage of estimates that are within the percentage level l of the actual values. The level of prediction is typically set at 25% below and above the actual value and an estimation method is considered good if it gives a result of more than 75%.
- *Standardised Accuracy (SA)* (proposed by Shepperd and MacDonnell): this measure overcomes all the problems of the MMRE. It is defined as the MAR relative to random guessing ($SA = 1 - \frac{MAR}{\overline{MAR}_{P_0}} \times 100$)
- *Random guessing*: $\overline{MAR}_{P_0}$ is defined as: predict a $\hat{y}_t$ for the target case t by randomly sampling (with equal probability) over all the remaining n-1

cases and take $\hat{y}_t = y_r$ where r is drawn randomly from 1 to n and $r \neq t$.

- *Exact* $\overline{MAR}_{P_0}$: it is an improvement over $\overline{MAR}_{P_0}$. For small datasets the "random guessing" can be computed exactly by iterating over all data points.

```r
library(foreign)
gm_mean = function(x, na.rm=TRUE){
  exp(sum(log(x[x > 0]), na.rm=na.rm) / length(x))}

chinaTrain <- read.arff("./datasets/effortEstimation/china3AttSelectedAFPTrain.arff")
logchina_size <- log(chinaTrain$AFP)
logchina_effort <- log(chinaTrain$Effort)
linmodel_logchina_train <- lm(logchina_effort ~ logchina_size)


chinaTest <- read.arff("./datasets/effortEstimation/china3AttSelectedAFPTest.arff")
b0 <- linmodel_logchina_train$coefficients[1]
b1 <- linmodel_logchina_train$coefficients[2]
china_size_test <- chinaTest$AFP
actualEffort <- chinaTest$Effort
# predEffort <- exp(b0+b1*log(china_size_test))  wr
predEffort <- exp(b0)*china_size_test^b1

err <- actualEffort - predEffort  #error or residual
ae <- abs(err)
hist(ae, main="Absolute Error in the China Test data")
```



```
mar <- mean(ae)
mre <- ae/actualEffort
mmre <- mean(mre)
mdmre <- median(mre)
gmar <- gm_mean(ae)
mar
```

```
## [1] 1867
```

```
mmre
```

```
## [1] 1.15
```

```
mdmre
```

```
## [1] 0.551
```

```
gmar
```

```
## [1] 833
```

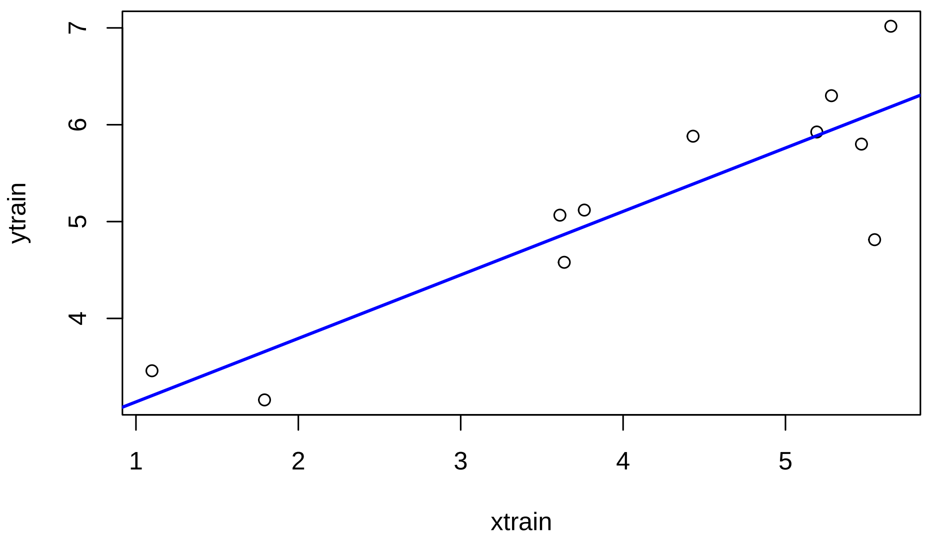
```
level_pred <- 0.25 #below and above (both)
lowpred <- actualEffort*(1-level_pred)
uppred <-  actualEffort*(1+level_pred)
pred  <-  predEffort <= uppred & predEffort >= lowpred  #pred is a vector with logical values
Lpred <- sum(pred)/length(pred)
Lpred
```

```
## [1] 0.186
```

6.3 Building a Linear Model on the Telecom1 dataset

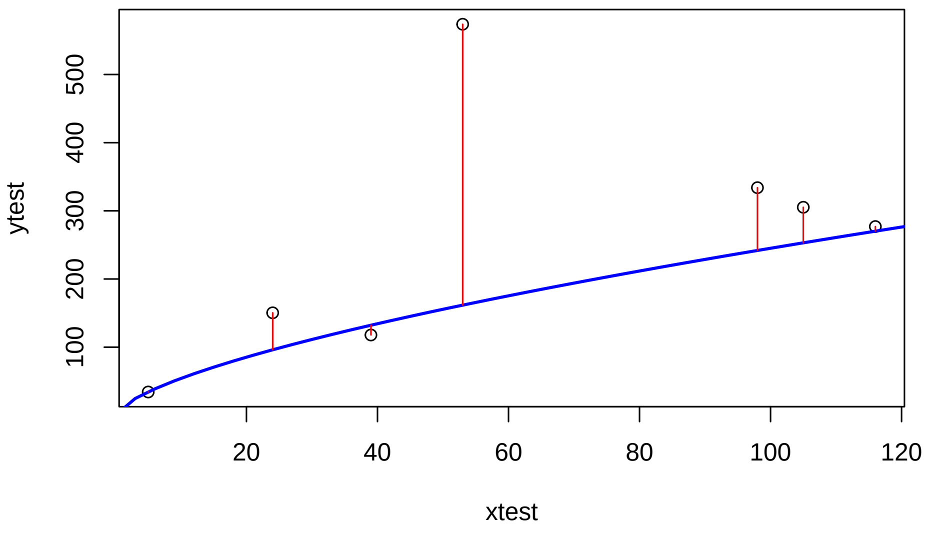
- Although there are few data points we split the file into Train (2/3) and Test (1/3)

```
telecom1 <- read.table("./datasets/effortEstimation/Telecom1.csv", sep=",",header=TRUE

samplesize <- floor(0.66*nrow(telecom1))
set.seed(012) # to make the partition reproducible
train_idx <- sample(seq_len(nrow(telecom1)), size = samplesize)
telecom1_train <- telecom1[train_idx, ]
telecom1_test <- telecom1[-train_idx, ]

par(mfrow=c(1,1))
# transformation of variables to log-log
xtrain <- log(telecom1_train$size)
ytrain <- log(telecom1_train$effort)

lmtelecom1 <- lm( ytrain ~ xtrain)
plot(xtrain, ytrain)


abline(lmtelecom1, lwd=2, col="blue")
```



```
b0_tel1 <- lmtelecom1$coefficients[1]
b1_tel1 <- lmtelecom1$coefficients[2]

# calculate residuals and predicted values
```

```r
res <- signif(residuals(lmtelecom1), 5)

xtest <- telecom1_test$size
ytest <- telecom1_test$effort
# pre_tel1 <- exp(b0_tel1+b1_tel1*log(xtest))
pre_tel1 <- exp(b0_tel1)*xtest^b1_tel1
# plot distances between points and the regression line
plot(xtest, ytest)
curve(exp(b0_tel1+b1_tel1*log(x)), from=0, to=300, add=TRUE, col="blue", lwd=2)
segments(xtest, ytest, xtest, pre_tel1, col="red")
```



6.4 Building a Linear Model on the Telecom1 dataset with all observations

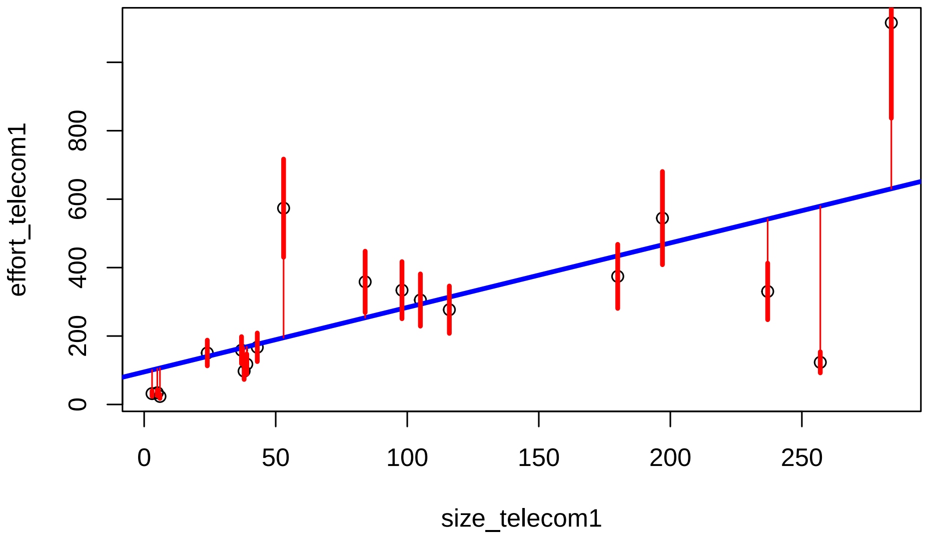
- Just to visualize results

```r
par(mfrow=c(1,1))

effort_telecom1 <- telecom1$effort
size_telecom1 <- telecom1$size

lmtelecom <- lm(effort_telecom1 ~ size_telecom1)
plot(size_telecom1, effort_telecom1)
abline(lmtelecom, lwd=3, col="blue")
# calculate residuals and predicted values
res <- signif(residuals(lmtelecom), 5)
predicted <- predict(lmtelecom)
```

```
# plot distances between points and the regression line
segments(size_telecom1, effort_telecom1, size_telecom1, predicted, col="red")

level_pred <- 0.25 #below and above (both)
lowpred <- effort_telecom1*(1-level_pred)
uppred <-  effort_telecom1*(1+level_pred)
predict_inrange  <-  predicted <= uppred & predicted >= lowpred  #pred is a vector wit
Lpred <- sum(predict_inrange)/length(predict_inrange)
Lpred
```

```
## [1] 0.444
```

```
#Visually plot lpred
segments(size_telecom1, lowpred, size_telecom1, uppred, col="red", lwd=3)
```



```
err_telecom1 <- abs(effort_telecom1 - predicted)
mar_tel1 <- mean(err_telecom1)
mar_tel1
```

```
## [1] 125
```

6.5 Standardised Accuracy Examples

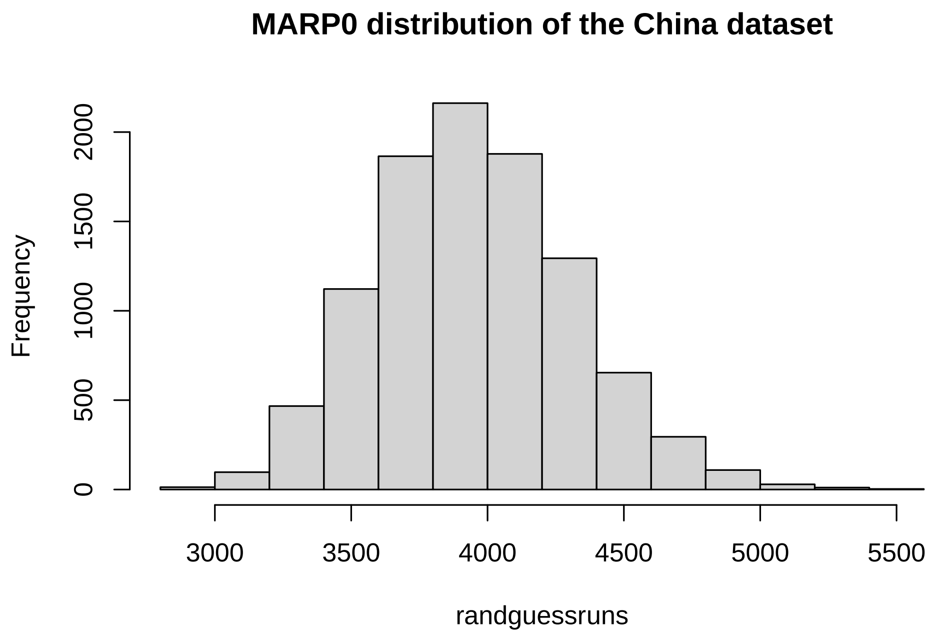
6.5.1 Standardised Accuracy MARP0 using the China Test dataset

- Computing $MARP_0$ in the China Test data

```
estimEffChinaTest <- predEffort  # This will be overwritten, no problem
numruns <- 9999
randguessruns <- rep(0, numruns)
for (i in 1:numruns) {
  for (j in 1:length(estimEffChinaTest)) {
    estimEffChinaTest[j] <- sample(actualEffort[-j],1)}#replacement with random guessingt
  randguessruns[i] <- mean(abs(estimEffChinaTest-actualEffort))
  }
marp0Chinatest <- mean(randguessruns)
marp0Chinatest
```

```
## [1] 3949
```

```
hist(randguessruns, main="MARP0 distribution of the China dataset")
```



```
saChina = (1- mar/marp0Chinatest)*100
saChina
```

```
## [1] 52.7
```

6.5.2 Standardised Accuracy. MARP0 using the Telecom1 dataset

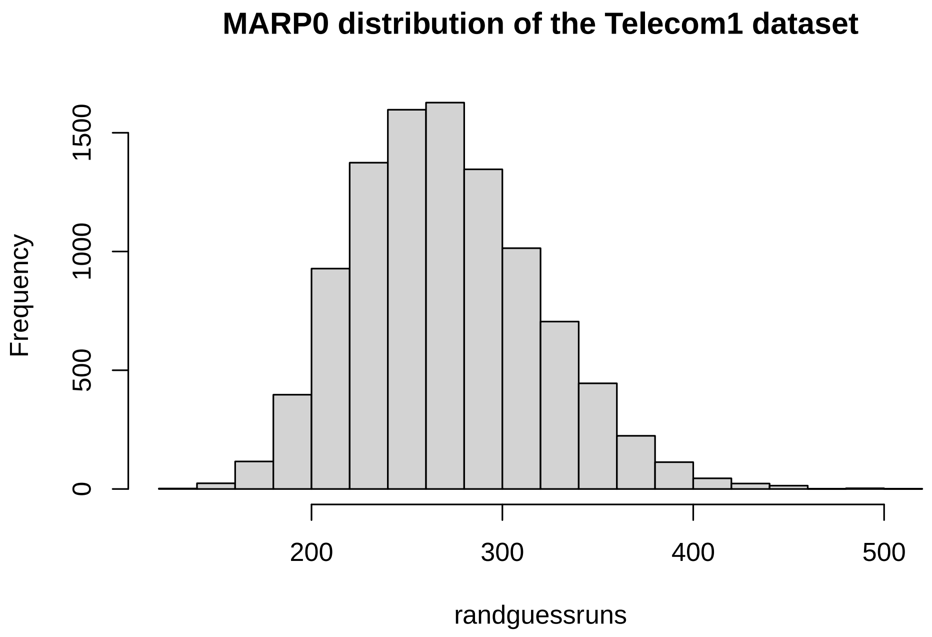
- Computing $MARP_0$

```
telecom1 <- read.table("./datasets/effortEstimation/Telecom1.csv", sep=",",header=TRUE, stringsAs
#par(mfrow=c(1,2))
```

```
#size <- telecom1[1]$size   not needed now
actualEffTelecom1 <- telecom1[2]$effort
estimEffTelecom1 <- telecom1[3]$EstTotal # this will be overwritten
numruns <- 9999
randguessruns <- rep(0, numruns)
for (i in 1:numruns) {
  for (j in 1:length(estimEffTelecom1)) {
    estimEffTelecom1[j] <- sample(actualEffTelecom1[-j],1)}#replacement with random gu
  randguessruns[i] <- mean(abs(estimEffTelecom1-actualEffTelecom1))
  }
marp0telecom1 <- mean(randguessruns)
marp0telecom1
```

```
## [1] 271
```

```
hist(randguessruns, main="MARP0 distribution of the Telecom1 dataset")
```



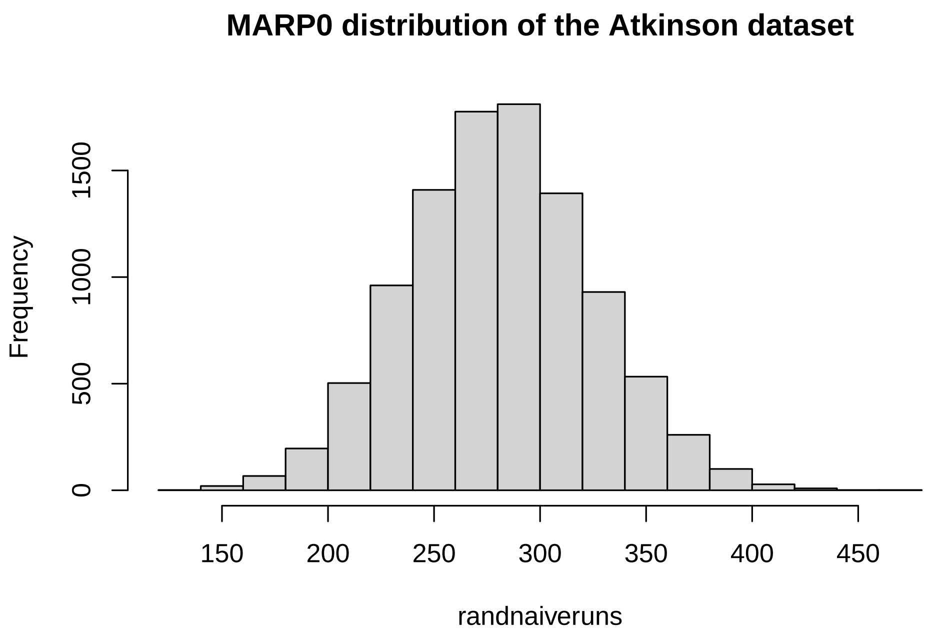
```
saTelecom1 <- (1- mar_tel1/marp0telecom1)*100
saTelecom1
```

```
## [1] 53.9
```

6.5.3 Standard Accuracy MARP0 using the Atkinson Dataset

- For checking results you may use figure Atkinson in Shepperd & MacDonnell

```
## [1] 281
```



6.6 Exact MARP0

Langdon et al(2016) provide a solution to calculate Shepperd and MacDonell's MAR_{P_0}$ exactly. An R code implementation is as follows.

```
#example dataset
atkinson_actual_effort <-
  c(670,912,218,595,267,344,229,190,869,109,289,616,557,416,578,438)
myabs <- function(x,y) abs(x-y)

#diffs is square array whose i,jth element = abs(actual_i - actual_j)
#in practice this is good enough but could be made more efficient by not
#explicitly storing the matrix and only using the values below the diagonal.

diffs <- outer(atkinson_actual_effort,atkinson_actual_effort,myabs)
marp0 <- mean(diffs)
marp0
```

```
## [1] 264
```

```r
#### same procedure without using the outer function
act_effort <-
  c(670,912,218,595,267,344,229,190,869,109,289,616,557,416,578,438)
n <- length(act_effort)
diffs_guess <- matrix(nrow=n, ncol=n)
colnames(diffs_guess) <- act_effort
rownames(diffs_guess) <- act_effort
for (i in 1:n){
  diffs_guess[i,] <- act_effort - act_effort[i]
}

diffs_guess <- abs(diffs_guess)
means_per_point <- apply(diffs_guess, 2, mean)
marp0 <- mean(means_per_point)
marp0
```

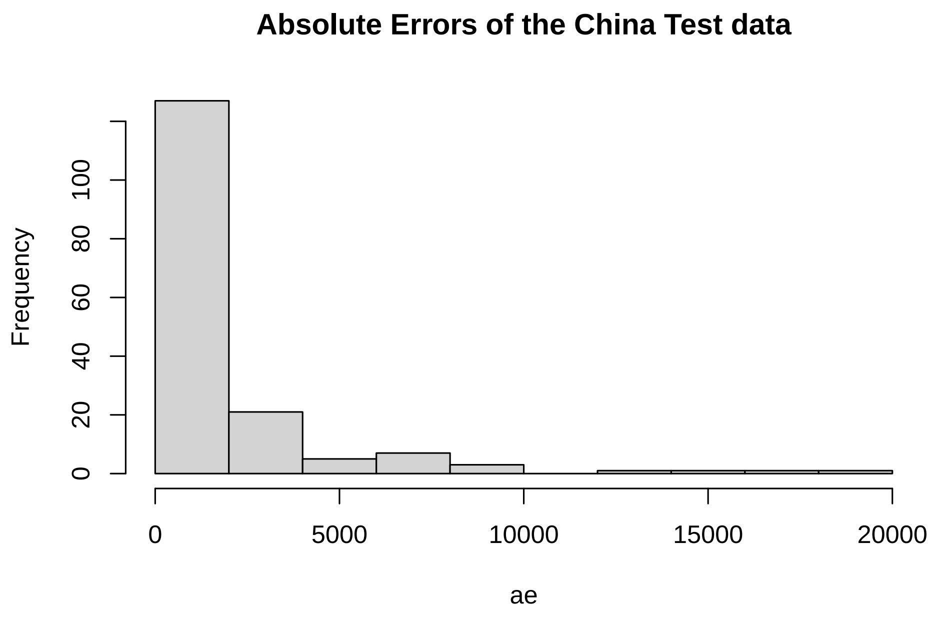
```
## [1] 264
```

6.7 Computing the bootstraped confidence interval of the mean for the Test observations of the China dataset:

```r
library(boot)
```

```
##
## Attaching package: 'boot'

## The following object is masked from 'package:survival':
##
##     aml

## The following object is masked from 'package:lattice':
##
##     melanoma

## The following object is masked from 'package:sm':
##
##     dogs
```

```
hist(ae, main="Absolute Errors of the China Test data")
```



```
level_confidence <- 0.95
repetitionsboot <- 9999
samplemean <- function(x, d){return(mean(x[d]))}
b_mean <- boot(ae, samplemean, R=repetitionsboot)
confint_mean_China <- boot.ci(b_mean)
```

```
## Warning in boot.ci(b_mean): bootstrap variances needed for studentized intervals
```

```
confint_mean_China
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 9999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b_mean)
##
## Intervals :
## Level      Normal              Basic
## 95%   (1420, 2316 )   (1386, 2284 )
##
## Level     Percentile            BCa
## 95%   (1450, 2348 )   (1496, 2419 )
## Calculations and Intervals on Original Scale
```

- Computing the bootstraped geometric mean

```
boot_geom_mean <- function(error_vec){
  log_error <- log(error_vec[error_vec > 0])
  log_error <-log_error[is.finite(log_error)] #remove the -Inf value before calculatin
  samplemean <- function(x, d){return(mean(x[d]))}
  b <- boot(log_error, samplemean, R=repetitionsboot) # with package boot
  # this is a boot for the logs
  return(b)
}
# BCAconfidence interval for the geometric mean
BCAciboot4geommean <- function(b){
  conf_int <- boot.ci(b, conf=level_confidence, type="bca")$bca #following 10.9 of Uga
  conf_int[5] <- exp(conf_int[5]) # the boot was computed with log. Now take the measu
  conf_int[4] <- exp(conf_int[4])
  return (conf_int)
}
# this is a boot object
b_gm <- boot_geom_mean(ae) #"ae" is the absolute error in the China Test data
print(paste0("Geometric Mean of the China Test data: ", round(exp(b_gm$t0), digits=3)))
```

```
## [1] "Geometric Mean of the China Test data: 832.55"
```

```
b_ci_gm <- BCAciboot4geommean(b_gm)
print(paste0("Confidence Interval: ", round(b_ci_gm[4], digits=3), " - ", round(b_ci_gm
```

```
## [1] "Confidence Interval: 679.439 - 1016.691"
```

```
# Make a % confidence interval bca
# BCAciboot <- function(b){
#   conf_int <- boot.ci(b, conf=level_confidence, type="bca")$bca #following 10.9 of U
#   return (conf_int)
# }
```

6.8 Defect prediction evaluation metrics

In addition to the machine learning metrics for classification, Jiang *et al.* provide a survey (Jiang et al., 2008).

Part IV

Advanced Topics

Chapter 7

Feature Selection

This technique consists in selecting the most relevant attributes. The need of applying FS includes the following points:

- A reduced volume of data allows different data mining or searching techniques to be applied.
- Irrelevant and redundant attributes can generate less accurate and more complex models. Furthermore, data mining algorithms can be executed faster.
- It is possible to avoid the collection of data for those irrelevant and redundant attributes in the future.

FS algorithms designed with different evaluation criteria broadly fall into two categories:

- The *filter model* relies on general characteristics of the data to evaluate and select feature subsets without involving any data mining algorithm.
- The *wrapper model* requires one predetermined mining algorithm and uses its performance as the evaluation criterion. It searches for features better suited to the mining algorithm aiming to improve mining performance, but it also tends to be more computationally expensive than filter model [11, 12].

7.1 Instance Selection

TBD

7.2 Missing Data Imputation

TBD

Chapter 8

Feature Selection Example

Feature Selection in R and Caret

```
library(caret)
library(doParallel) # parallel processing
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
library(dplyr) # Used by caret
library(pROC) # plot the ROC curve
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(foreign)

### Use the segmentationData from caret
# Load the data and construct indices to divided it into training and test data sets.
#set.seed(10)
kc1 <- read.arff("./datasets/defectPred/D1/KC1.arff")
```

```
 inTrain <- createDataPartition(y = kc1$Defective,
  ## the outcome data are needed
  p = .75,
  ## The percentage of data in the
  ## training set
  list = FALSE)
```

The function `createDataPartition` does a stratified partitions.

```
training <- kc1[inTrain,]
nrow(training)
```

```
## [1] 1573
```

```
testing <- kc1[-inTrain, ]
nrow(testing)
```

```
## [1] 523
```

The train function can be used to + evaluate, using resampling, the effect of model tuning parameters on performance + choose the "optimal" model across these parameters + estimate model performance from a training set

```
fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated ten times
                           repeats = 10)
```

gbmFit1 <- train(Defective ~ ., data = training, method = "gbm", trControl = fitControl, ## This last option is actually one ## for gbm() that passes through verbose = FALSE) gbmFit1

```
plsFit <- train(Defective ~ .,
 data = training,
 method = "pls",
 ## Center and scale the predictors for the training
 ## set and all future samples.
 preProc = c("center", "scale")
)
```

```
# To fix
# testPred <- predict(plsFit, testing)
# postResample(testPred, testing$Defective)
# sensitivity(testPred, testing$Defective)
# confusionMatrix(testPred, testing$Defective)
```

When there are three or more classes, confusion matrix will show the confusion matrix and a set of "one-versus-all" results.

Chapter 9

Further Classification Models

9.1 Multilabel classification

Some datasets, for example, reviews of applications and mobile applications repositories such as App Store or Google play contain reviews that can have several labels at the same time (e.g. bugs, feature requests, etc.)

9.2 Semi-supervised Learning

Self train a model on semi-supervised data http://www.inside-r.org/packages/cran/dmwr/docs/SelfTrain

```
library(DMwR2)

## Small example with the Iris classification data set
data(iris)

## Dividing the data set into train and test sets
idx <- sample(150,100)
tr <- iris[idx,]
ts <- iris[-idx,]

## Learn a tree with the full train set and test it
stdTree <- rpartXse(Species~ .,tr,se=0.5)
table(predict(stdTree,ts,type='class'),ts$Species)

## Now let us create another training set with most of the target
## variable values unknown
trSelfT <- tr
nas <- sample(100,70)
trSelfT[nas,'Species'] <- NA

## Learn a tree using only the labelled cases and test it
baseTree <- rpartXse(Species~ .,trSelfT[-nas,],se=0.5)
table(predict(baseTree,ts,type='class'),ts$Species)

## The user-defined function that will be used in the self-training process
f <- function(m,d) {
      l <- predict(m,d,type='class')
      c <- apply(predict(m,d),1,max)
      data.frame(cl=l,p=c)
}

## Self train the same model using the semi-superside data and test the
## resulting model
treeSelfT <- SelfTrain(Species~ .,trSelfT,learner('rpartXse',list(se=0.5)),'f')
table(predict(treeSelfT,ts,type='class'),ts$Species)
```

Chapter 10

Social Network Analysis in SE

In this example, we will data from the MSR14 challenge. Further information and datasets: http://openscience.us/repo/msr/msr14.html

Similar databases can be obtained using MetricsGrimoire or other tools.

In this simple example, we create a network form the users and following extracted from GitHub and stored in a MySQL database.

We can read a file directely from MySQL dump

```
library(RMySQL)

# Connecting to MySQL
mydb = dbConnect(MySQL(), user='msr14', password='msr14', dbname='msr14', host='localhost')

# Retrieving data from MySQL
sql <- "select user_id, follower_id from followers limit 100;"
rs = dbSendQuery(mydb, sql)
data <- fetch(rs, n=-1)
```

Alternatively, we can create e CSV file directly from MySQL and load it

```
$mysql -u msr14 -pmsr14 msr14



> SELECT 'user','follower'
UNION ALL
SELECT user_id,follower_id
    FROM followers
    LIMIT 1000
    INTO OUTFILE "/tmp/followers.csv"
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n';
```

```
# Data already extracted and stored as CSV file (for demo purposes)
dat = read.csv("./datasets/sna/followers.csv", header = FALSE, sep = ",")
dat <- head(dat,100)
```

We can now create the graph

```
library(igraph)
```

```
##
## Attaching package: 'igraph'

## The following object is masked from 'package:arules':
##
##     union

## The following object is masked from 'package:class':
##
##     knn

## The following object is masked from 'package:modeltools':
##
##     clusters

## The following objects are masked from 'package:lubridate':
##
##     %--%, union

## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```
# Create a graph
g <- graph.data.frame(dat, directed = TRUE)
```

Some values:

```
summary(g);
```

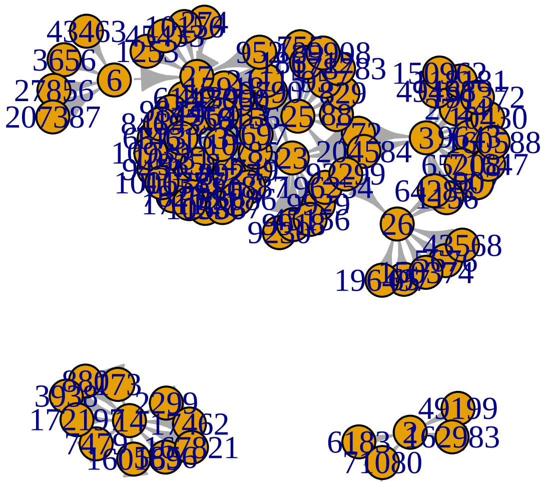
```
## IGRAPH 122e5d9 DN-- 95 100 --
## + attr: name (v/c)
```

Plotting the graph:

```
layout1 <-  layout.fruchterman.reingold(g)
plot(g, layout1)
```

Other layout

```
plot(g, layout=layout.kamada.kawai)
```



A tk application can launched to show the plot interactively:

```
plot(g, layout = layout.fruchterman.reingold)
```

Some metrics:

```
metrics <- data.frame(
  deg = degree(g),
  bet = betweenness(g),
  clo = closeness(g),
  eig = evcent(g)$vector,
  cor = graph.coreness(g)
)

#
head(metrics)
```

```
##         deg bet   clo     eig cor
## 6183      1   0 1.000 0.00000   1
## 49199     1   0 1.000 0.00000   1
## 71080     1   0 1.000 0.00000   1
## 162983   1   0 1.000 0.00000   1
## 772      3   0 0.333 0.10409   2
## 907      1   0 1.000 0.00814   1
```

To fix and to do: Explain metrics and better graphs

```
library(ggplot2)

ggplot(
  metrics,
  aes(x=bet, y=eig,
    label=rownames(metrics),
    colour=res, size=abs(res))
)+
xlab("Betweenness Centrality")+
ylab("Eigenvector Centrality")+
geom_text()
+
theme(title="Key Actor Analysis")


V(g)$label.cex <- 2.2 * V(g)$degree / max(V(g)$degree)+ .2
V(g)$label.color <- rgb(0, 0, .2, .8)
V(g)$frame.color <- NA
egam <- (log(E(g)$weight)+.4) / max(log(E(g)$weight)+.4)
E(g)$color <- rgb(.5, .5, 0, egam)
E(g)$width <- egam
# plot the graph in layout1
plot(g, layout=layout1)
```

Further information:

http://sna.stanford.edu/lab.php?l=1

Chapter 11

Text Mining Software Engineering Data

In software engineering, there is a lot of information in plain text such as requirements, bug reports, mails, reviews from applicatons, etc. Typically that information can be extracted from Software Configuration Management Systems (SCM), Bug Tracking Systems (BTS) such as Bugzilla or application stores such as Google Play or Apple's AppStore, etc. can be mined to extract relevant information. Here we briefly explain the text mining process and how this can be done with R.

A well-known package for *text mining* is `tm` (Feinerer and Hornik, 2015, Feinerer et al. (2008)). Another popular package is `wordcloud`.

11.1 Terminology

The workflow that we follow for analyzing a set of text documents are:

1. Importing data. A *Corpus* is a collection of text documents, implemented as VCorpus (corpora are R object held in memory). The `tm` provides several corpus constructors: `DirSource`, `VectorSource`, or `DataframeSource` (`getSources()`).

There are several parameters that control the creation of a *Corpus*. ((The parameter readerControl of the corpus constructor has to be a list with the named components reader and language))

2. Preprocessing: in this step we may remove common words, punctuation and we may perform other operations. We may do this operations after creating the DocumentTermMatrix.

3. Inspecting and exploring data: Individual documents can be accessed via [[

4. Transformations: Transformations are done via the `tm_map()` function. + `tm_map(_____, stripWhitespace)` + `tm_map(_____, content_transformer(tolower))` + `tm_map(_____, removeWords, stopwords("english"))` + `tm_map(_____, stemDocument)`

5. Creating `Term-Document` Matrices: TermDocumentMatrix and DocumentTermMatrix + A document term matrix is a matrix with documents as the rows and terms as the columns. Each cell of the matrix contains the count of the frequency of words. We use DocumentTermMatrix() to create the matrix. + `inspect(DocumentTermMatrix( newsreuters, list(dictionary = c("term1", "term2", "term3"))))`. It displays detailed information on a corpus or a term-document matrix.

6. Relationships between terms. + `findFreqTerms(_____, anumber)` + `findAssocs(Mydtm, "aterm", anumbercorrelation)` + A dictionary is a (multi-)set of strings. It is often used to denote relevant terms in text mining.

7. Clustering and Classification

11.2 Example of classifying bugs from Bugzilla

Bugzilla is Issue Tracking System that allow us to follow the evolution of a project.

The following example shows how to work with entries from Bugzilla. It is assumed that the data has been extracted and we have the records in a flat file (this can be done using Web crawlers or directly using the SQL database).

```
library(foreign)
# path_name <- file.path("C:", "datasets", "textMining")
# path_name
# dir(path_name)

#Import data
options(stringsAsFactors = FALSE)
d <- read.arff("./datasets/textMining/reviewsBugs.arff" )
str(d) #print out information about d
```

```
## 'data.frame':    789 obs. of  2 variables:
##  $ revContent: chr  "Can't see traffic colors now With latest updates I can't see t
##  $ revBug    : Factor w/ 2 levels "N","Y": 2 1 1 1 1 1 2 1 2 1 ...
```

```
head(d,2) # the first two rows of d.
```

```
##
```

```
## 1 Can't see traffic colors now With latest updates I can't see the traffic green/red/yellow -
## 2
##   revBug
## 1      Y
## 2      N
```

```
# fifth entry
d$revContent[5]
```

```
## [1] "Just deleted No I don't want to sign in or sign up for anything stop asking"
```

```
d$revBug[5]
```

```
## [1] N
## Levels: N Y
```

Creating a Document-Term Matrix (DTM)

Now, we can explore things such as “which words are associated with”feature”?”

```
# which words are associated with "bug"?
findAssocs(dtm, 'bug', .3) # minimum correlation of 0.3. Change accordingly.
```

```
## $bug
##     it?    mini   major   users causing    ipad
##    1.00    0.92    0.91    0.80    0.62    0.57
```

And find frequent terms.

```
findFreqTerms(dtm, 15) #terms that appear 15 or more times, in this case
```

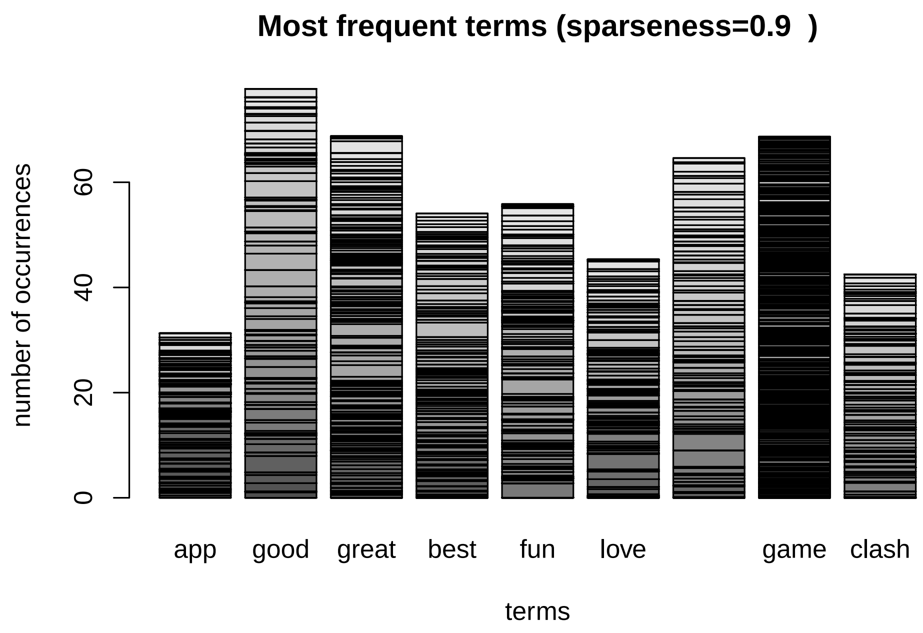
```
##  [1] "google"    "map"       "like"      "app"       "just"      "good"
##  [7] "crashes"   "maps"      "time"      "get"       "much"      "really"
## [13] "update"    "great"     "nice"      "best"      "ever"      "fun"
## [19] "review"    "love"      "awesome"   "cool"      "amazing"   "game"
## [25] "clans"     "clash"     "game."     "game!"     "addicting" "play"
## [31] "playing"   "addictive"
```

Remove some terms

```
sparseparam <- 0.90 # will make the matrix 90% empty space, maximum. Change this, as you like.
dtm_sprs <- removeSparseTerms(dtm,sparse=sparseparam)
inspect(dtm_sprs)
```

```
## <<DocumentTermMatrix (documents: 789, terms: 9)>>
## Non-/sparse entries: 1233/5868
## Sparsity           : 83%
## Maximal term length: 7
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
## Sample             :
##      Terms
## Docs  app awesome best clash fun game good great love
##    159   0    0.00    0   1.6   0    0  0.0     0 1.46
##    163   0    3.12    0   0.0   0    0  0.0     0 0.00
##    178   0    3.12    0   0.0   0    0  0.0     0 0.00
##    400   0    0.00    0   0.0   0    0  3.1     0 0.00
##    421   0    0.00    0   0.0   0    0  3.1     0 0.00
##    472   0    0.00    0   0.0   0    0  3.1     0 0.00
##    50    0    0.00    0   0.0   0    0  3.1     0 0.00
##    525   0    1.56    0   0.0   0    0  0.0     0 1.46
##    527   0    0.00    0   0.0   0    0  3.1     0 0.00
##    532   0    0.00    0   1.6   0    0  0.0     0 1.46
```

```
maintitle <-paste0("Most frequent terms (sparseness=" ,sparseparam , "  )")
barplot(as.matrix(dtm_sprs),xlab="terms",ylab="number of occurrences", main=maintitle)
```



```
# organize terms by their frequency

freq_dtm_sprs <- colSums(as.matrix(dtm_sprs))
length(freq_dtm_sprs)
```

```
## [1] 9
```

```
sorted_freq_dtm_sprs <- sort(freq_dtm_sprs, decreasing = TRUE)
sorted_freq_dtm_sprs
```

```
##     good    great     game awesome      fun     best     love    clash      app
##     77.8     68.8     68.7    64.6     55.8     54.1     45.4     42.5     31.3
```

Create a data frame that will be the input to the classifier. Last column will be the label.

As data frame:

```
#dtmdf <- as.data.frame(dtm.90)
#dtmdf <- as.data.frame(inspect(dtm_sprs))
dtmdf <- as.data.frame(as.matrix(dtm_sprs))
# rownames(dtm)<- 1:nrow(dtm)

class <- d$revBug
dtmdf <- cbind(dtmdf,class)
head(dtmdf, 3)
```

Use any classifier now: - split the dataframe into training and testing - Build the classification model using the training subset - apply the model to the testing subset and obtain the Confusion Matrix - Analise the results

```
library(caret)
library(randomForest)


inTraining <- createDataPartition(dtmdf$class, p = .75, list = FALSE)
training <- dtmdf[ inTraining,]
testing  <- dtmdf[-inTraining,]

fitControl <- trainControl(## 5-fold CV
                           method = "repeatedcv",
                           number = 5,
                           ## repeated ten times
                           repeats = 5)


gbmFit1 <- train(class ~ ., data = training,
                 method = "gbm",
                 trControl = fitControl,
                 ## This last option is actually one
                 ## for gbm() that passes through
                 verbose = FALSE)

gbmFit1
```

```
## Stochastic Gradient Boosting
##
## 593 samples
##   9 predictor
##   2 classes: 'N', 'Y'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 475, 474, 474, 474, 475, 475, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy  Kappa
##   1                   50      0.798     0.000
##   1                  100      0.801     0.081
##   1                  150      0.806     0.189
##   2                   50      0.805     0.149
##   2                  100      0.802     0.219
##   2                  150      0.798     0.211
##   3                   50      0.806     0.215
##   3                  100      0.802     0.236
##   3                  150      0.801     0.233
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
# trellis.par.set(caretTheme())
# plot(gbmFit1)
#
# trellis.par.set(caretTheme())
# plot(gbmFit1, metric = "Kappa")

head(predict(gbmFit1, testing, type = "prob"))
```

```
##       N      Y
## 1 0.690 0.3105
## 2 0.583 0.4166
## 3 0.690 0.3105
## 4 0.908 0.0919
## 5 0.353 0.6472
## 6 0.690 0.3105
```

```
conf_mat <- confusionMatrix(testing$class, predict(gbmFit1, testing))
conf_mat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N 152   5
##          Y  27  12
##
##                Accuracy : 0.837
##                  95% CI : (0.777, 0.886)
##     No Information Rate : 0.913
##     P-Value [Acc > NIR] : 0.999819
##
##                   Kappa : 0.35
##
##  Mcnemar's Test P-Value : 0.000205
##
##             Sensitivity : 0.849
##             Specificity : 0.706
##          Pos Pred Value : 0.968
##          Neg Pred Value : 0.308
##              Prevalence : 0.913
##          Detection Rate : 0.776
##    Detection Prevalence : 0.801
##       Balanced Accuracy : 0.778
##
##        'Positive' Class : N
##
```

We may compute manually all derived variables from the Confusion Matrix. See Section – with the description of the Confusion Matrix

```
# str(conf_mat)
TruePositive <- conf_mat$table[1,1]
TruePositive
```

```
## [1] 152
```

```
FalsePositive <- conf_mat$table[1,2]
FalsePositive
```

```
## [1] 5
```

```
FalseNegative <- conf_mat$table[2,1]
FalseNegative
```

```
## [1] 27
```

```
TrueNegative <- conf_mat$table[2,2]
TrueNegative
```

```
## [1] 12
```

```
# Sum columns in the confusion matrix
ConditionPositive <- TruePositive + FalseNegative
ConditionNegative <- FalsePositive + TrueNegative
```

```r
TotalPopulation <- ConditionPositive + ConditionNegative
TotalPopulation
```

```
## [1] 196
```

```r
#Sum rows in the confusion matrix
PredictedPositive <- TruePositive + FalsePositive
PredictedNegative <- FalseNegative + TrueNegative
# Total Predicted must be equal to the total population
PredictedPositive+PredictedNegative
```

```
## [1] 196
```

```r
SensitivityRecall_TPR <- TruePositive / ConditionPositive
SensitivityRecall_TPR
```

```
## [1] 0.849
```

```r
Specificity_TNR_SPC <- TrueNegative / ConditionNegative
Specificity_TNR_SPC
```

```
## [1] 0.706
```

```r
Precision_PPV <- TruePositive / PredictedPositive
Precision_PPV
```

```
## [1] 0.968
```

```r
NegativePredictedValue_NPV <- TrueNegative / PredictedNegative
NegativePredictedValue_NPV
```

```
## [1] 0.308
```

```r
Prevalence <- ConditionPositive / TotalPopulation
Prevalence
```

```
## [1] 0.913
```

```r
Accuracy_ACC <- (TruePositive + TrueNegative) / TotalPopulation
Accuracy_ACC
```

```
## [1] 0.837
```

```r
FalseDiscoveryRate_FDR <- FalsePositive / PredictedPositive
FalseDiscoveryRate_FDR
```

```
## [1] 0.0318
```

```r
FalseOmisionRate_FOR <- FalseNegative / PredictedNegative
FalseOmisionRate_FOR
```

```
## [1] 0.692
```

```
FallOut_FPR <- FalsePositive / ConditionNegative
FallOut_FPR
```

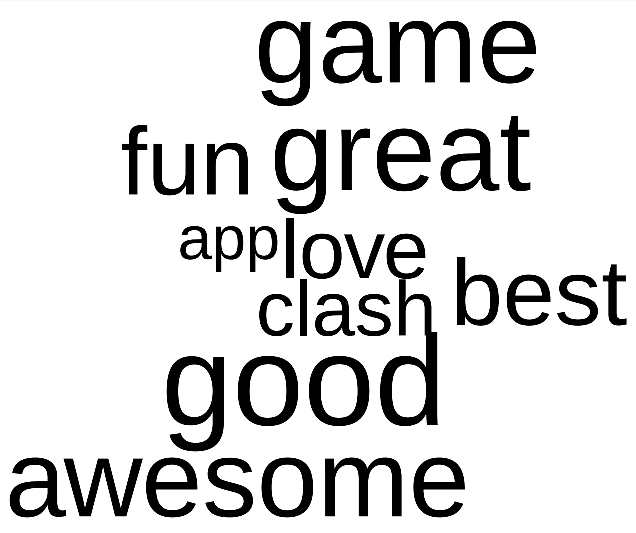
```
## [1] 0.294
```

```
MissRate_FNR <- FalseNegative / ConditionPositive
MissRate_FNR
```

```
## [1] 0.151
```

And finally, a word cloud as an example that appears everywhere these days.

```
library(wordcloud)

# calculate the frequency of words and sort in descending order.
wordFreqs=sort(colSums(as.matrix(dtm_sprs)),decreasing=TRUE)

wordcloud(words=names(wordFreqs),freq=wordFreqs)
```



11.3 Extracting data from Twitter

The hardest bit is to link with Twitter. Using the TwitteR package is explained following this example.

Chapter 12

Time Series

Many sources of information are time related. For example, data from Software Configuration Management (SCM) such as Git, GitHub) systems or Dashboards such as Metrics Grimoire from Bitergia or SonarQube

With MetricsGrimore or SonarQube we can extract datasets or dump of databases. For example, a dashboard for the OpenStack project is located at http://activity.openstack.org/dash/browser/ and provides datasets as MySQL dumps or JSON files.

With R we can read a JSON file as follows:

```
library(jsonlite)
# Get the JSON data
# gm <- fromJSON("http://activity.openstack.org/dash/browser/data/json/nova.git-scm-rep-evolutio
gm <- fromJSON('./datasets/timeSeries/nova.git-scm-rep-evolutionary.json')
str(gm)
```

```
## List of 13
##  $ added_lines  : num [1:287] 431874 406 577 697 7283 ...
##  $ authors      : int [1:287] 1 1 4 2 7 5 4 9 8 11 ...
##  $ branches     : int [1:287] 1 1 1 1 1 1 1 1 1 1 ...
##  $ commits      : int [1:287] 3 4 16 11 121 38 35 90 66 97 ...
##  $ committers   : int [1:287] 1 1 4 2 7 5 4 9 8 11 ...
##  $ date         : chr [1:287] "May 2010" "May 2010" "Jun 2010" "Jun 2010" ...
##  $ files        : int [1:287] 1878 9 13 7 144 111 28 1900 89 101 ...
##  $ id           : int [1:287] 0 1 2 3 4 5 6 7 8 9 ...
##  $ newauthors   : int [1:287] 1 1 2 0 4 1 0 4 2 3 ...
##  $ removed_lines: num [1:287] 864 530 187 326 2619 ...
##  $ repositories : int [1:287] 1 1 1 1 1 1 1 1 1 1 ...
##  $ unixtime     : chr [1:287] "1274659200" "1275264000" "1275868800" "1276473600" ...
##  $ week         : int [1:287] 201021 201022 201023 201024 201025 201026 201027 201028 201029 2
```

Now we can use time series packages. First, after loading the libraries, we need to create a time series object.

```
# TS libraries
library(xts)
```

```
##
## Attaching package: 'xts'
```

```
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
library(forecast)

# Library to deal with dates
library(lubridate)

# Ceate a time series object
gmts <- xts(gm$commits,seq(ymd('2010-05-22'),ymd('2015-11-16'), by = '1 week'))

# TS Object
str(gmts)
```

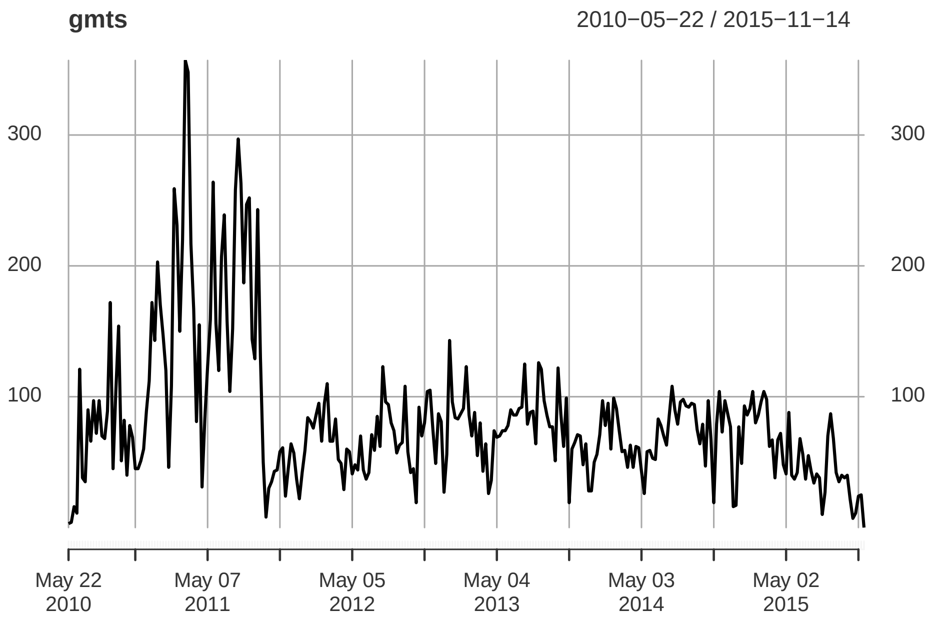
```
## An 'xts' object on 2010-05-22/2015-11-14 containing:
##   Data: int [1:287, 1] 3 4 16 11 121 38 35 90 66 97 ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
head(gmts, 3)
```

```
##            [,1]
## 2010-05-22    3
## 2010-05-29    4
## 2010-06-05   16
```

Visualise the time series object

```
plot(gmts)
```



Arima model:

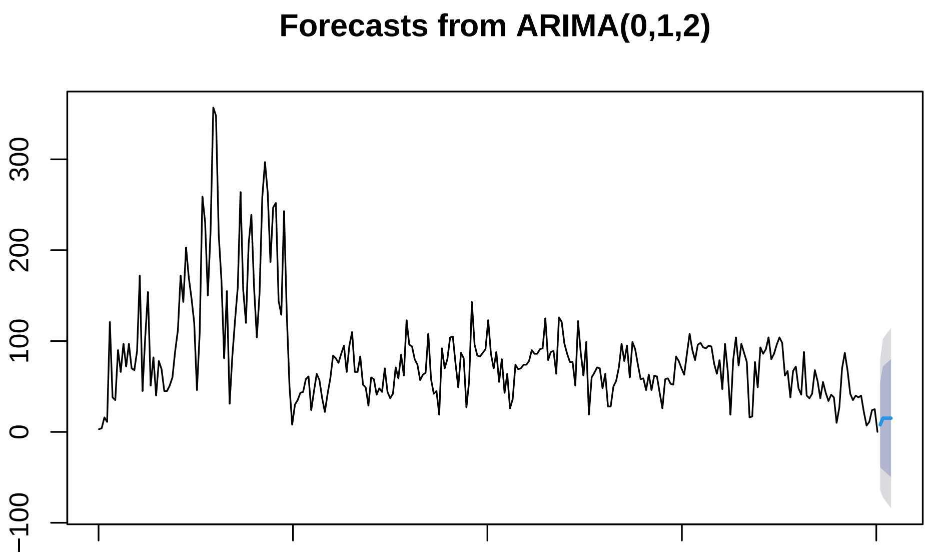
```
fit <- auto.arima(gmts)
fit
```

```
## Series: gmts
## ARIMA(0,1,2)
##
## Coefficients:
##          ma1     ma2
##       -0.312  -0.307
## s.e.   0.058   0.064
##
## sigma^2 = 1341:  log likelihood = -1435
## AIC=2876   AICc=2876   BIC=2887
```

```
forecast(fit, 5)
```

```
##      Point Forecast Lo 80 Hi 80 Lo 95 Hi 95
## 2010           7.75 -39.2  54.7 -64.0  79.5
## 2017          15.16 -41.8  72.1 -72.0 102.3
## 2024          15.16 -44.6  74.9 -76.2 106.5
## 2031          15.16 -47.2  77.5 -80.2 110.5
## 2038          15.16 -49.7  80.0 -84.0 114.3
```

```
plot(forecast(fit, 5))
```



12.1 Web tutorials about Time Series:

http://www.statoek.wiso.uni-goettingen.de/veranstaltungen/zeitreihen/sommer03/ts_r_intro.pdf

http://www.statmethods.net/advstats/timeseries.html

http://a-little-book-of-r-for-time-series.readthedocs.org/en/latest/

https://media.readthedocs.org/pdf/a-little-book-of-r-for-time-series/latest/a-little-book-of-r-for-time-series.pdf

http://www.stat.pitt.edu/stoffer/tsa3/

Part V

Bibliography

Bibliography

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874. ROC Analysis in Pattern Recognition.

Feinerer, I. and Hornik, K. (2015). *tm: Text Mining Package*. R package version 0.6-2.

Feinerer, I., Hornik, K., and Meyer, D. (2008). Text mining infrastructure in r. *Journal of Statistical Software*, 25(5):1–54.

Jiang, Y., Cukic, B., and Ma, Y. (2008). Techniques for evaluating fault prediction models. *Empirical Software Engineering*, 13(5):561–595.

Langdon, W. B., Dolado, J., Sarro, F., and Harman, M. (2016). Exact mean absolute error of baseline predictor, marp0. *Information and Software Technology*, 73(C):16–18.

Menzies, T., Greenwald, J., and Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*.