Data Analysis in Software Engineering using R

Daniel Rodriguez and Javier Dolado

2018-12-26

Contents

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

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

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")`.
- Uncomment the following lines for installing all missing packages (this will take some time):

```
# listofpackages <- c("bookdown", "ggplot2", "vioplot", "UsingR", "fpc", "reshape", "arules","arulesViz
# newpackages <- listofpackages[!(listofpackages %in% installed.packages()[,"Package"])]
# if(length(newpackages)>0) install.packages(newpackages,dependencies = TRUE)
#
# # install from archive
# 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 libraries, e.g
# 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
```

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)
  - `help("    ")` command
- R as a calculator. Console: It uses the command-line interface.

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 scalar
```

```
## [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
```

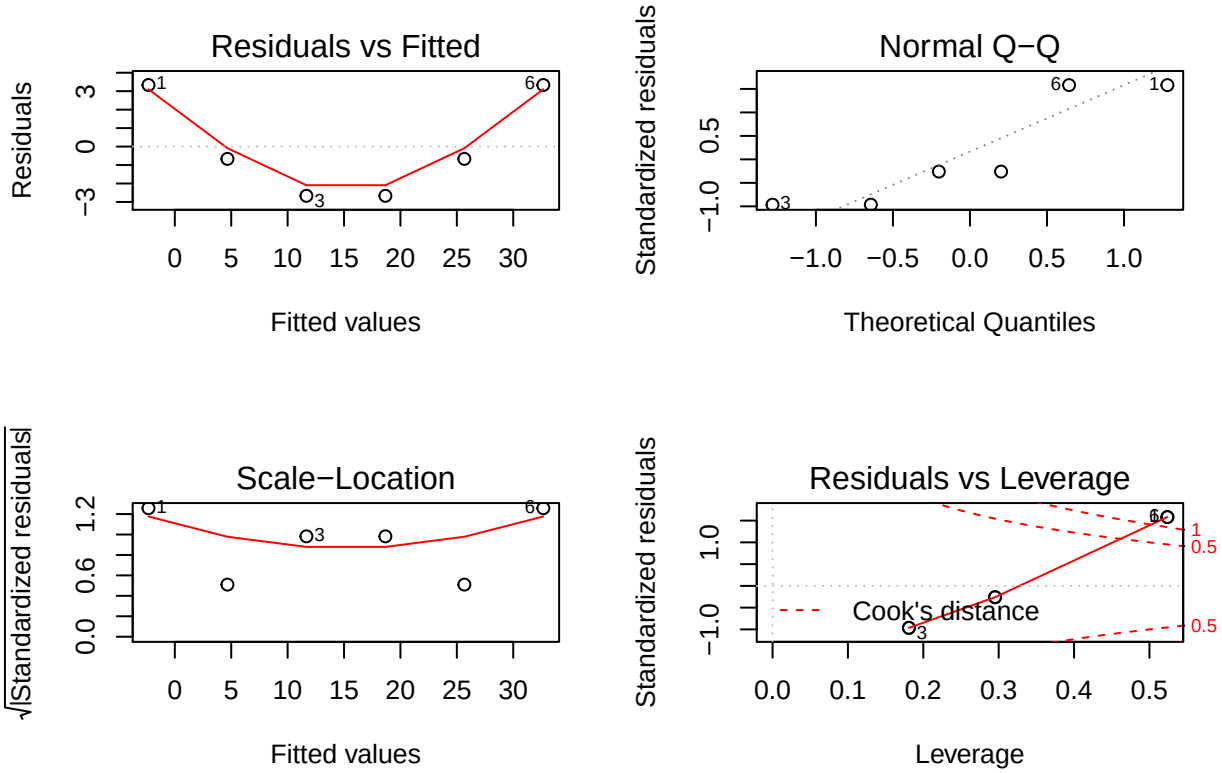
```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##      -9.333        7.000
```

```
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
```

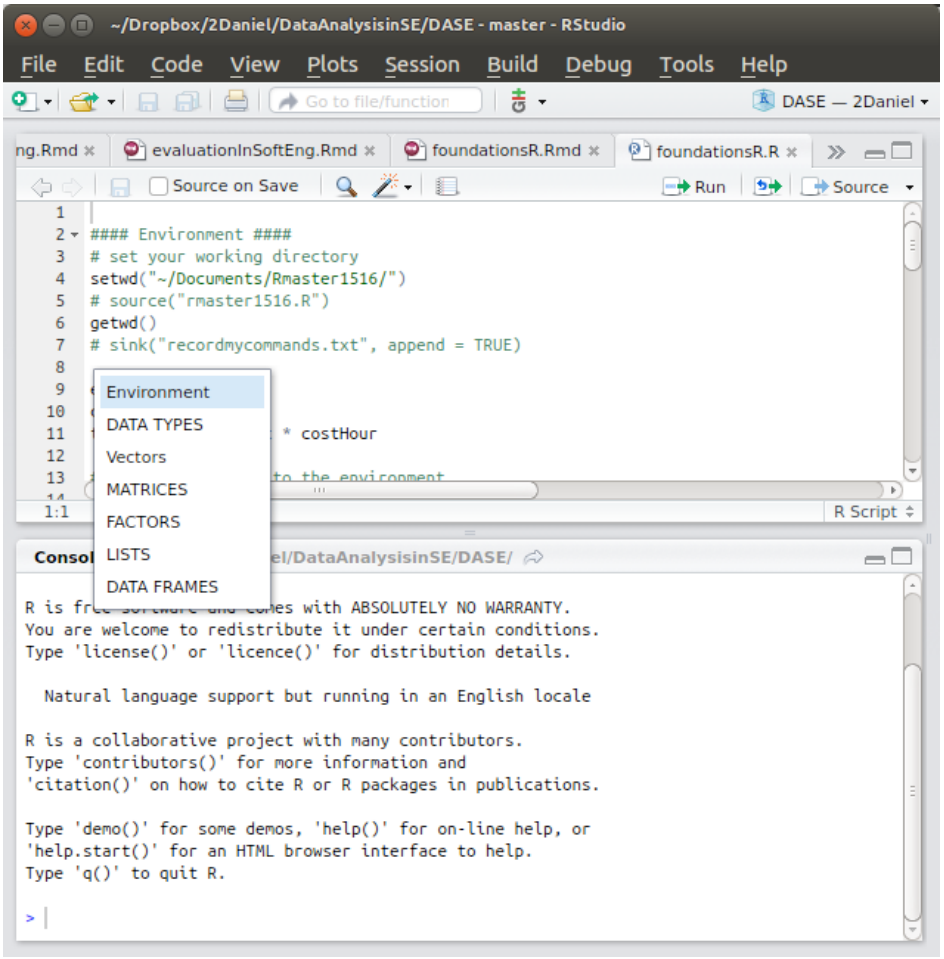


- R script. # A file with R commands `# comments source("filewithcommands.R") sink("recordmycommands.lis") savehistory()`
- From command line:
  - Rscript
  - Rscript file with `-e` (e.g. `Rscript -e 2+2`)
  - To exit R: `quit()`
- Variables
- Operators
  - assign operator `<-`
  - sequence operator, 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



Working directories:

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

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

```
# 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
```

```
# 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
```

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

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

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

It is done in all elements. 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!
```

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 sequenceswith  : 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

```
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)
mymat3
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   10   -6   -7   -2
## [2,]    5   -4    0   -8
## [3,]   15    3    9   11
```

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

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

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

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

```
mymat3[3,]
```

```
## [1] 15  3  9 11
```

```
mymat3[,4]
```

```
## [1] -2 -8 11
```

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

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

```
mymat3[9]
```

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

```
## Subsetting multiple elements

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

```
## [1] 5 0
```

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

```
##      [,1] [,2] [,3]
## [1,]    5    0   -8
## [2,]   15    9   11
```

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

```
## c1 c3
##  5  0
```

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

```
## c1 c3
##  5  0
```

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

```
##    c1 c3 c4
## r2  5  0 -8
## r3 15  9 11
```

```
### 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
```

```
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 in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed.

```
personnel <- c("Analyst1", "ManagerL2", "Analyst1", "Analyst2",  "Boss", "ManagerL1", "ManagerL2", "Prog

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

```
##  [1] Analyst1    ManagerL2   Analyst1    Analyst2    Boss
##  [6] ManagerL1   ManagerL2   Programmer1 Programmer2 Programmer3
## [11] Designer1   Designer2   OtherStaff
```

```
## 11 Levels: Analyst1 Analyst2 Boss Designer1 Designer2 ... 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
personnel2
```

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

- '[' returns a list
- '[[' returns the list element
- '$' returns the content of that element in the list

```
c("R good times", 190, 5)
```

```
## [1] "R good times" "190"          "5"
```

```
song <- list("R good times", 190, 5)
is.list(song)
```

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

```
str(song)
```

```
## List of 3
##  $ : chr "R good times"
##  $ : num 190
##  $ : num 5
```

```
names(song) <- c("title", "duration", "track")
song
```

```
## $title
## [1] "R good times"
##
## $duration
## [1] 190
##
## $track
## [1] 5
```

```
song$title
```

```
## [1] "R good times"
```

```
song2 <- list(title="Good Friends",
              duration = 125,
              track = 2,
              rank = 6)

song3 <- list(title="Many Friends",
              duration = 125,
              track= 2,
              rank = 1,
              similar2 = song2)

song[1]
```

```
## $title
## [1] "R good times"
```

```
song$title
```

```
## [1] "R good times"
```

```
str(song[1])
```

```
## List of 1
##  $ title: chr "R good times"
```

```
song[[1]]
```

```
## [1] "R good times"
```

```
str(song[[1]])
```

```
##  chr "R good times"
```

```
song2[3]
```

```
## $track
## [1] 2
```

```
song3[5]  # a list
```

```
## $similar2
## $similar2$title
## [1] "Good Friends"
##
## $similar2$duration
## [1] 125
##
## $similar2$track
## [1] 2
##
## $similar2$rank
## [1] 6
```

```
str(song3[5])
```

```
## List of 1
##  $ similar2:List of 4
##   ..$ title   : chr "Good Friends"
##   ..$ duration: num 125
##   ..$ track   : num 2
##   ..$ rank    : num 6
```

```
song3[[5]]
```

```
## $title
## [1] "Good Friends"
##
## $duration
## [1] 125
##
## $track
## [1] 2
##
## $rank
## [1] 6
```

```
song3$similar2
```

```
## $title
```

```
## [1] "Good Friends"
##
## $duration
## [1] 125
##
## $track
## [1] 2
##
## $rank
## [1] 6
```

```
song[c(1,3)]
```

```
## $title
## [1] "R good times"
##
## $track
## [1] 5
```

```
str(song[c(1,3)])
```

```
## List of 2
##  $ title: chr "R good times"
##  $ track: num 5
```

```
result <- song[c(1,3)]
result[1]
```

```
## $title
## [1] "R good times"
```

```
result[[1]]
```

```
## [1] "R good times"
```

```
str(result)
```

```
## List of 2
##  $ title: chr "R good times"
##  $ track: num 5
```

```
result$title
```

```
## [1] "R good times"
```

```
result$track
```

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

```
# access with [[ to content
song3[[5]][[1]]
```

```
## [1] "Good Friends"
```

```
song3$similar2[[1]]
```

```
## [1] "Good Friends"
```

```
# Subsets
### subset by names
song[c("title", "track")]
```

```
## $title
## [1] "R good times"
##
## $track
## [1] 5
```

```
song3["similar2"]
```

```
## $similar2
## $similar2$title
## [1] "Good Friends"
##
## $similar2$duration
## [1] 125
##
## $similar2$track
## [1] 2
##
## $similar2$rank
## [1] 6
```

```
resultsimilar <- song3["similar2"]
str(resultsimilar)
```

```
## List of 1
##  $ similar2:List of 4
##   ..$ title   : chr "Good Friends"
##   ..$ duration: num 125
##   ..$ track   : num 2
##   ..$ rank    : num 6
```

```
resultsimilar1 <-song3[["similar2"]]
str(resultsimilar1)
```

```
## List of 4
##  $ title   : chr "Good Friends"
##  $ duration: num 125
##  $ track   : num 2
##  $ rank    : num 6
```

```
resultsimilar1$title
```

```
## [1] "Good Friends"
```

```
# subset by logicals
song[c(TRUE, FALSE, TRUE, FALSE)]
```

```
## $title
## [1] "R good times"
##
## $track
## [1] 5
```

```
result3 <- song[c(TRUE, FALSE, TRUE, FALSE)]  # is a list of two elements

# extending the list
shared <- c("Hillary", "Javi", "Mikel", "Patty")
```

```
song3$shared <- shared
str(song3)
```

```
## List of 6
##  $ title   : chr "Many Friends"
##  $ duration: num 125
##  $ track   : num 2
##  $ rank    : num 1
##  $ similar2:List of 4
##   ..$ title   : chr "Good Friends"
##   ..$ duration: num 125
##   ..$ track   : num 2
##   ..$ rank    : num 6
##  $ shared  : chr [1:4] "Hillary" "Javi" "Mikel" "Patty"
```

```
cities <- list("Bilbao", "New York", "Tartu")
song3[["cities"]] <- cities
str(song3)
```

```
## List of 7
##  $ title   : chr "Many Friends"
##  $ duration: num 125
##  $ track   : num 2
##  $ rank    : num 1
##  $ similar2:List of 4
##   ..$ title   : chr "Good Friends"
##   ..$ duration: num 125
##   ..$ track   : num 2
##   ..$ rank    : num 6
##  $ shared  : chr [1:4] "Hillary" "Javi" "Mikel" "Patty"
##  $ cities  :List of 3
##   ..$ : chr "Bilbao"
##   ..$ : chr "New York"
##   ..$ : chr "Tartu"
```

1.8 Data frames

```
thenames <- c("Ane", "Mike", "Xabi", "Viktoria", "Edurne")
ages <- c(44, 20, 33, 15, 65)
employee <- c(FALSE, FALSE, TRUE, TRUE, FALSE)

mydataframe <- data.frame(thenames, ages, employee)
mydataframe
```

```
##   thenames ages employee
## 1      Ane   44    FALSE
## 2     Mike   20    FALSE
## 3     Xabi   33     TRUE
## 4 Viktoria   15     TRUE
## 5   Edurne   65    FALSE
```

```
names(mydataframe) <- c("FirstName", "Age", "Employee")
str(mydataframe)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ FirstName: Factor w/ 5 levels "Ane","Edurne",..: 1 3 5 4 2
##  $ Age      : num  44 20 33 15 65
##  $ Employee : logi  FALSE FALSE TRUE TRUE FALSE
```

```
#strings are not factors!

mydataframe <- data.frame(thenames, ages, employee,
                          stringsAsFactors=FALSE)
names(mydataframe) <- c("FirstName", "Age", "Employee")
str(mydataframe)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ FirstName: chr  "Ane" "Mike" "Xabi" "Viktoria" ...
##  $ Age      : num  44 20 33 15 65
##  $ Employee : logi  FALSE FALSE TRUE TRUE FALSE
```

```
# subset data frame

mydataframe[4,2]
```

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

```
mydataframe[4, "Age"]
```

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

```
mydataframe[, "FirstName"]
```

```
## [1] "Ane"      "Mike"     "Xabi"     "Viktoria" "Edurne"
```

```
mydataframe[c(2,5), c("Age", "Employee")]
```

```
##   Age Employee
## 2  20    FALSE
## 5  65    FALSE
```

```
matfromframe <- as.matrix(mydataframe[c(2,5), c("Age", "Employee")])
str(matfromframe)
```

```
##  num [1:2, 1:2] 20 65 0 0
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:2] "2" "5"
##   ..$ : chr [1:2] "Age" "Employee"
```

```
mydataframe[3]
```

```
##   Employee
## 1    FALSE
## 2    FALSE
## 3     TRUE
## 4     TRUE
## 5    FALSE
```

```
# convert to vector
mydf0 <- mydataframe[3] #data.frame
```

```
str(mydf0)
```

```
## 'data.frame':    5 obs. of  1 variable:
##  $ Employee: logi  FALSE FALSE TRUE TRUE FALSE
```

```
myvec <- mydataframe[[3]]  #vector
str(myvec)
```

```
##  logi [1:5] FALSE FALSE TRUE TRUE FALSE
```

```
mydf0asvec <- as.vector(mydataframe[3]) # but it doesn't work . Use [[]]
str(mydf0asvec)
```

```
## 'data.frame':    5 obs. of  1 variable:
##  $ Employee: logi  FALSE FALSE TRUE TRUE FALSE
```

```
mydf0asvec <- as.vector(mydataframe[[3]])
str(mydf0asvec)
```

```
##  logi [1:5] FALSE FALSE TRUE TRUE FALSE
```

```
# add column
height <- c(166, 165, 158, 176, 199)
weight <- c(66, 77, 99, 88, 109)
mydataframe$height <- height
mydataframe[["weight"]] <- weight
mydataframe
```

```
##   FirstName Age Employee height weight
## 1       Ane  44    FALSE    166     66
## 2      Mike  20    FALSE    165     77
## 3      Xabi  33     TRUE    158     99
## 4  Viktoria  15     TRUE    176     88
## 5    Edurne  65    FALSE    199    109
```

```
# add a column

birthplace <- c("Tallinn", "London", "Donostia", "Paris", "New York")

mydataframe <- cbind(mydataframe, birthplace)
mydataframe
```

```
##   FirstName Age Employee height weight birthplace
## 1       Ane  44    FALSE    166     66    Tallinn
## 2      Mike  20    FALSE    165     77     London
## 3      Xabi  33     TRUE    158     99   Donostia
## 4  Viktoria  15     TRUE    176     88      Paris
## 5    Edurne  65    FALSE    199    109   New York
```

```
# add a row

anton <- data.frame(FirstName = "Anton", Age = 77, Employee=TRUE, height= 170, weight = 65, birthplace =
mydataframe <- rbind (mydataframe, anton)
mydataframe
```

```
##   FirstName Age Employee height weight birthplace
## 1       Ane  44    FALSE    166     66    Tallinn
## 2      Mike  20    FALSE    165     77     London
```

```
## 3      Xabi  33     TRUE    158     99   Donostia
## 4  Viktoria  15     TRUE    176     88      Paris
## 5    Edurne  65    FALSE    199    109   New York
## 6     Anton  77     TRUE    170     65  Amsterdam
```

```
# sorting

mydataframeSorted <- mydataframe[order(mydataframe$Age, decreasing = TRUE), ]  #all columns
mydataframeSorted
```

```
##   FirstName Age Employee height weight birthplace
## 6     Anton  77     TRUE    170     65  Amsterdam
## 5    Edurne  65    FALSE    199    109   New York
## 1       Ane  44    FALSE    166     66    Tallinn
## 3      Xabi  33     TRUE    158     99   Donostia
## 2      Mike  20    FALSE    165     77     London
## 4  Viktoria  15     TRUE    176     88      Paris
```

```
mydataframeSorted2 <- mydataframe[order(mydataframe$Age, decreasing = TRUE), c(1,2,6) ]
mydataframeSorted2
```

```
##   FirstName Age birthplace
## 6     Anton  77  Amsterdam
## 5    Edurne  65   New York
## 1       Ane  44    Tallinn
## 3      Xabi  33   Donostia
## 2      Mike  20     London
## 4  Viktoria  15      Paris
```

1.9 Reading Data

```
library(foreign)
isbsg <- read.arff("datasets/effortEstimation/isbsg10teaser.arff")

mydataISBSG <- isbsg[, c("FS", "N_effort")]

str(mydataISBSG)
```

```
## 'data.frame':    37 obs. of  2 variables:
##  $ FS      : num  225 599 333 748 158 427 461 257 115 116 ...
##  $ N_effort: num  1856 10960 5661 1518 3670 ...
```

1.10 Plots

There are several graphic packages that are recommended, in particular `ggplot`. However, there is some basic support in the R base for graphics. The following Figure 1.1 shows a simple plot.

```
plot(mydataISBSG$FS, mydataISBSG$N_effort)
```

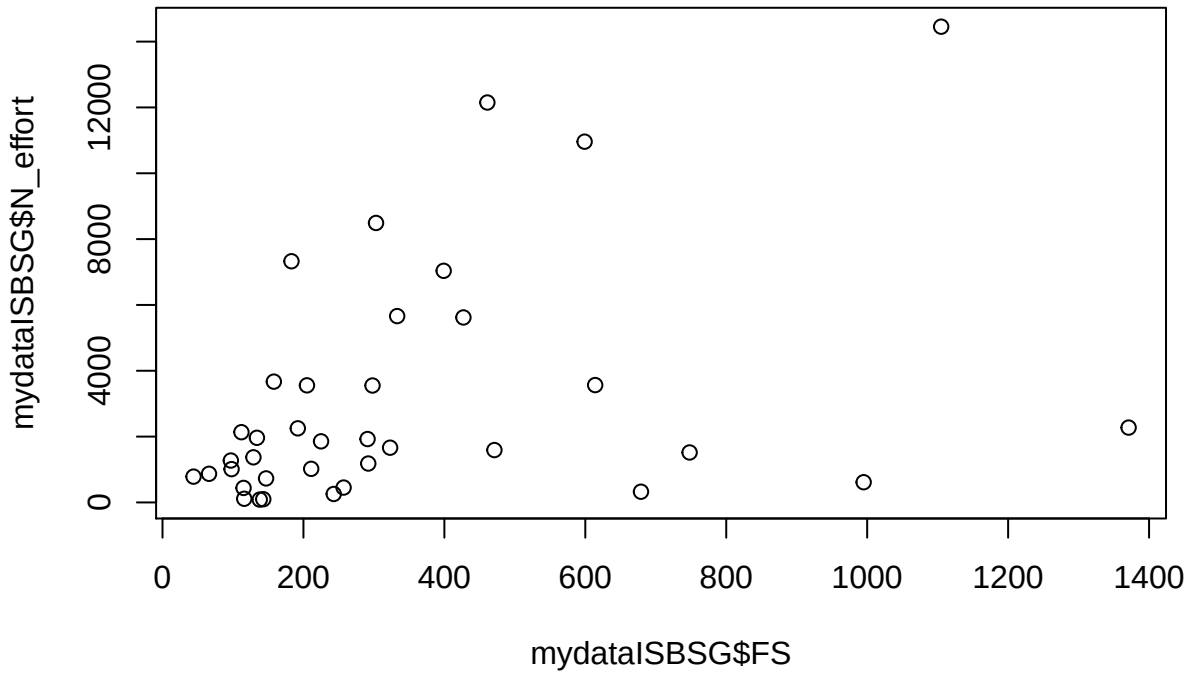


Figure 1.1: Simple plot

1.11 Flow of Control

Ifelse:

```
library(foreign)
kc1 <- read.arff("datasets/defectPred/D1/KC1.arff")
kc1$Defective <- ifelse(kc1$Defective == "Y", 1, 0)
head(kc1, 1)
```

```
##   LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT LOC_COMMENTS
## 1         0            1                    0            0
##   CYCLOMATIC_COMPLEXITY DESIGN_COMPLEXITY ESSENTIAL_COMPLEXITY
## 1                     1                 1                    1
##   LOC_EXECUTABLE HALSTEAD_CONTENT HALSTEAD_DIFFICULTY HALSTEAD_EFFORT
## 1              3            11.58                2.67           82.35
##   HALSTEAD_ERROR_EST HALSTEAD_LENGTH HALSTEAD_LEVEL HALSTEAD_PROG_TIME
## 1               0.01              11           0.38               4.57
##   HALSTEAD_VOLUME NUM_OPERANDS NUM_OPERATORS NUM_UNIQUE_OPERANDS
## 1           30.88            4             7                   3
##   NUM_UNIQUE_OPERATORS LOC_TOTAL Defective
## 1                    4         5         0
```

1.12 Rattle

There is graphical interface, Rattle, that allow us to perform some data mining tasks with R (Williams, 2011).

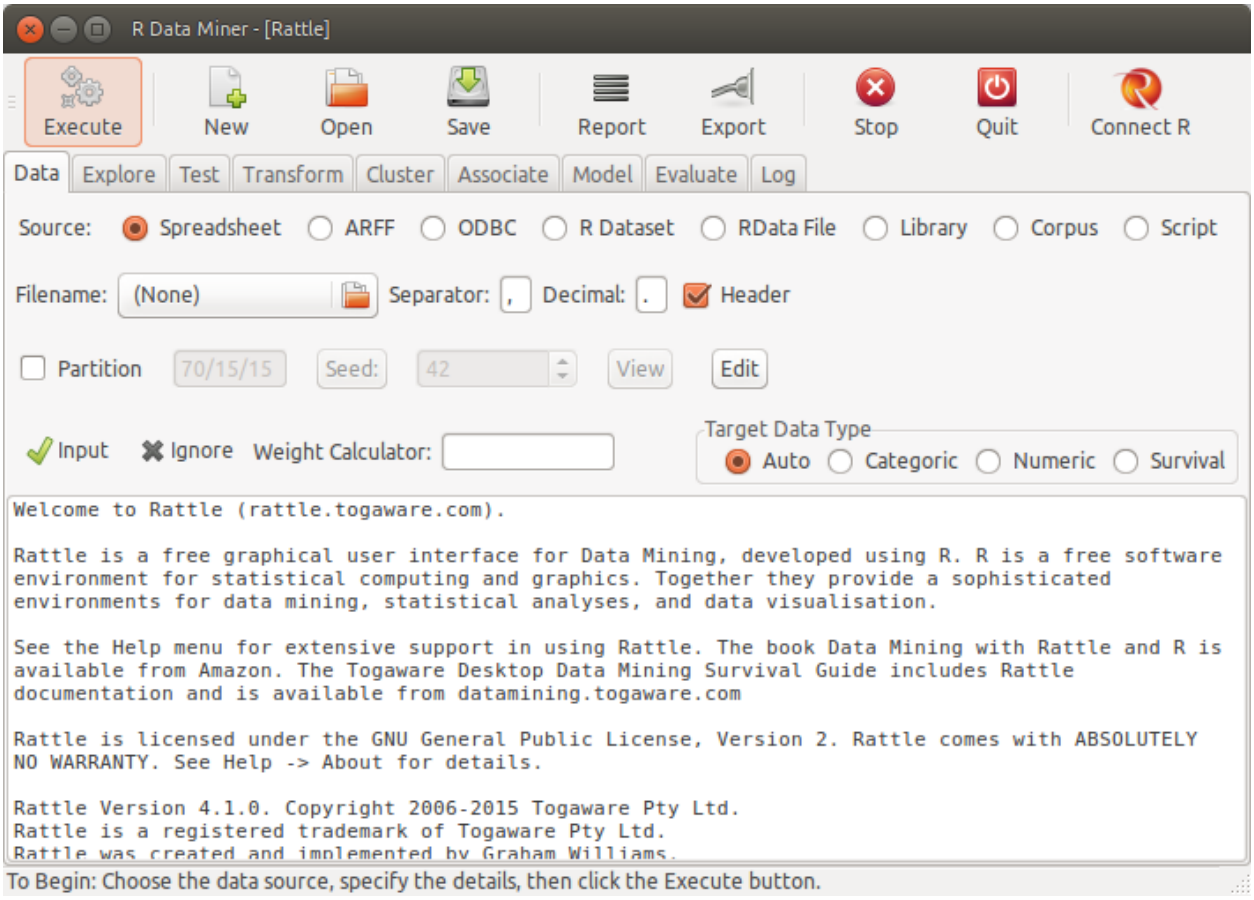


Figure 1.2: Rattle: GUI for Data mining with R

Part II

Introduction to Data Mining

We will deal with extracting information from data, either for estimation, defect prediction, planning, etc. We will provide an overview of data analsyis using different techniques.

Chapter 2

What is Data Mining / Knowledge Discovery in Databases (KDD)

The non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data (Fayyad et al., 1996)

The Cross Industry Process for Data Mining (CRISP-DM) also provides a common and well-developed framework for delivering data mining projects identifying six steps (Shearer, 2000):

1. Problem Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Deployment

2.1 The Aim of Data Analysis and Statistical Learning

- The aim of any data analysis is to **understand the data**
- and to build models for making predictions and estimating future events based on past data
- and to make statistical inferences from our data.
- We may want to test different hypothesis on the data
- We want to generate conclusions about the population where our sample data comes from
- Most probably we are interested in building a model for quality, time, defects or effort prediction

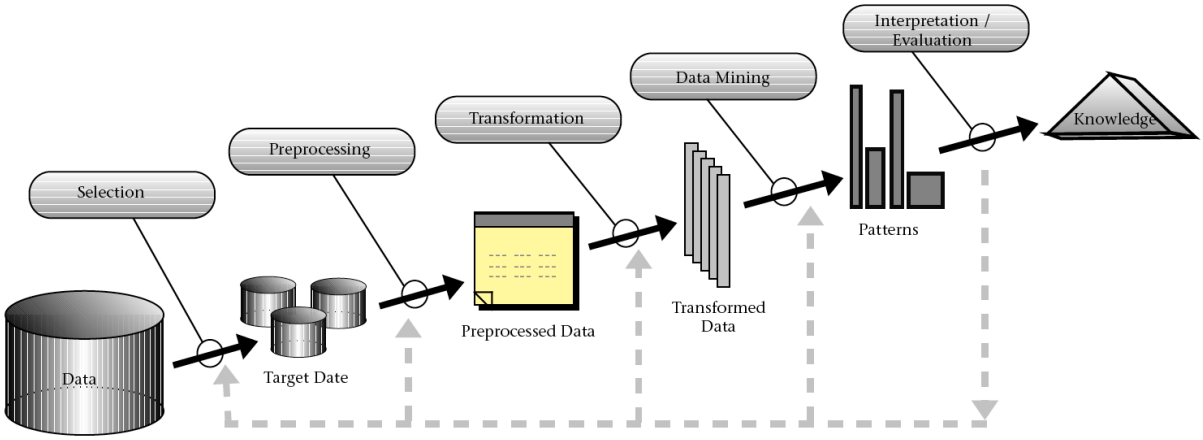
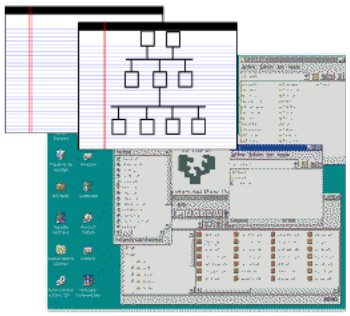


Figure 2.1: KDD Process

Figure 2.2: CRISP-DM (Wikipedia)



Parameters, data collected, previous projects, etc.

- We want to find a function $f()$, that given $X1, X2, ...$ computes $Y = f(X1, X2, ..., Xn)$

2.2 Basic References

Generic books about statistics:

- John Verzani, *simpleR - Using R for Introductory Statistics*
- Peter Dalgaard, *Introductory Statistics with R*, 2nd Edt., Springer, 2008
- Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, 2013
- Geoff Cumming, *Understanding the New Statistics: Effect Sizes, Confidence Intervals, and Meta-Analysis*, Routledge, New York, 2012

2.3 Data Mining with R

- Graham Williams, *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*, Springer 2011

  Also the author maintains a Web site: http://rattle.togaware.com/
- Luis Torgo, *Data Mining with R: Learning with Case Studies*, Chapman and Hall/CRC, 2010
- http://www.rdatamining.com/

2.4 Data Mining with Weka

Weka is another popular framework in Java:

- Ian Witten, Eibe Frank, Mark Hall, Christopher J. Pal, Data Mining: Practical Machine Learning Tools and Techniques (4th Edt), Morgan Kaufmann, 2016, ISBN: 978-0128042915

Part III

Data Sources and Metrics and Standards in Software Engineering Defect Prediction

Chapter 3

Data Sources in Software Engineering

We classify this trail in the following categories:

- *Source code* can be studied to measure its properties, such as size or complexity.
- *Source Code Management Systems* (SCM) make it possible to store all the changes that the different source code files undergo during the project. Also, SCM systems allow for work to be done in parallel by different developers over the same source code tree. Every change recorded in the system is accompanied with meta-information (author, date, reason for the change, etc) that can be used for research purposes.
- *Issue or Bug tracking systems* (ITS). Bugs, defects and user requests are managed in ISTs, where users and developers can fill tickets with a description of a defect found, or a desired new functionality. All the changes to the ticket are recorded in the system, and most of the systems also record the comments and communications among all the users and developers implied in the task.
- *Messages* between developers and users. In the case of free/open source software, the projects are open to the world, and the messages are archived in the form of mailing lists and social networks which can also be mined for research purposes. There are also some other open message systems, such as IRC or forums.
- *Meta-data about the projects.* As well as the low level information of the software processes, we can also find meta-data about the software projects which can be useful for research. This meta-data may include intended-audience, programming language, domain of application, license (in the case of open source), etc.
- *Usage data.* There are statistics about software downloads, logs from servers, software reviews, etc.

Types of information stored in the repositories:

- Meta-information about the project itself and the people that participated.
  - Low-level information
    - Mailing Lists (ML)
    - Bug Tracking Systems (BTS) or Project Tracker System (PTS)
    - Software Configuration Management Systems (SCM)
  - Processed information. For example project management information about the effort estimation and cost of the project.
- Whether the repository is public or not
- Single project vs. multiprojects. Whether the repository contains information of a single project with multiples versions or multiples projects and/or versions.

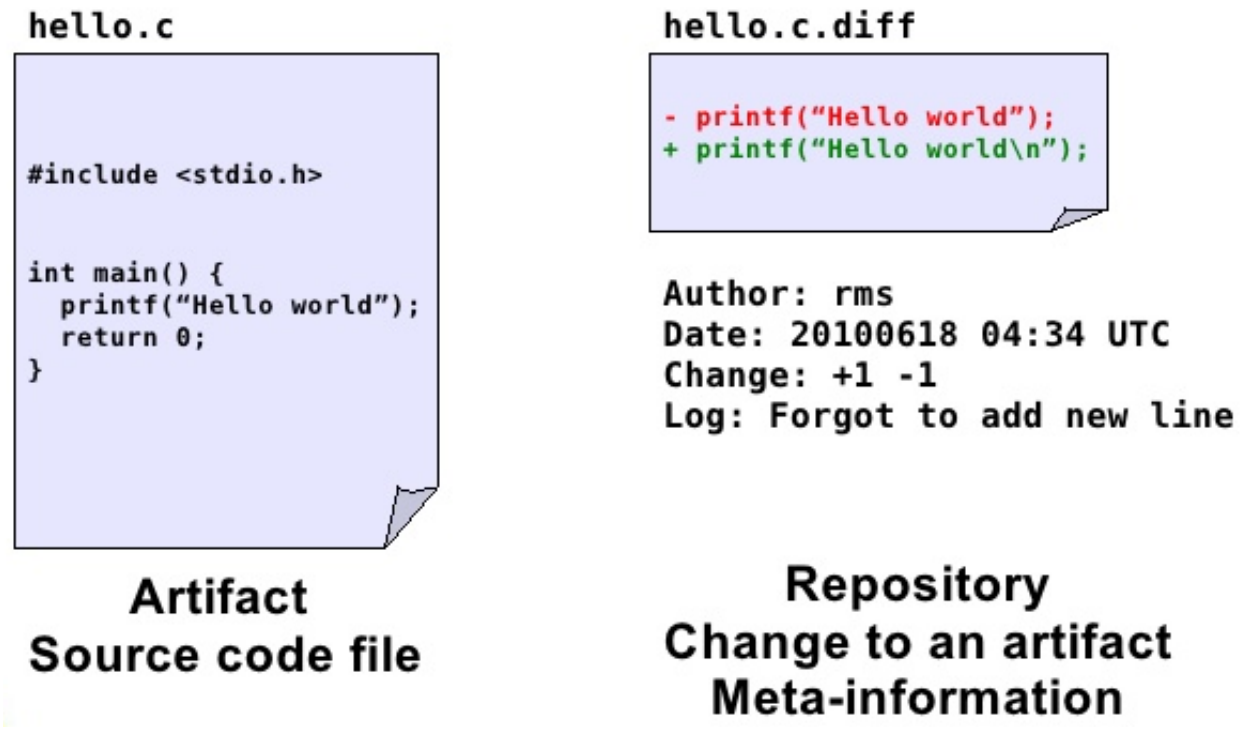


Figure 3.1: Metadata (source: Israel Herraiz)

- Type of content, open source or industrial projects
- Format in which the information is stored and formats or technologies for accessing the information:
  - Text. It can be just plain text, CSV (Comma Separated Values) files, Attribute-Relation File Format (ARFF) or its variants
  - Through databases. Downloading dumps of the database.
  - Remote access such as APIs of Web services or REST

Chapter 4

Repositories

There is a number of open research repositories in Software Engineering. Among them:

- PROMISE (PRedictOr Models In Software Engineering): http://openscience.us/repo/
- Finding Faults using Ensemble Learners (ELFF) (Shippey et al., 2016) http://www.elff.org.uk/
- FLOSSMole (Howison et al., 2006) http://flossmole.org/
- FLOSSMetrics (Herraiz et al., 2009): http://flossmetrics.org/
- Qualitas Corpus (QC) (Tempero et al., 2010): http://qualitascorpus.com/
- Sourcerer Project (Linstead et al., 2009): http://sourcerer.ics.uci.edu/
- Ultimate Debian Database (UDD) (Nussbaum and Zacchiroli, 2010) http://udd.debian.org/
- SourceForge Research Data Archive (SRDA) (Van Antwerp and Madey, 2008) http://zerlot.cse.nd.edu/
- SECOLD (Source code ECOsystem Linked Data): http://www.secold.org/
- Software-artifact Infrastructure Repository (SIR) [http://sir.unl.edu]
- OpenHub: https://www.openhub.net/

Not openly available (and mainly for effort estimation):

- The International Software Benchmarking Standards Group (ISBSG) http://www.isbsg.org/
- TukuTuku http://www.metriq.biz/tukutuku/

Some papers and publications/theses that have been used in the literature:

- Helix Data Set (Vasa, 2010): http://www.ict.swin.edu.au/research/projects/helix/
- Bug Prediction Dataset (BPD) (D'Ambros et al., 2010, D'Ambros et al. (2011)): http://bug.inf.usi.ch/
- Eclipse Bug Data (EBD) (Zimmermann et al., 2007, Nagappan et al. (2012)): http://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/

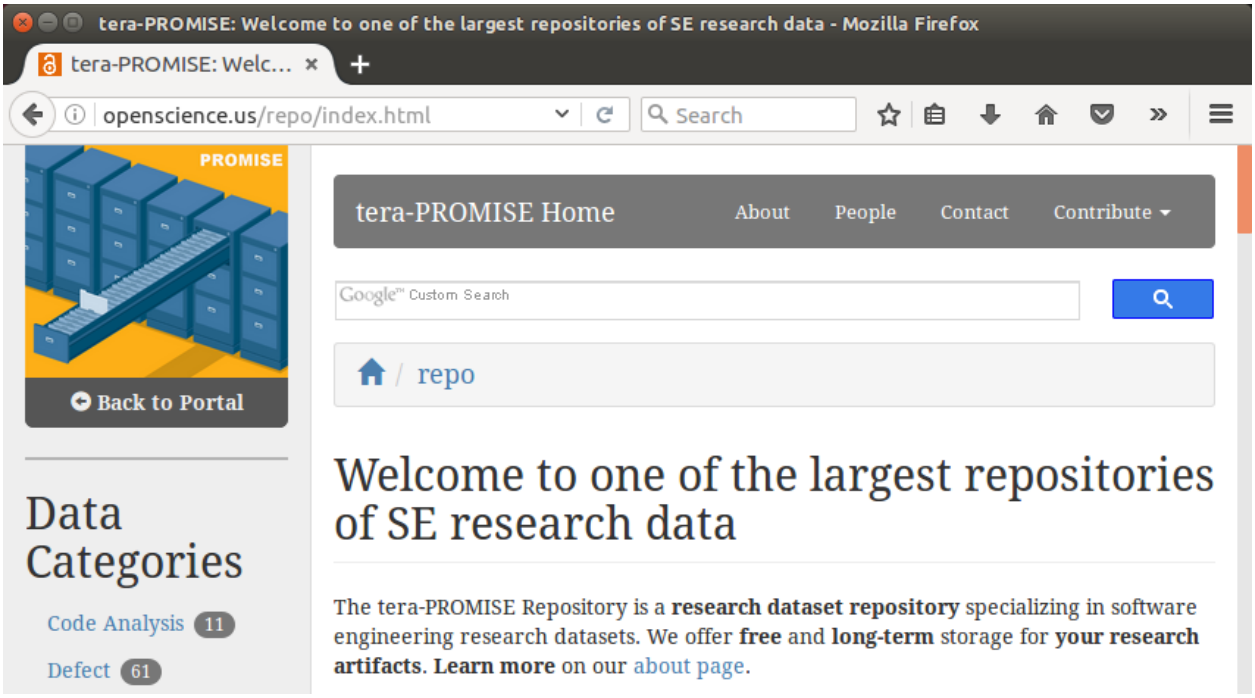


Figure 4.1: Promise Repository

Chapter 5

Open Tools/Dashboards to extract data

Process to extract data:

Within the open source community, several toolkits allow us to extract data that can be used to explore projects:

Metrics Grimoire http://metricsgrimoire.github.io/

SonarQube http://www.sonarqube.org/

CKJM (OO Metrics tool) http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/

Collects a large number of object-oriented metrics from code.

5.1 Issues

There are problems such as different tools report different values for the same metric (Lincke et al., 2008)

It is well-know that the NASA datasets have some problems:

- (Gray et al., 2011) The misuse of the NASA metrics data program data sets for automated software defect prediction
- (Shepperd et al., 2013) Data Quality: Some Comments on the NASA Software Defect Datasets

5.2 Effort Estimation Data in Software Engineering

It is worth highlighting the case of software effort estimation datasets with their peculiarities. First, most effort estimation datasets used in the literature are scattered through research papers with the exception of a few kept in the PROMISE repository. Mair et al (2005) also have analysed available datasets in the field of cost estimation identifying 65 different datasets in 50 papers.

Figure 5.1: Process

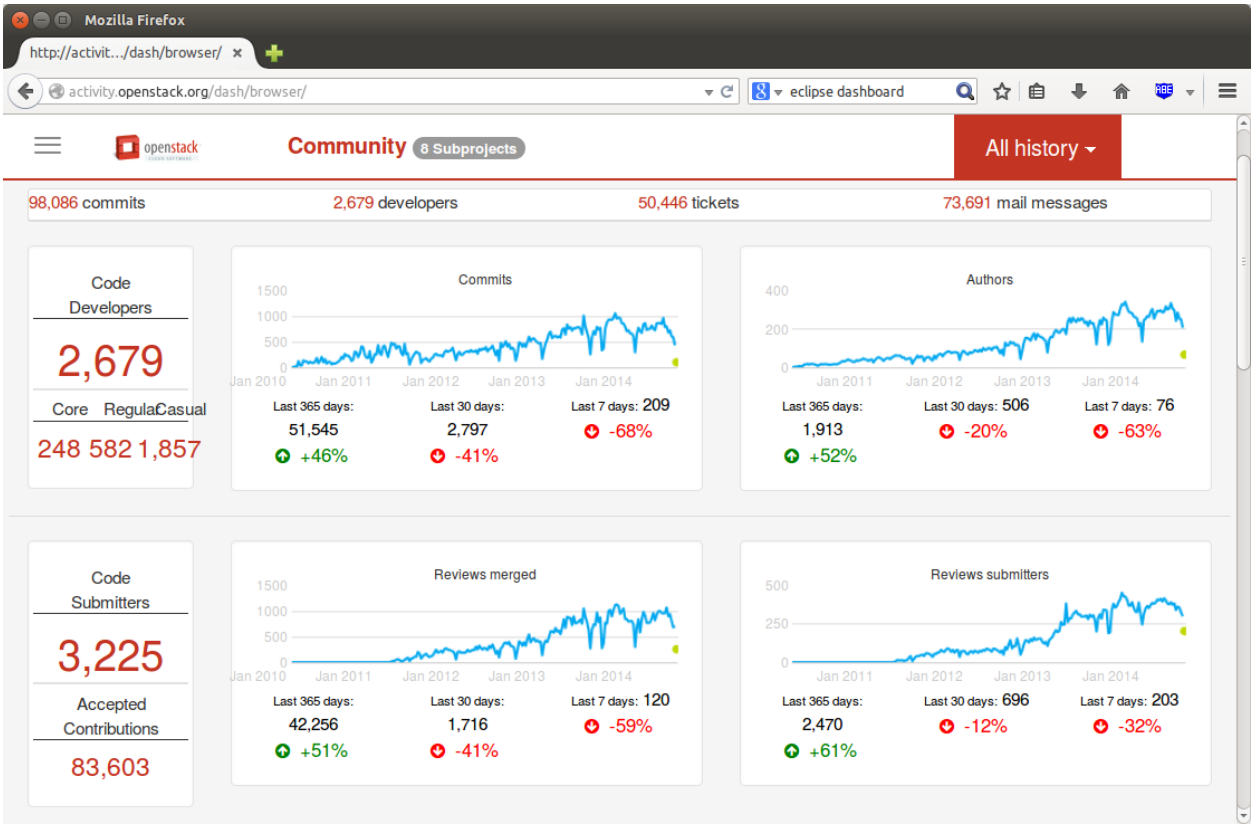


Figure 5.2: Grimoire

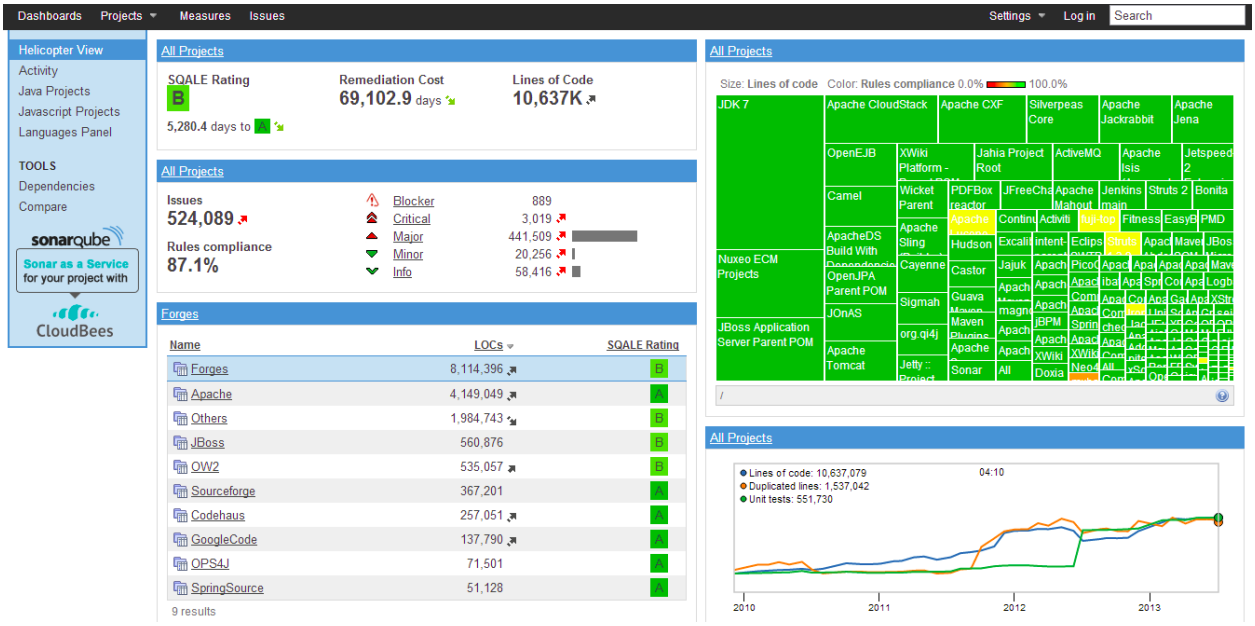


Figure 5.3: SonarQube

Second, their size is very small with the exception of ISBSG repository discussed previously which a small sample is available through PROMISE and the China dataset with 499 instances.

Third, some can be quite old in a context and time that is not applicable to current development environments. The authors noted that the oldest datasets (COCOMO, Desharnais, Kemerer and Albrecht and Gaffney) tend to be the most studied ones and a subset of the most relevant ones. Also, from the artificial intelligence or data mining point of view effort estimation has been mainly tackled with different types of regression techniques and more recently with techniques which are also typically considered under the umbrella of data mining techniques. However, as the number of examples per dataset is increasing, other machine learning techniques are also being studied (e.g.: Dejaeger et al (2012) report on a comparison of several machine learning techniques to effort estimation with only 5 out the 9 used datasets publicly available). From the data mining point of view, the small number of instances hinders the application of machine learning techniques.

However, software effort and cost estimation still remain one of the main challenges in software engineering and have attracted a great deal of interest by many researchers (2007). For example, there are continuous analyses of whether software development follows economies or diseconomies of scale (see (Dolado, 2001, Banker et al. (1994),Kitchenham (2002))).

Next Table 5.1 (following Mair et al (2005)) shows the most open cost/effort datasets available in the literature with their main reference.

Table 5.1: Effort Estimation Dataset from articles

Reference	Instances	Attributes
Abran and Robillard (1996)	21	31
Albrecht-Gaffney (1983)	24	7
Bailey and Basili (1981)	18	9
Belady and Lehman (1979)	33	
Boehm (aka COCOMO Dataset) (1981)	63	43
China dataset[1]	499	18
Desharnais (1988)	61	10
Dolado (1997)	24	7
Hastings and Sajeev (2001)	8	14
Heiat and Heiat (Heiat and Heiat, 1997)	35	4
Jeffery and Stathis (1996)	17	7
Jorgensen (2004)	47	4
Jorgensen et al. (2003)	20	4
Kemerer (1987)	15	5
Kitchenham (Mermaid 2) (2002)	30	5
Kitchenham et al. (CSC) (2002)	145	9
Kitchenham and Taylor (ICL) (1985)	10	6
Kitchenham and Taylor (BT System X) (1985)	10	3
Kitchenham and Taylor (BT Software Houses) (1985)	12	6
Li et al.(USP05) (2007)[2]	202	16
Mišić and Tevsić (1998)	6	16
Maxwell (Dev Effort) (2002)	63	32
Maxwell (Maintenance Eff) (2002)	67	28
Miyazaki et al. (1994)	47	9
Moser et al. (1999)	37	4
Shepperd and Cartwright (Shepperd and Cartwright, 2001)	39	3
Shepperd and Schofield (Telecom 1) (1997)	18	5

[1]Donated through PROMISE.
[2]Only a subset of the data in the paper, the complete dataset is donated through PROMISE

Reference	Instances	Attributes
Schofield (real-time 1) (1998, Shepperd and Schofield (1997))	21	4
Schofield (Mermaid) (1998)	30	18
Schofield (Finnish) (1998)	39	30
Schofield (Hughes) (1998)	33	14
Woodfield et al. (1981)	63	8

Part IV

Exploratory and Descriptive Data analysis

Chapter 6

Exploratory Data Analysis

6.1 Descriptive statistics

The first task with any dataset is to characterise it in terms of summary statistics and graphics.

Displaying information graphically will help us to identify the main characteristics of the data. To describe a distribution we often want to know where it is centered and and what the spread is (mean, median, quantiles)

6.2 Basic Plots

- *Histogram* defines a sequence of breaks and then counts the number of observations in the bins formed by the breaks.
- *Boxplot* used to summarize data succinctly, quickly displaying if the data is symmetric or has suspected outliers.
- *Q-Q plot* is used to determine if the data is close to being normally distributed. The quantiles of the standard normal distribution is represented by a straight line. The normality of the data can be evaluated by observing the extent in which the points appear on the line. When the data is normally distributed around the mean, then the mean and the median should be equal.
- *Scatterplot* provides a graphical view of the relationship between two sets of numbers: one numerical variable against another.
- *Kernel Density* plot visualizes the underlying distribution of a variable. Kernel density estimatiion is a non-parametric method of estimating the probability density function of continuous random variable. It helps to identify the distribution of the variable.
- *Violin plot* is a combination of a boxplot and a kernel density plot.

6.3 Normality

- A normal distribution is an arrangement of a data set in which most values cluster in the middle of the range
- A graphical representation of a normal distribution is sometimes called a *bell curve* because of its shape.
- Many procedures in statistics are based on this property. *Parametric* procedures require the normality property.

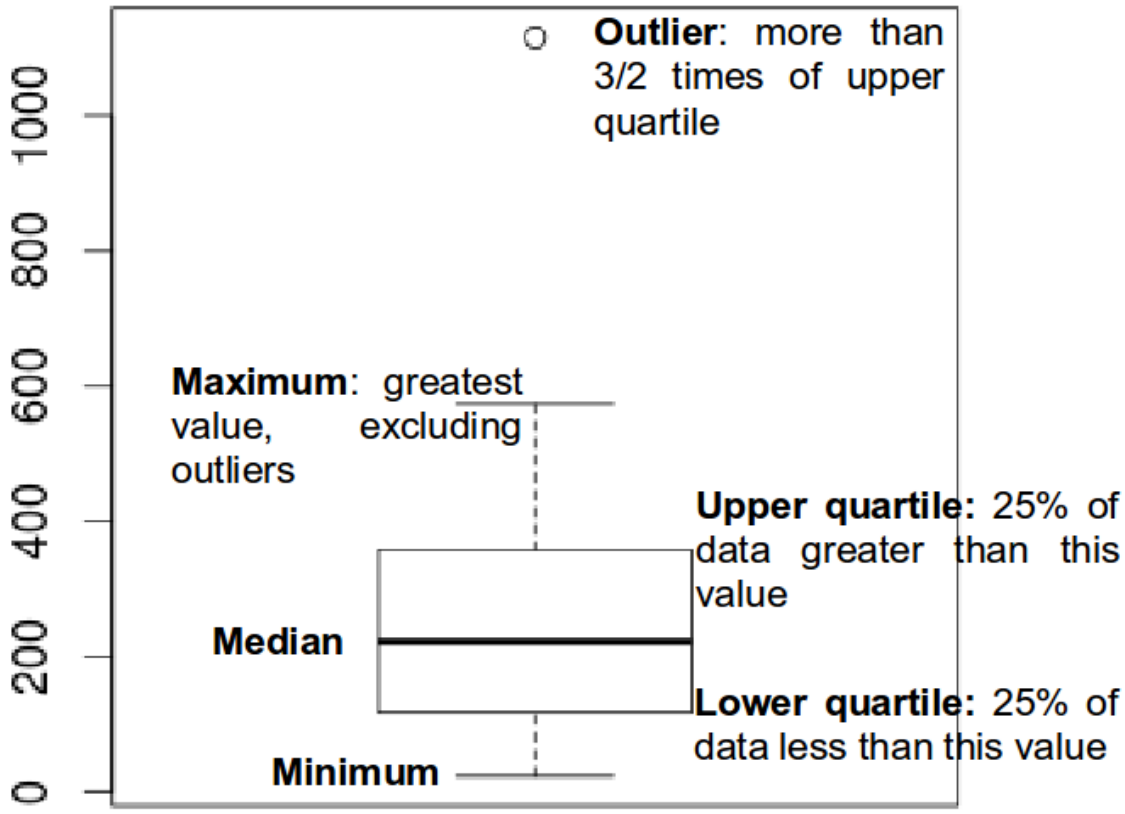


Figure 6.1: Boxplot description

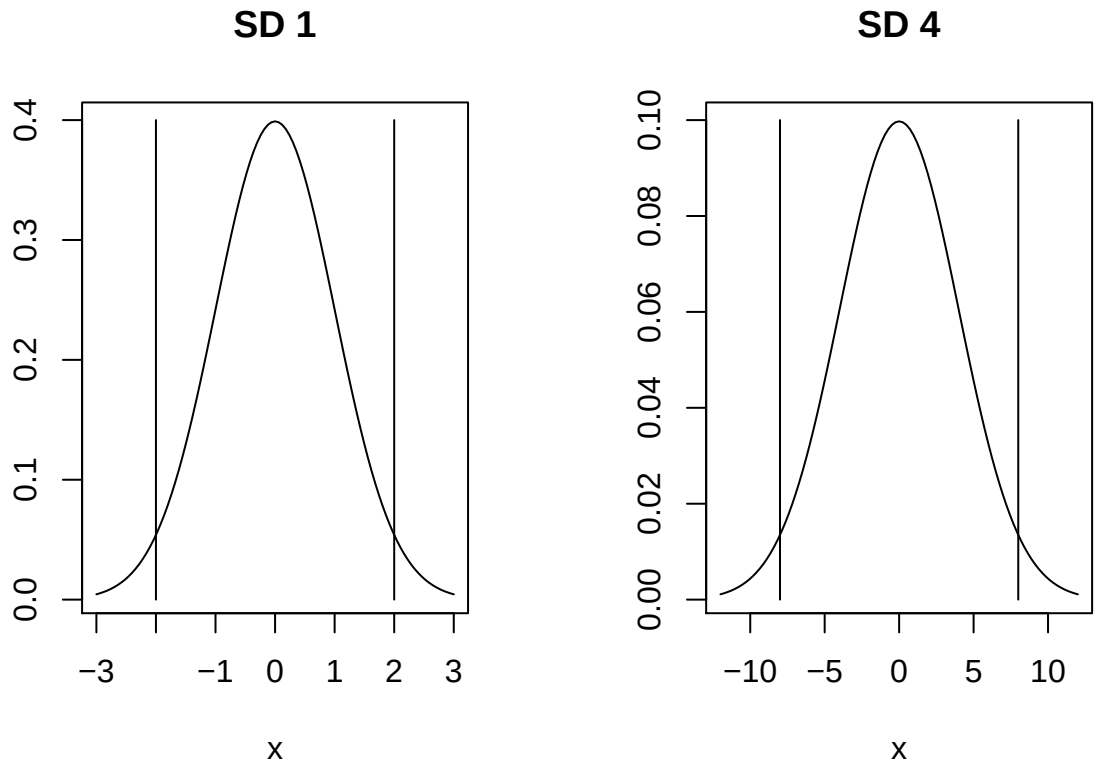


Figure 6.2: Plot exaple of the area within 2 and 4SD of the mean respectively

- In a normal distribution about 95% of the probability lies within 2 Standard Deviations of the mean.
- Two examples: one population with mean 60 and the standard deviation of 1, and the other with mean 60 and $sd = 4$ (means shifted to 0)

```
# Area within 2SD of the mean
par(mfrow = c(1, 2))
plot(function(x) dnorm(x, mean = 0, sd = 1), xlim = c(-3, 3), main = "SD 1",
    xlab = "x", ylab = "", cex = 2)
segments(-2, 0, -2, 0.4)
segments(2, 0, 2, 0.4)
# Area within 4SD of the mean
plot(function(x) dnorm(x, mean = 0, sd = 4), xlim = c(-12, 12), main = "SD 4",
    xlab = "x", ylab = "", cex = 2)
segments(-8, 0, -8, 0.1)
segments(8, 0, 8, 0.1)
```

- if we sample from this population we get "another population":

```
sample.means <- rep(NA, 1000)
for (i in 1:1000) {
    sample.40 <- rnorm(40, mean = 60, sd = 4)
    # rnorm generates random numbers from normal distribution
    sample.means[i] <- mean(sample.40)
}
means40 <- mean(sample.means)
sd40 <- sd(sample.means)
means40
```

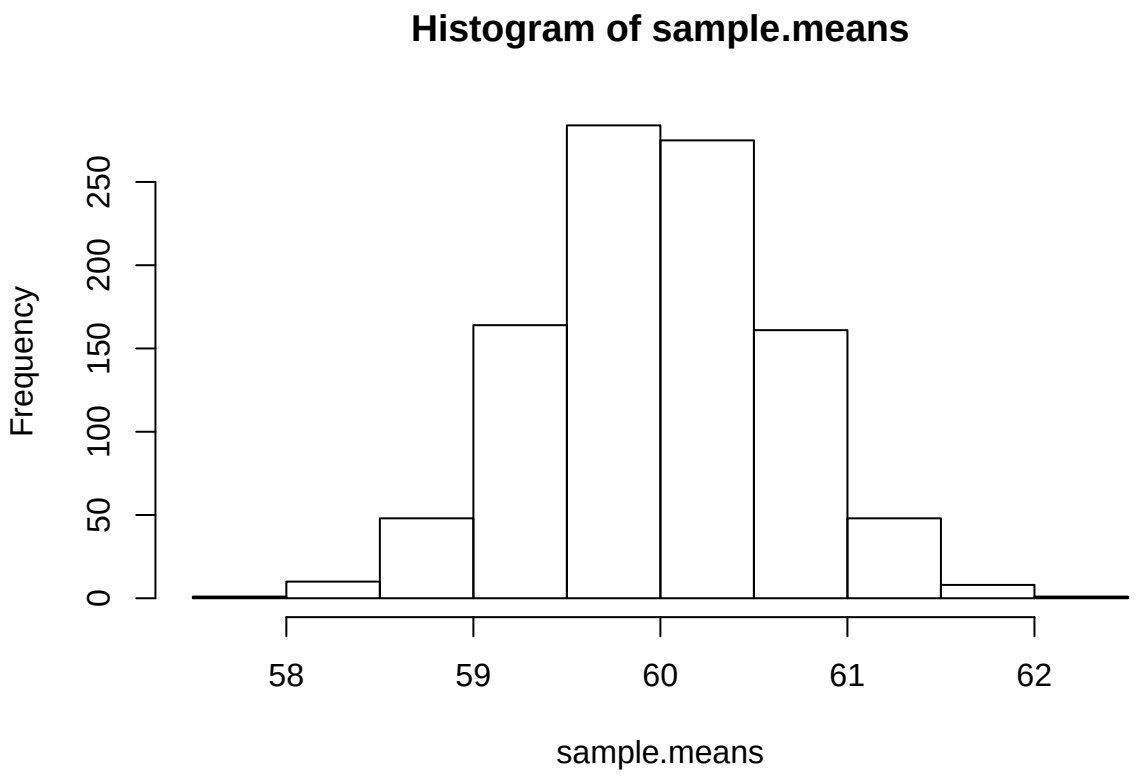


Figure 6.3: Sample means histogram

```
## [1] 59.99101
```

```
sd40
```

```
## [1] 0.6430586
```

- These sample means are another "population". The sampling distribution of the sample mean is normally distributed meaning that the "mean of a representative sample provides an estimate of the unknown population mean". This is shown in Figure 6.3

```
hist(sample.means)
```

6.4 Using a running Example to visualise the different plots

As a running exmple we do next:

1. Set the path to to the file
2. Read the *Telecom1* dataset and print out the summary statistics with the command `summary`

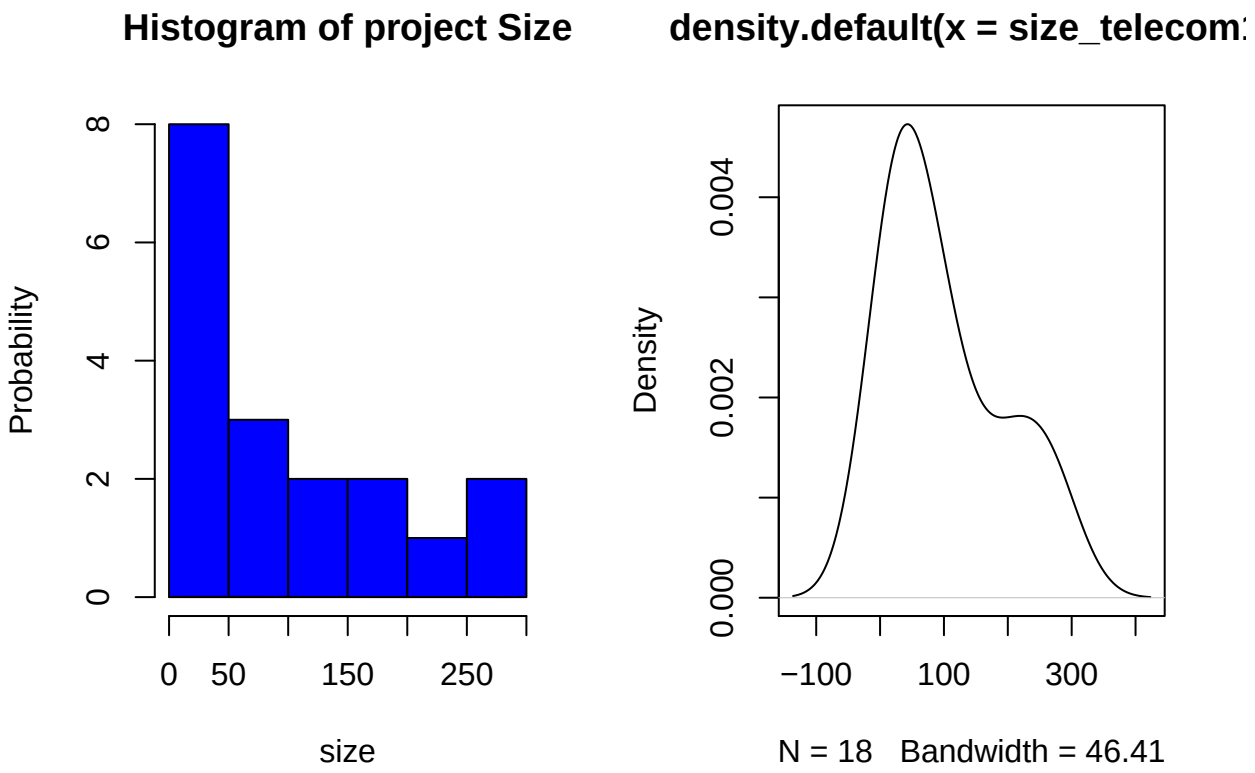
```
options(digits = 3)
telecom1 <- read.table("./datasets/effortEstimation/Telecom1.csv", sep = ",",
    header = TRUE, stringsAsFactors = FALSE, dec = ".")  #read data
summary(telecom1)
```

```
##       size           effort        EstTotal
##  Min.   :  3.0   Min.   :  24   Min.   : 30
##  1st Qu.: 37.2   1st Qu.: 119   1st Qu.:142
```

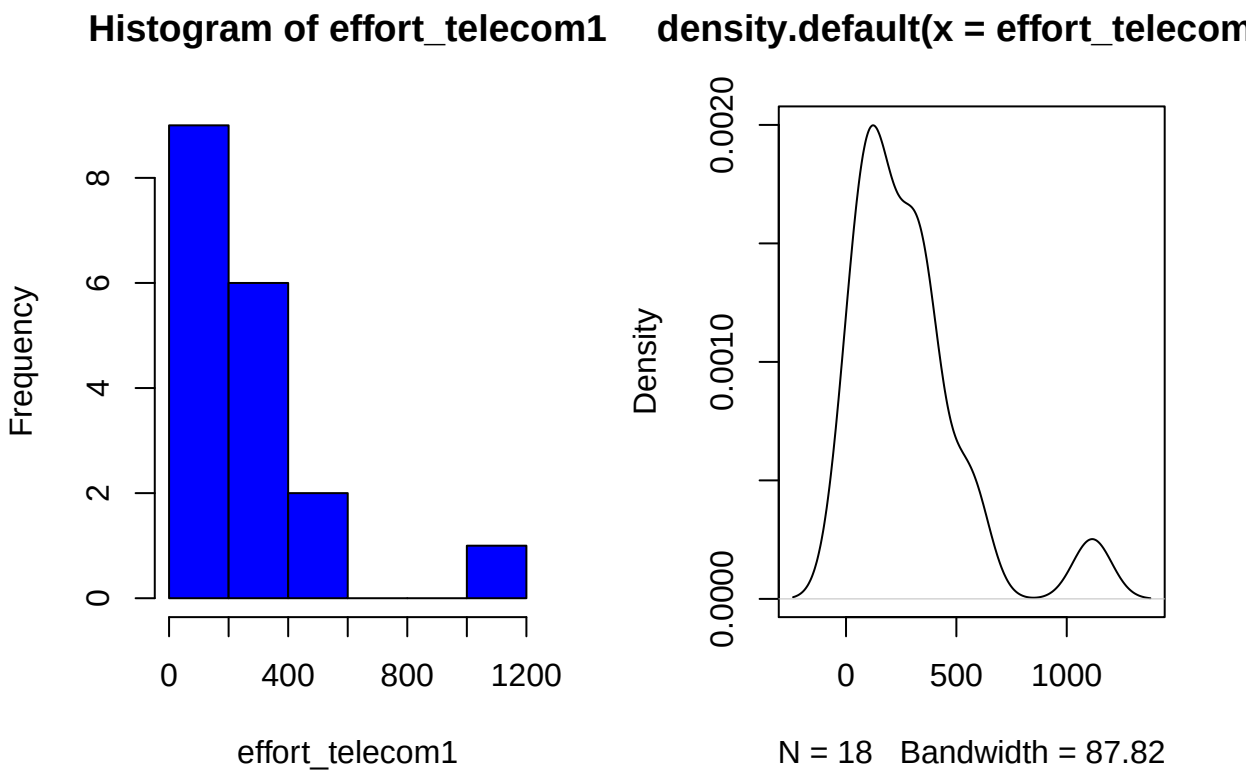
```
##  Median : 68.5   Median : 222   Median :289
##  Mean   :100.3   Mean   : 284   Mean   :320
##  3rd Qu.:164.0   3rd Qu.: 352   3rd Qu.:472
##  Max.   :284.0   Max.   :1116   Max.   :777
```

- We see that this dataset has three variables (or parameters) and few data points (18)
  - *size*: the independent variable
  - *effort*: the dependent variable
  - *EstTotal*: the estimates coming from an estimation method
- Basic Plots

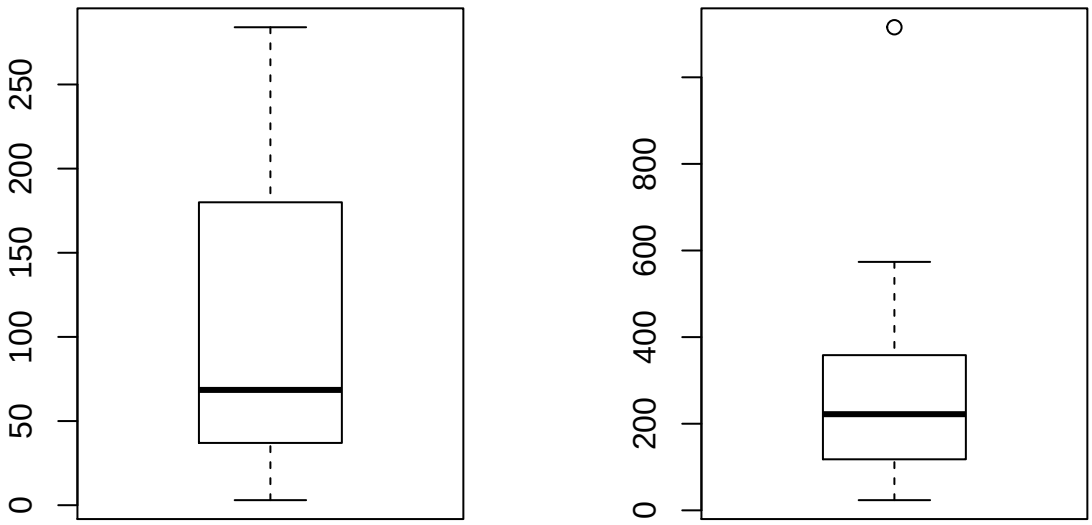
```
par(mfrow = c(1, 2))  #n figures per row
size_telecom1 <- telecom1$size
effort_telecom1 <- telecom1$effort

hist(size_telecom1, col = "blue", xlab = "size", ylab = "Probability", main = "Histogram of project Siz
lines(density(size_telecom1, na.rm = T, from = 0, to = max(size_telecom1)))
plot(density(size_telecom1))
```



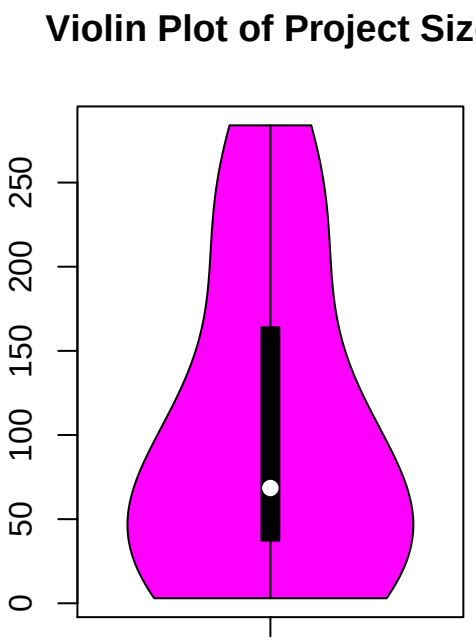

```
hist(effort_telecom1, col = "blue")
plot(density(effort_telecom1))
```

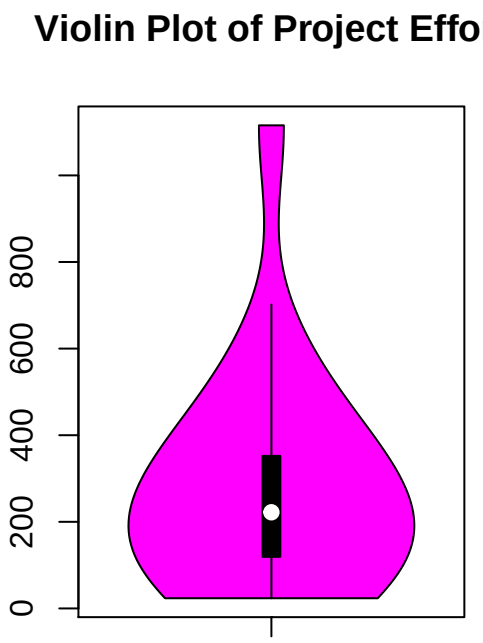



```r
boxplot(size_telecom1)
boxplot(effort_telecom1)
```

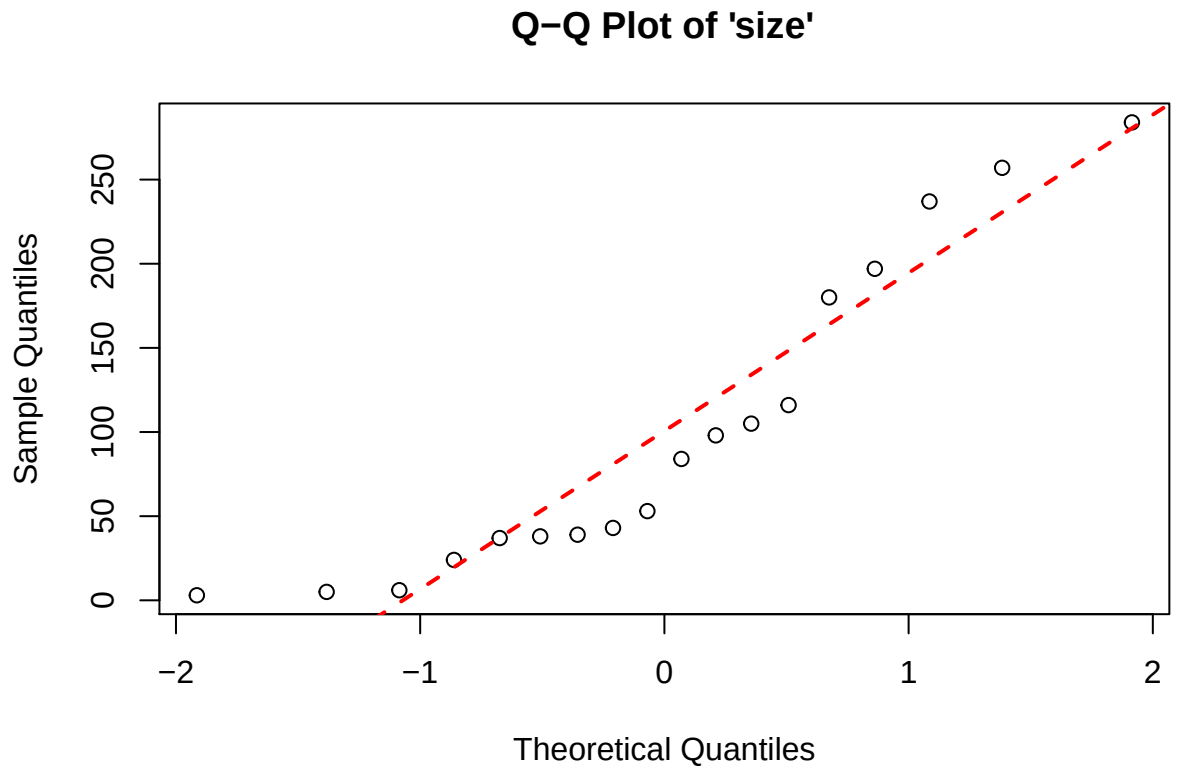



```r
# violin plots for those two variables
library(vioplot)
vioplot(size_telecom1, names = "")
title("Violin Plot of Project Size")
vioplot(effort_telecom1, names = "")
title("Violin Plot of Project Effort")
```






```
par(mfrow = c(1, 1))
qqnorm(size_telecom1, main = "Q-Q Plot of 'size'")
qqline(size_telecom1, col = 2, lwd = 2, lty = 2)  #draws a line through the first and third quartiles
```




```
qqnorm(effort_telecom1, main = "Q-Q Plot of 'effort'")
qqline(effort_telecom1)
```

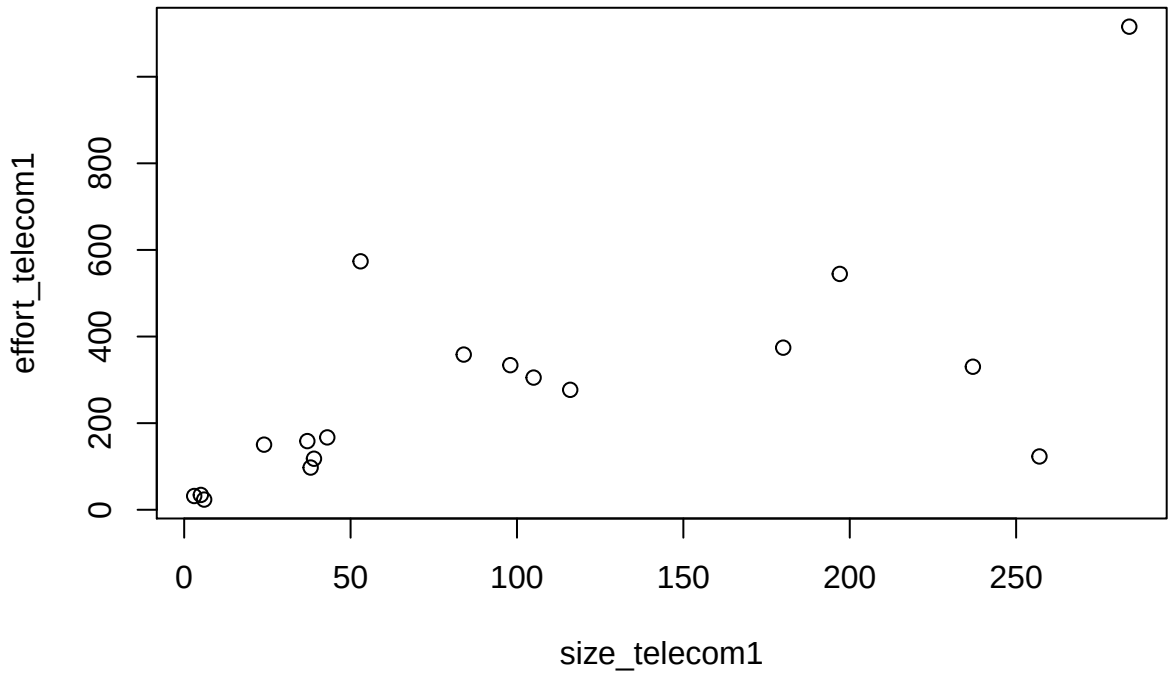
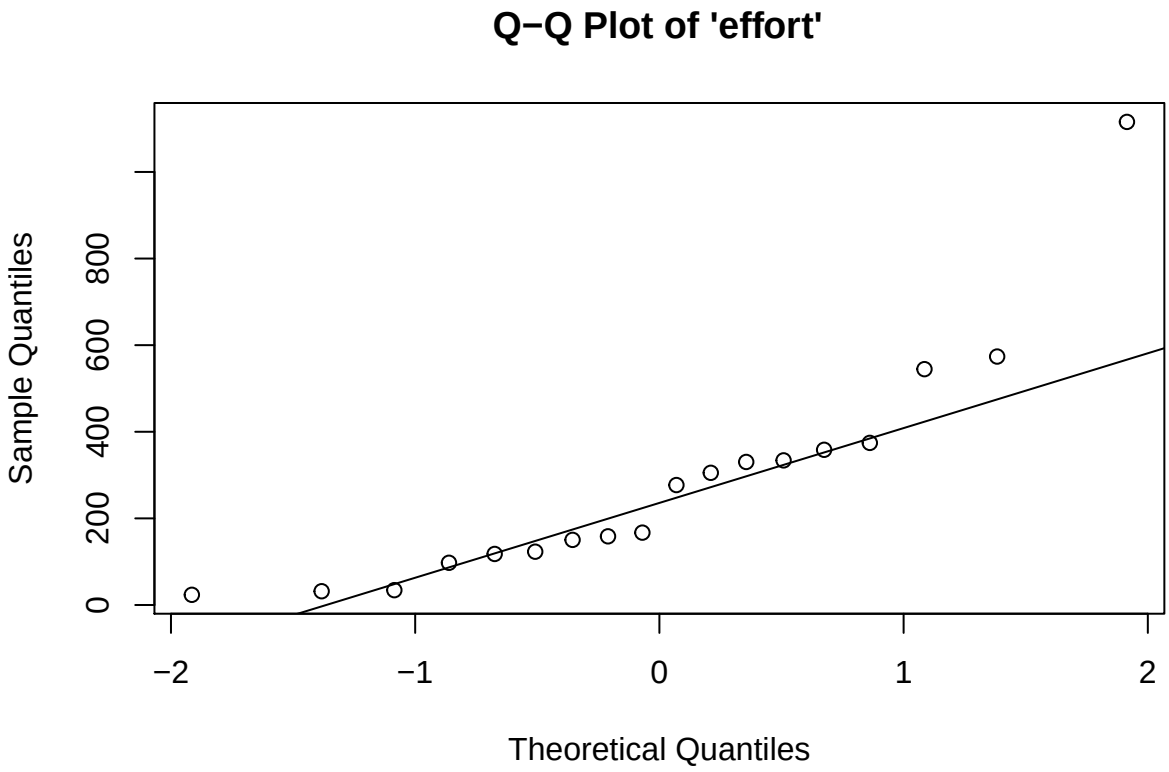


Figure 6.4: Scatterplot. Relationship between size and effort



- We can observe the non-normality of the data.
- We may look the posible relationship between size and effort with a scatterplot

```
plot(size_telecom1, effort_telecom1)
```

6.4.1 Example with the China dataset (from the tera-Promise Repository)

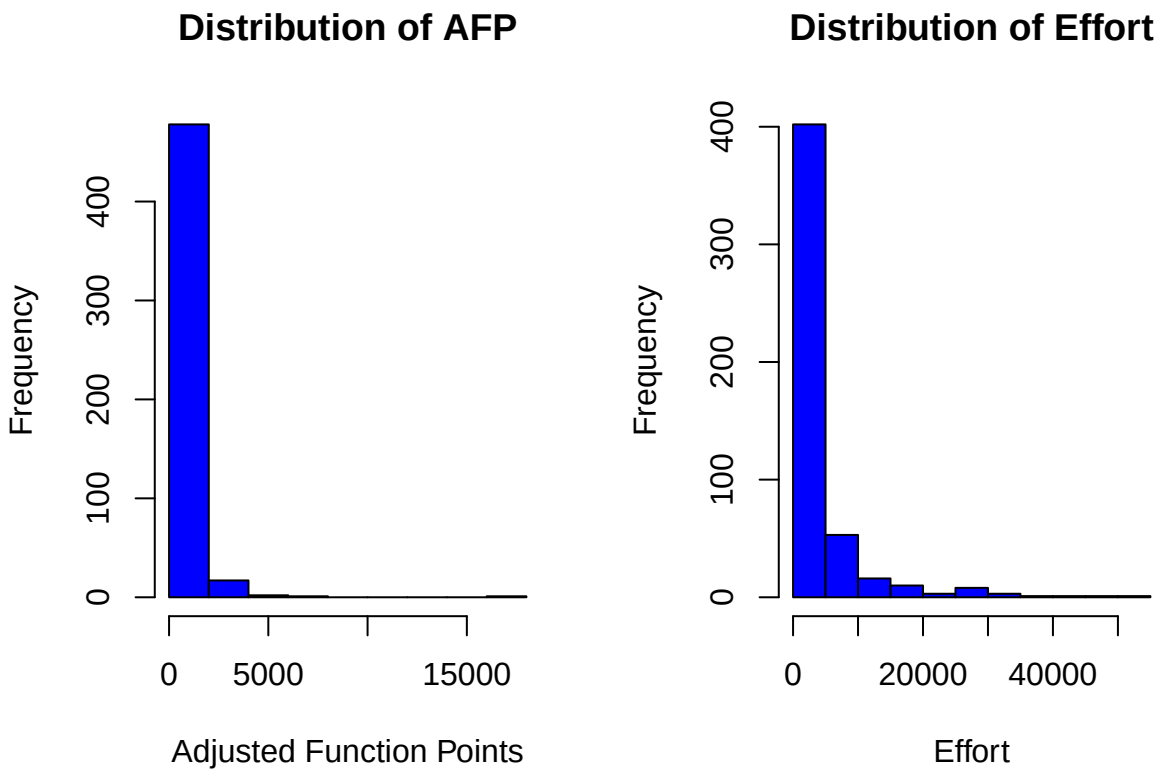
```
library(foreign)
china <- read.arff("./datasets/effortEstimation/china.arff")
china_size <- china$AFP
summary(china_size)
```

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

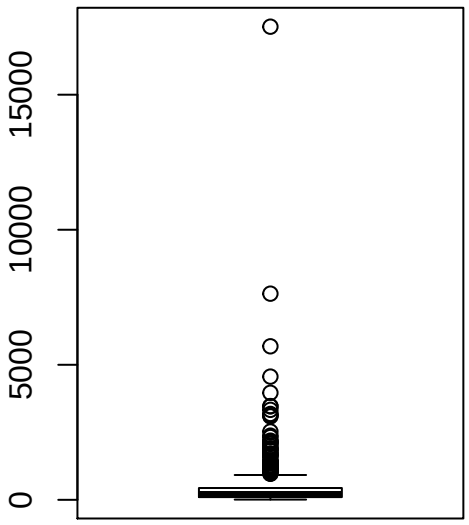
```
china_effort <- china$Effort
summary(china_effort)
```

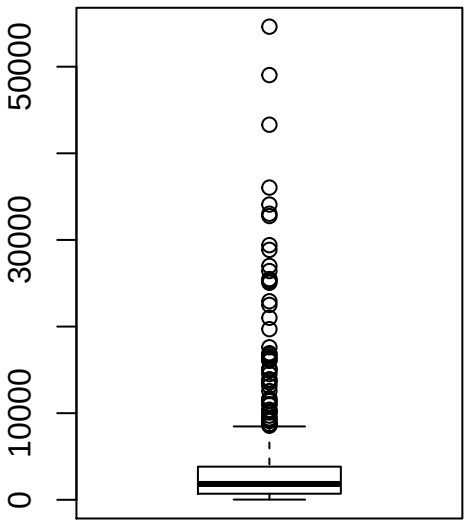
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      26     704    1829    3921    3826   54620
```

```
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")
```

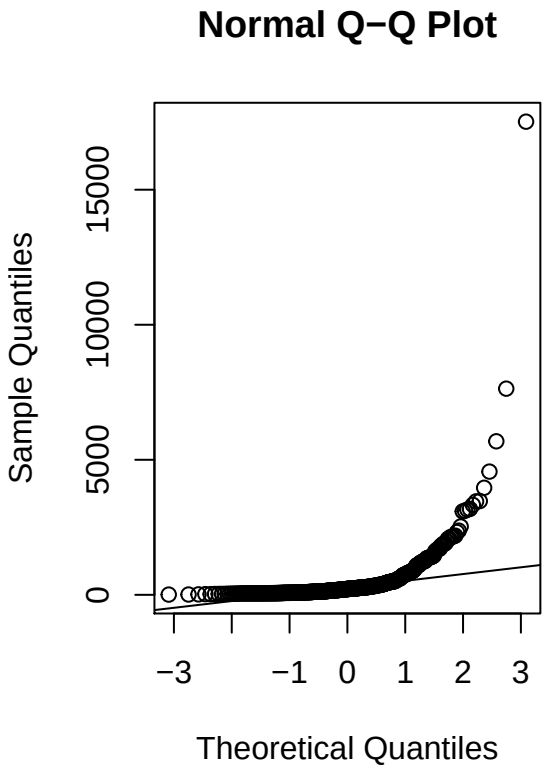


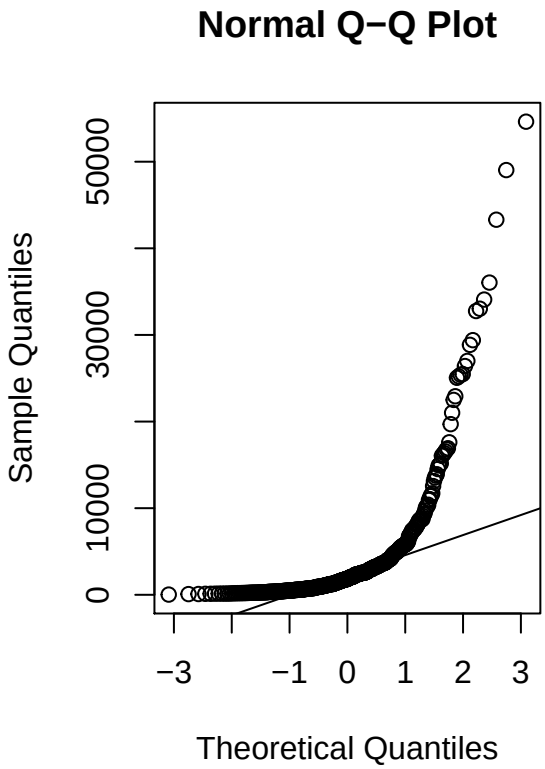

```
boxplot(china_size)
boxplot(china_effort)
```






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

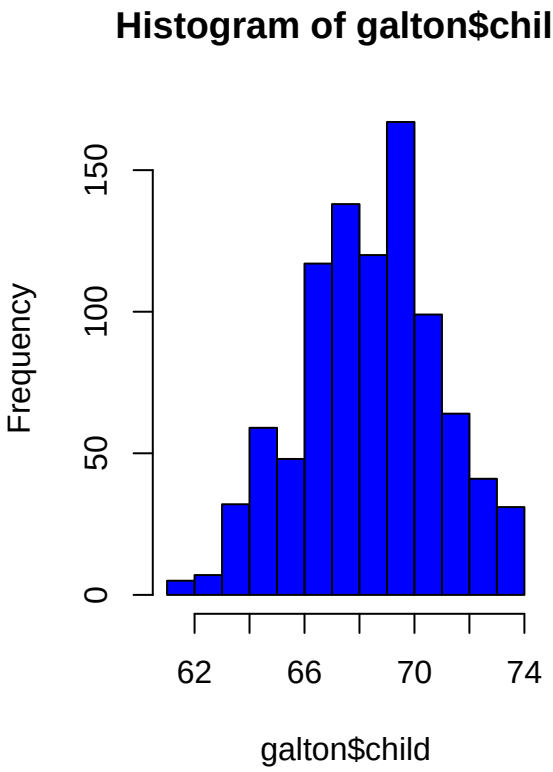


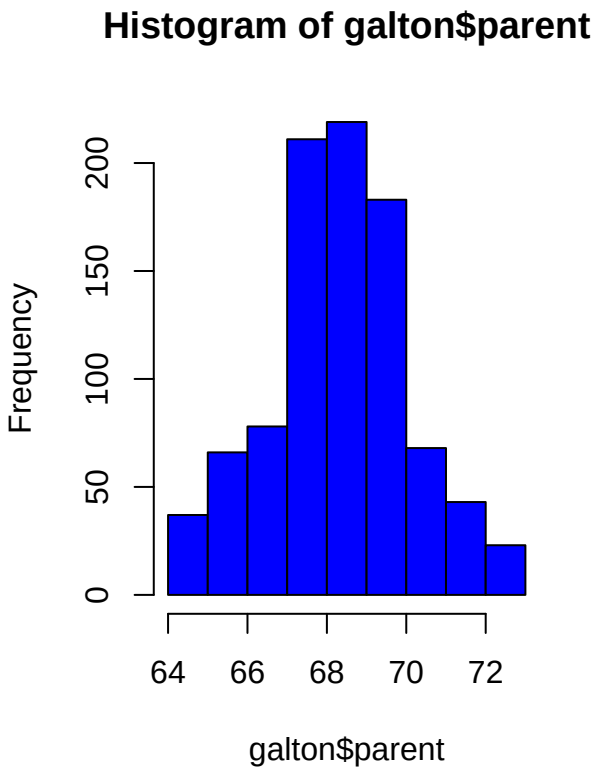


* We observe the non-normality of the data.

6.4.1.1 Normality. Galton data

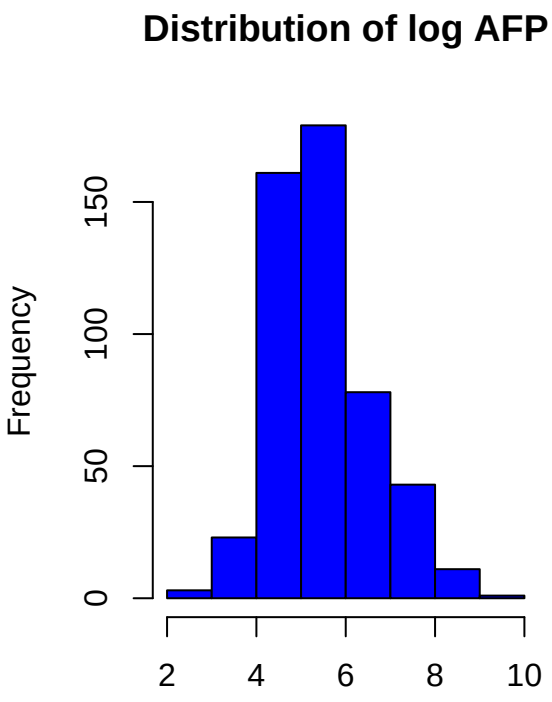
It is the data based on the famous 1885 Francis Galton's study about the relationship between the heights of adult children and the heights of their parents.

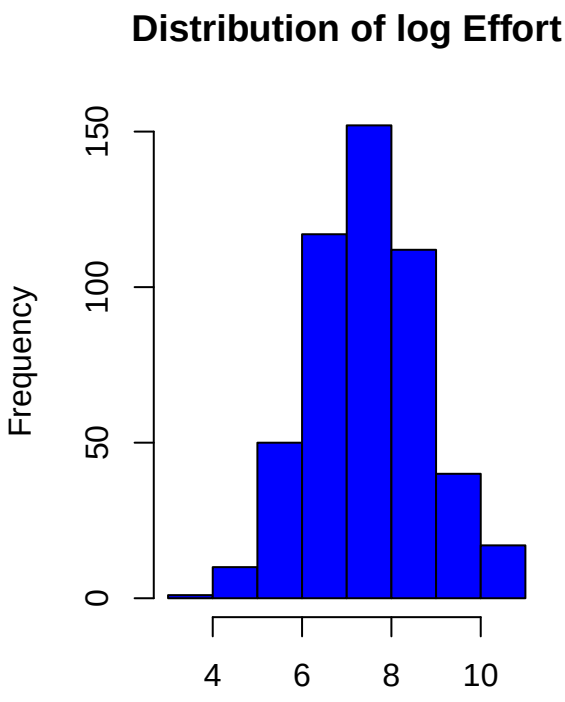


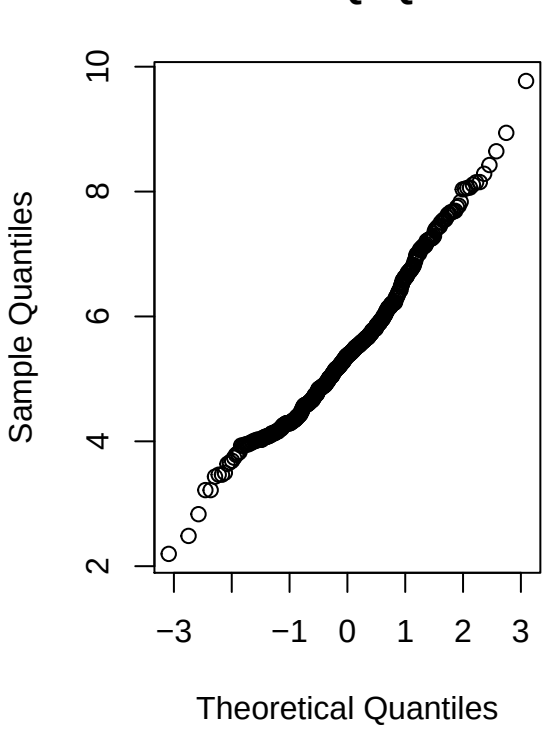


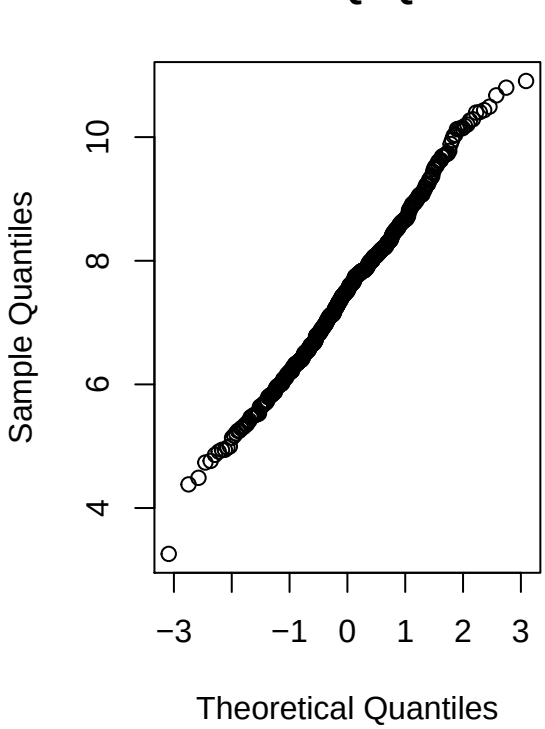
6.4.1.2 Normalization

Take *log*s in both independent variables. For example, with the *China* dataset.









- If the *log* transformation is used, then the estimation equation is:

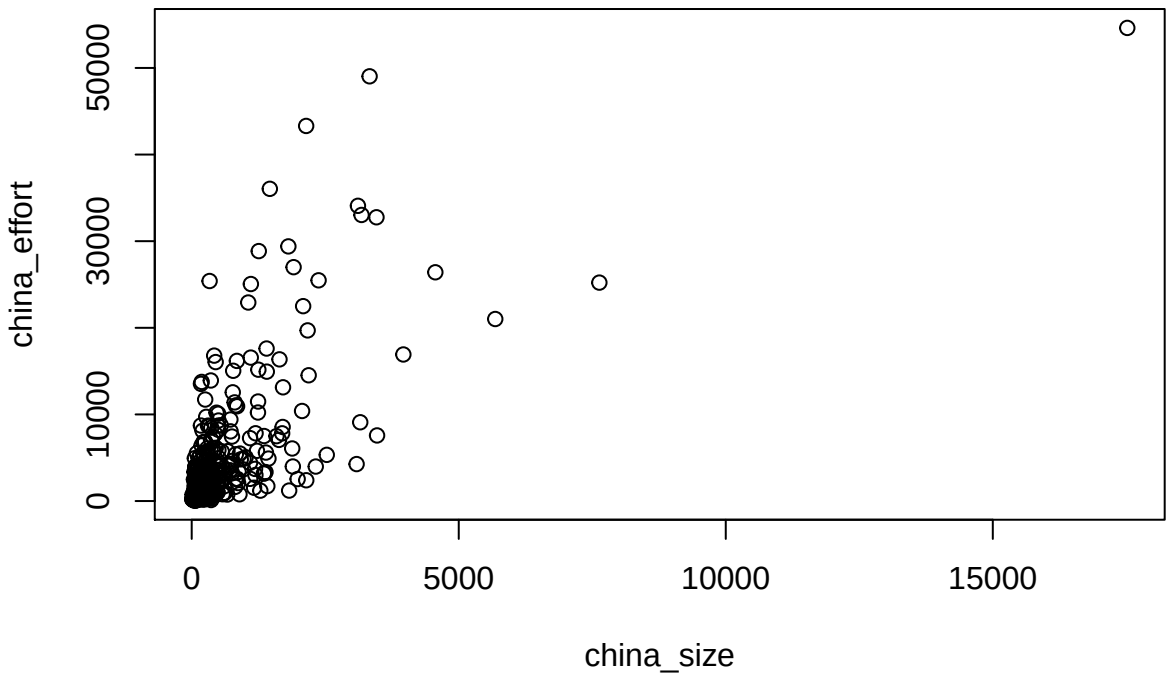
$$y = e^{b_0 + b_1 log(x)}$$

6.5 Correlation

Correlation is a statistical relationship between two sets of data. With the whole dataset we may check for the linear Correlation of the variables we are interested in.

As an example with the China dataset

```
par(mfrow=c(1,1))
plot(china_size,china_effort)
```




```
cor(china_size,china_effort)
```

```
## [1] 0.685
```

```
cor.test(china_size,china_effort)
```

```
##
##  Pearson's product-moment correlation
##
## data:  china_size and china_effort
## t = 20, df = 500, p-value <2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.635 0.729
## sample estimates:
##   cor
## 0.685
```

```
cor(china_size,china_effort, method="spearman")
```

```
## [1] 0.649
```

```
cor(china_size,china_effort, method="kendall")
```

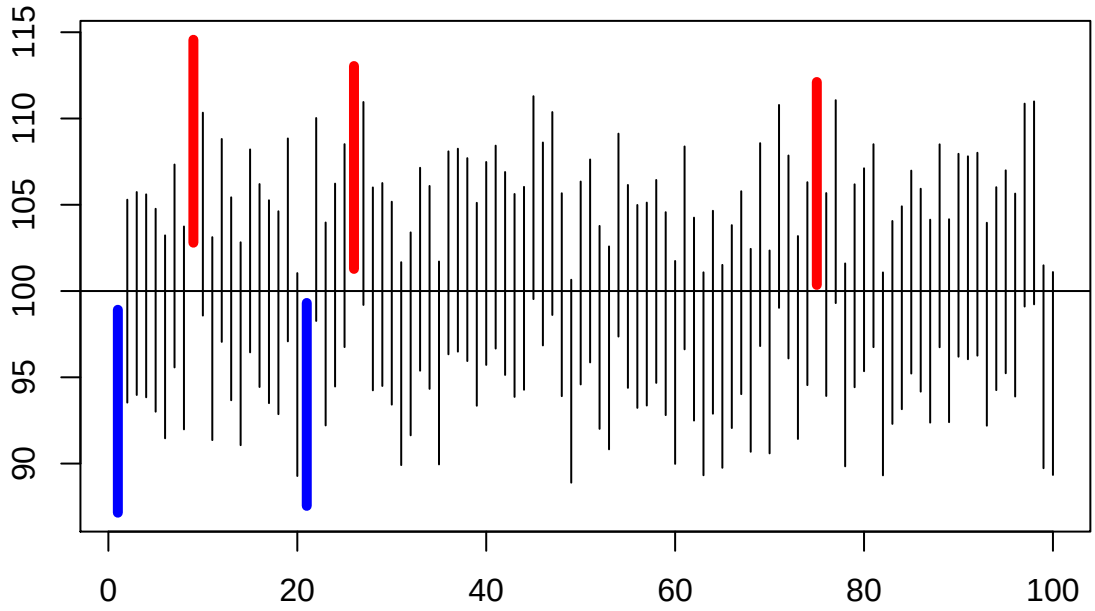
```
## [1] 0.468
```

6.6 Confidence Intervals. Bootstrap

- Until now we have generated point estimates
- A *confidence interval* (CI) is an interval estimate of a population parameter. The parameter can be the mean, the median or other. The frequentist CI is an observed interval that is different from sample to sample. It frequently includes the value of the unobservable parameter of interes if the experiment is repeated. The *confidence level* is the value that measures the frequency that the constructed intervals contain the true value of the parameter.
- The construction of a confidence interval with an exact value of confidence level for a distribution requires some statistical properties. Usually, *normality* is one of the properties required for computing confidence intervals.
  - Not all confidence intervals contain the true value of the parameter.
  - Simulation of confidence intervals

An example from Ugarte et al. (Ugarte et al., 2015)

```
set.seed(10)
norsim(sims = 100, n = 36, mu = 100, sigma = 18, conf.level = 0.95)
```

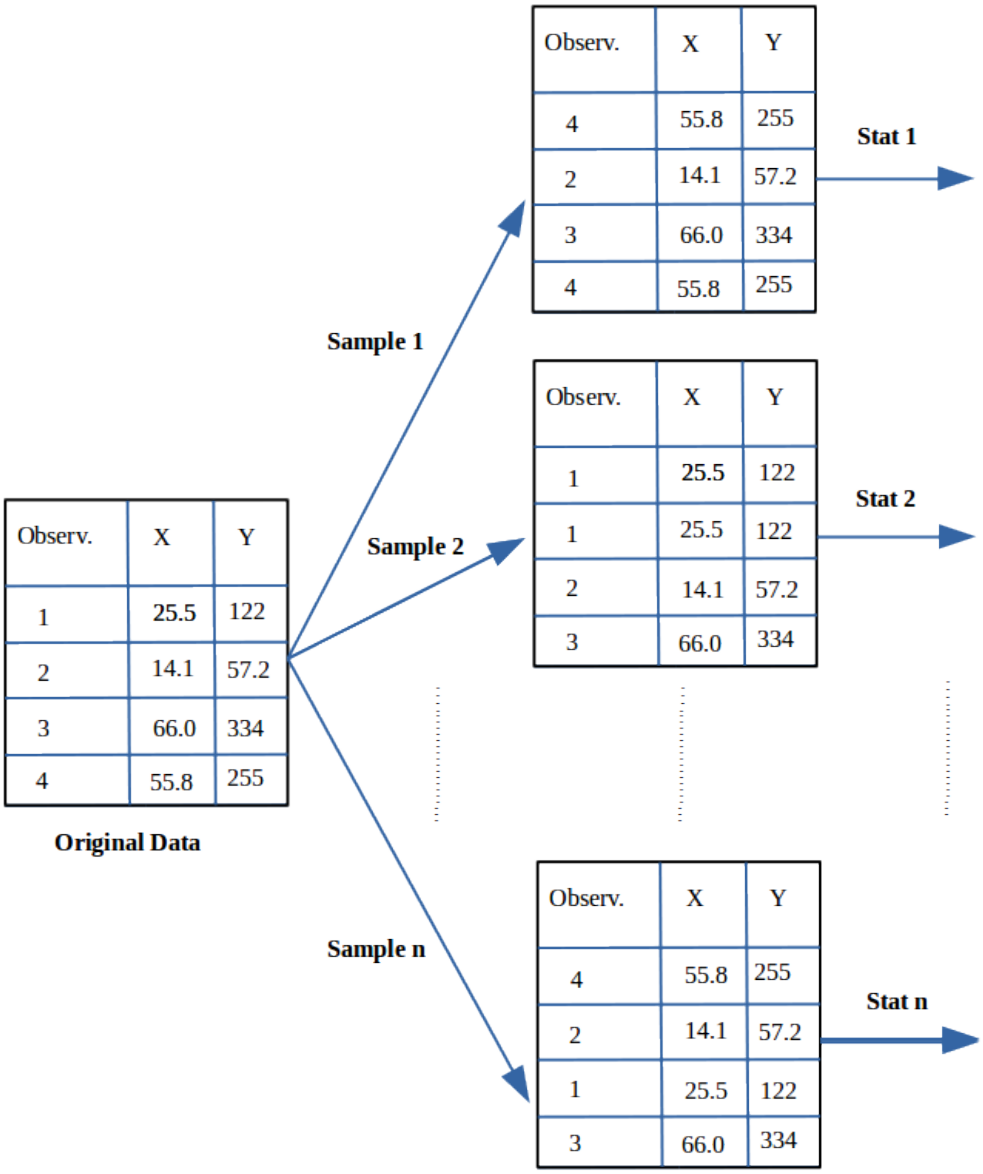


- The range defined by the confidence interval will vary with each sample, because the sample size will vary each time and the standard deviation will vary too.
- 95% confidence interval: it is the probability that the hypothetical confidence intervals (that would be computed from the hypothetical repeated samples) will contain the population mean.
- the particular interval that we compute on one sample does not mean that the population mean lies within that interval with a probability of 95%.
- Recommended reading: (Hoekstra et al., 2014) *Robust misinterpretation of confidence intervals*

6.7 Nonparametric Bootstrap

- For computing CIs the important thing is to know the assumptions that are made to "know" the distribution of the statistic.

- There is a way to compute confidence intervals without meeting the requirements of parametric methods.
- **Resampling** or **bootstraping** is a method to calculate estimates of a parameter taking samples from the original data and using those *resamples* to calculate statistics. Using the resamples usually gives more accurate results than using the original single sample to calculate an estimate of a parameter.


Observ. X Y
1 25.5 122
2 14.1 57.2
3 66.0 334
4 55.8 255
Original Data
Sample 1
Sample 2
Sample n
Observ. X Y
4 55.8 255
2 14.1 57.2
3 66.0 334
4 55.8 255
Stat 1
Observ. X Y
1 25.5 122
1 25.5 122
2 14.1 57.2
3 66.0 334
Stat 2
Observ. X Y
4 55.8 255
2 14.1 57.2
1 25.5 122
3 66.0 334
Stat n

An example of bootstrap CI can be found in Chapter 13, “Evaluation in Software Engineering”

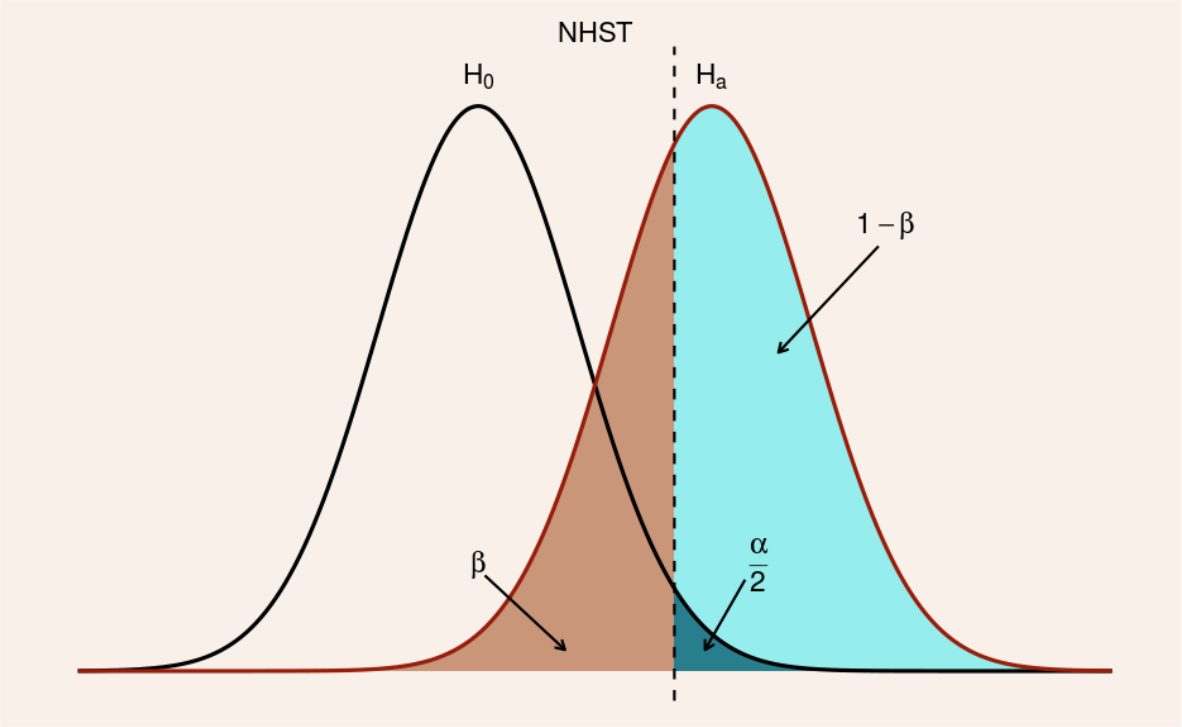
Chapter 7

Classical Hypothesis Testing

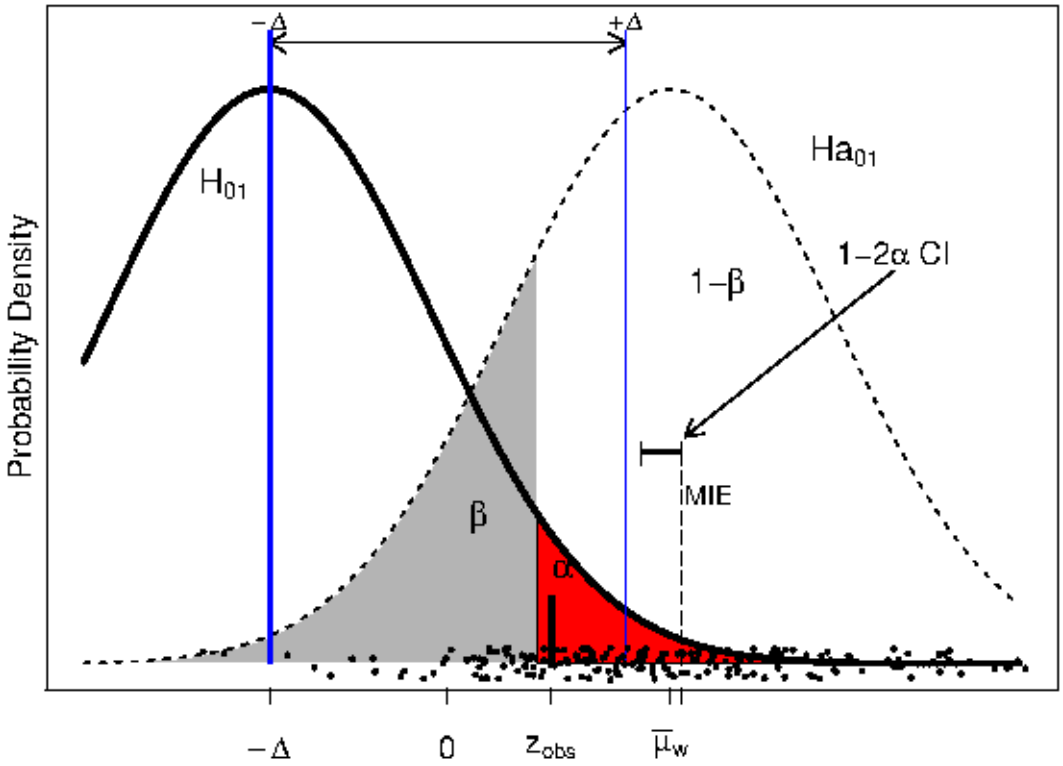
- By "classical" we mean the standard "frequentist" approach to hypothesis testing. The "frequentist" approach to probability sees it as the frequency of events in the long run. We repeat experiments over and over and we count the times that our object of interest appears in the sequence.
- The classical approach is usually called **null hypothesis significance testing** (NHST) because the process starts by setting a null hypothesis H_0 which is the opposite about what we think is true.
- The rationale of the process is that the statistical hypothesis should be *falsifiable*, that is, we can find evidence that the hypothesis is not true. We try to find evidence against the null hypothesis in order to support our alternative hypothesis H_A
- Usually, the null hypothesis is described as the situation of "no effect" and the alternative hypothesis describes the effect that we are looking for.
- After collecting data, taking an actual sample, we measure the distance of our parameter of interest from the hypothesized population parameter, and use the facts of the sampling distribution to determine the probability of obtaining such a sample *assuming the hypothesis is true*. This is amounts to a test of the hypothesis.
- If the probability of our sample, given the null hypothesis is high, this provides evidence that the null hypothesis is true. Conversely, if the probability of the sample is low (given the hypothesis), this is evidence against the null hypothesis. The hypothesis being tested in this way is named the *null hypothesis*.
- The goal of the test is to determine if the null hypothesis can be rejected. A statistical test can either reject or fail to reject a null hypothesis, but never prove it true.
- We can make two types of errors: false positive (Type I) and false negative (Type II)
- Type I and Type II errors

		Null hypothesis (H_0) is	
		Valid/True	**Invalid/False**
Judgement of Null Hypothesis (H_0)	**Reject**	Type I error False Positive	Correct inference True Positive
	Fail to reject	Correct inference True Negative	Type II error False negative

- Two-tailed NHST



- One-tailed NHST



- elementary example

```
data = c(52.7, 53.9, 41.7, 71.5, 47.6, 55.1, 62.2, 56.5, 33.4, 61.8, 54.3, 50.0, 45.3, 63.4, 53.9, 65.5
t.test(data, mu=50, alternative = 'greater')
```

```
##
##  One Sample t-test
##
## data:  data
## t = 2, df = 20, p-value = 0.02
## alternative hypothesis: true mean is greater than 50
## 95 percent confidence interval:
##  50.9  Inf
## sample estimates:
## mean of x
##      54.3
```

- Keeping this simple, we could start hypothesis testing about one sample median with the wilcoxon test for non-normal distributions.

  - "ae" is the absolute error in the China Test data

```
median(ae)
```

```
## [1] 867
```

```
mean(ae)
```

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

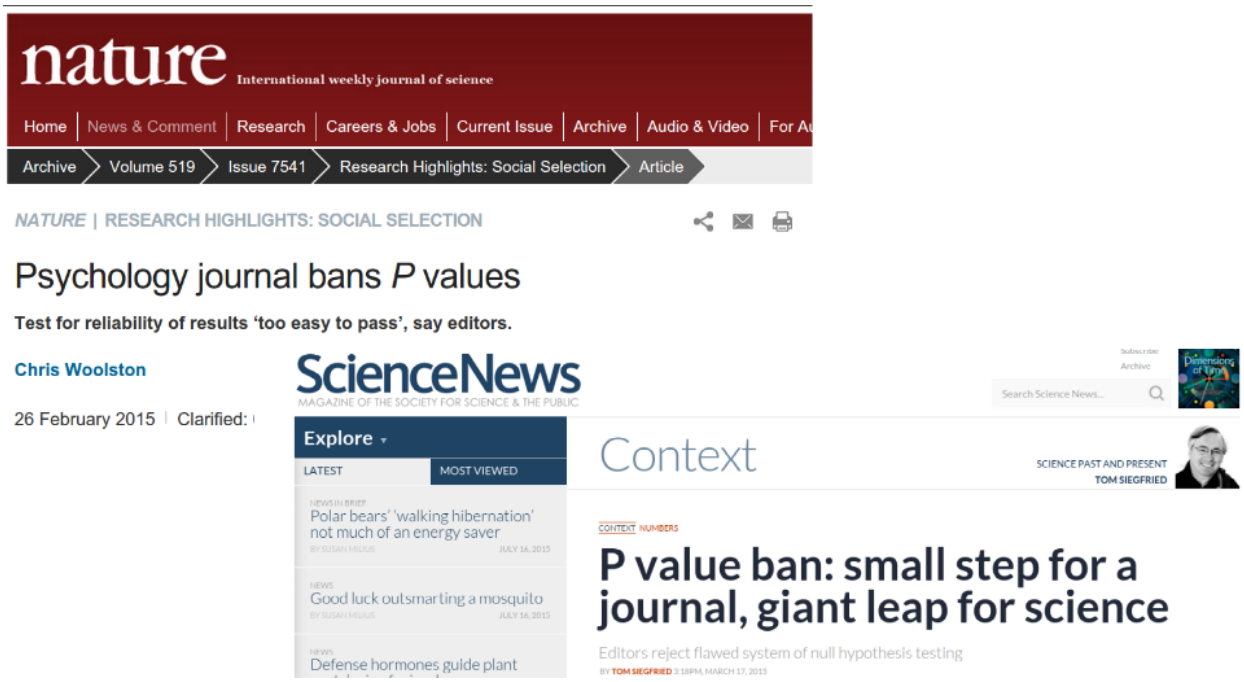
```
wilcox.test(ae, mu=800, alternative = 'greater') #change the values of mu and see the results
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  ae
## V = 9000, p-value = 8e-04
## alternative hypothesis: true location is greater than 800
```

- Quick introduction at https://psychstatsworkshop.wordpress.com/2014/08/06/lesson-9-hypothesis-testing/

7.1 p-values

- p-value: the p-value of a statistical test is the probability, computed assuming that H_0 is true, that the test statistic would take a value as extreme or more extreme than that actually observed.
- http://www.nature.com/news/psychology-journal-bans-p-values-1.17001
- https://www.sciencenews.org/blog/context/p-value-ban-small-step-journal-giant-leap-science



nature
International weekly journal of science
Home | News & Comment | Research | Careers & Jobs | Current Issue | Archive | Audio & Video | For Au
Archive > Volume 519 > Issue 7541 > Research Highlights: Social Selection > Article

NATURE | RESEARCH HIGHLIGHTS: SOCIAL SELECTION

Psychology journal bans *P* values

Test for reliability of results 'too easy to pass', say editors.

Chris Woolston

26 February 2015 | Clarified:

ScienceNews
MAGAZINE OF THE SOCIETY FOR SCIENCE & THE PUBLIC

Explore
LATEST | MOST VIEWED

NEWS IN BRIEF
Polar bears' 'walking hibernation' not much of an energy saver
JULY 16, 2015

NEWS
Good luck outsmarting a mosquito
JULY 16, 2015

NEWS
Defense hormones guide plant

Search Science News...

Context

SCIENCE PAST AND PRESENT
TOM SIEGFRIED

CONTEXT NUMBERS

P value ban: small step for a journal, giant leap for science

Editors reject flawed system of null hypothesis testing

BY TOM SIEGFRIED 3:18PM, MARCH 17, 2015

Part V

Preprocessing

Chapter 8

Preprocessing

Following the data mining process, we describe what is meant by preprocessing, classical supervised models, unsupervised models and evaluation in the context of software engineering with examples

This task is probably the hardest and where most of effort is spend in the data mining process. It is quite typical to transform the data, for example, finding inconsistencies, normalising, imputing missing values, tranforming input data, merging variables, etc.

Typically, preprocessing consist of the following tasks (subprocesses):

- Data cleaning (consistency, noise detection, outliers)
- Data integration
- Data transformation (normalisation, discretisation) and derivation of new attributes from existing ones (e.g., population density from population and area)
- Missing data imputation
- Data reduction (feature selection and instace selection)

8.1 Data

Consistent data are semantically correct based on real-world knowledge of the problem, i.e., no constrains are violated and data that can be used for inducing models and analysis. For example, the LoC or effort is constrained to non-negative values. We can also consider that to multiple attributes are consistent among them, and even datasets (e.g., same metrics but collected by different tools)

8.2 Missing values

Missing values will have a negative effect when analysing the data or learning models. The results can be biased when compared with the models induced from the complete data, the results can be harder to analyse, it may be needed to discard records with missing values depending on the algorithm and this can be an important problems with small datasets such as the effort estimation ones.

Missing data is typically classified into: * MCAR (Missing Completely at Random) or MAR (Missing At Random) where there is no reason for those missing values and we can assume that the distribution could follow the attribute's distribution. * MNAR (Missing Not At Random) where there is a pattern for those missing values and it may may be advisable to check the data gathering process to try to understand why such information is missing.

Imputation consists in replacing missing values for estimates of those missing values. Many algorithms do cannot handle missing values and therefore, imputation methods are needed. We can use simple approaches such as the replacing the missing values with the mean or mode of the attribute. More elaborated approaches include:

- EM (Expectation-Maximisation)
- Distance-based
  - kNN (k Nearest Neighbours)
  - Clustering

In R, a missing value is represented with `NA` and the analyst must decide what to do with missing data. The simplest approach is to leave out instances (ignore missing -IM-) with with missing data. This functionality is supported by many base functions through the `na.rm` option.

The `mice` R package. MICE (Multivariate Imputation via Chained Equations) assumes that data are missing at random. Other packages include `Amelia`, `missForest`, `Hmisc` and `mi`.

8.3 Noise

Imperfections of the real-world data that influences negatively in the induced machine learning models. Approaches to deal with noisy data include: * Robust learners capable of handling noisy data (e.g., C4.5 through pruning strategies) * Data polishing methods which aim to correct noisy instances prior training * Noise filters which are used to identify and eliminate noisy instances from the training data.

Types of noise data: * Class Noise (aka label noise). + There can be contradictory cases (all attributes have the same value except the class) + Misclassifications. The class attribute is not labeled with the true label (golden truth) * Attribute Noise. Values of attributes that are noise, missing or unknown.

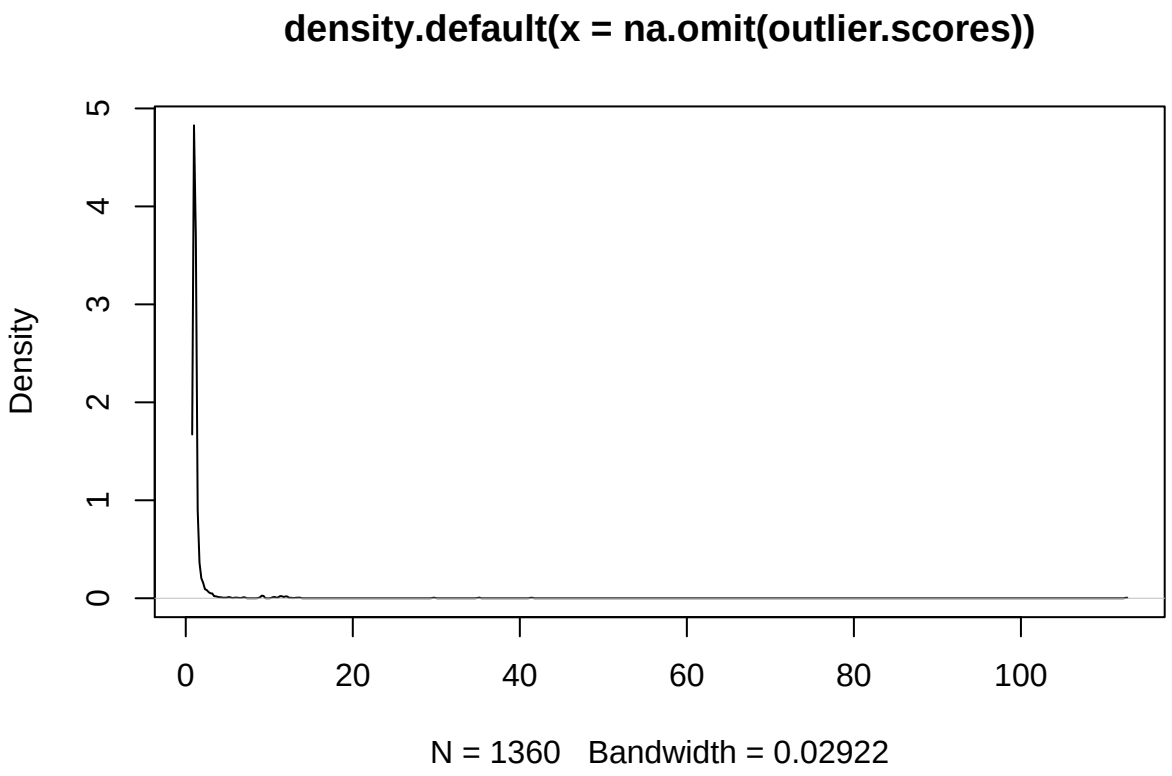
8.4 Outliers

There is a large amount of literature related to outlier detection, and furthermore several definitions of outlier exist.

```
library(DMwR)
library(foreign)

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

The LOF algorithm (`lofactor`), given a data set it produces a vector of local outlier factors for each case.

```
kc1num <- kc1[,1:21]
outlier.scores <- lofactor(kc1num, k=5)
plot(density(na.omit(outlier.scores)))
```




```
outliers <- order(outlier.scores, decreasing=T)[1:5]
print(outliers)
```

```
## [1]  1  6 14 31 33
```

Another simple method of Hiridoglou and Berthelot for positive observations.

8.5 Feature selection

Feature Selection (FS) aims at identifying the most relevant attributes from a dataset. It is important in different ways:

- 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 avoids the collection of data for those irrelevant and redundant attributes in the future.

The problem of FS received a thorough treatment in pattern recognition and machine learning. Most of the FS algorithms tackle the task as a *search* problem, where each state in the search specifies a distinct subset of the possible attributes (Blum and Langley, 1997). The search procedure is combined with a criterion to evaluate the merit of each candidate subset of attributes. There are a multiple possible combinations between each procedure search and each attribute measure (Liu and Yu, 2005).

There are two major approaches in FS from the method's output point of view:

- *Feature subset selection* (FSS)
- *Feature ranking* in which attributes are ranked as a list of features which are ordered according to evaluation measures (a subset of features is often selected from the top of the ranking list).

FFS 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 (Kohavi and John, 1997, Langley (1994)).

Feature subset algorithms search through candidate feature subsets guide by a certain evaluation measure (Liu and Motoda, 1998) which captures the goodness of each subset. An optimal (or near optimal) subset is selected when the search stops.

Some existing evaluation measures that have been shown effective in removing both irrelevant and redundant features include the consistency measure (Dash et al., 2000), the correlation measure (Hall, 1999) and the estimated accuracy of a learning algorithm (Kohavi and John, 1997).

- *Consistency* measure attempts to find a minimum number of features that separate classes as consistently as the full set of features can. An inconsistency is defined as to instances having the same feature values but different class labels.
- *Correlation* measure evaluates the goodness of feature subsets based on the hypothesis that good feature subsets contain features highly correlated to the class, yet uncorrelated to each other.
- *Wrapper-based* attribute selection uses the target learning algorithm to estimate the worth of attribute subsets. The feature subset selection algorithm conducts a search for a good subset using the induction algorithm itself as part of the evaluation function.

Langley (1994) notes that feature selection algorithms that search through the space of feature subsets must address four main issues: (i) the starting point of the search, (ii) the organization of the search, (iii) the evaluation of features subsets and (iv) the criterion used to terminate the search. Different algorithms address theses issues differently.

It is impractical to look at all possible feature subsets, even with a small number of attributes. Feature selection algorithms usually proceed greedily and are be classified into those that add features to an initially empty set (*forward selection*) and those that remove features from an initially complete set (*backwards elimination*). Hybrids both add and remove features as the algorithm progresses. Forward selection is much faster than backward elimination and therefore scales better to large data sets. A wide range of search strategies can be used: best-first, branch-and-bound, simulated annealing, genetic algorithms (see Kohavi and John (1997) for a review).

8.5.1 FSelector package in R

The FSelector package in R implements many algorithms available in Weka

```
library(FSelector)
library(foreign)

cm1 <- read.arff("./datasets/defectPred/D1/CM1.arff")

cm1RFWeigths <- random.forest.importance(Defective ~ ., cm1)
cutoff.biggest.diff(cm1RFWeigths)
```

```
## [1] "LOC_COMMENTS"         "NUM_UNIQUE_OPERATORS"
```

Using the Information Gain meaure as ranking:

```
cm1GRWeights <- gain.ratio(Defective ~ ., cm1)
cm1GRWeights
```

```
##                                 attr_importance
## LOC_BLANK                                0.0000
## BRANCH_COUNT                             0.0000
## CALL_PAIRS                               0.0000
## LOC_CODE_AND_COMMENT                     0.0000
## LOC_COMMENTS                             0.0754
## CONDITION_COUNT                          0.0000
## CYCLOMATIC_COMPLEXITY                    0.0000
## CYCLOMATIC_DENSITY                       0.0000
## DECISION_COUNT                           0.0000
## DECISION_DENSITY                         0.0000
## DESIGN_COMPLEXITY                        0.0000
## DESIGN_DENSITY                           0.0000
## EDGE_COUNT                               0.0000
## ESSENTIAL_COMPLEXITY                     0.0000
## ESSENTIAL_DENSITY                        0.0000
## LOC_EXECUTABLE                           0.0888
## PARAMETER_COUNT                          0.0000
## HALSTEAD_CONTENT                         0.0701
## HALSTEAD_DIFFICULTY                      0.0000
## HALSTEAD_EFFORT                          0.0375
## HALSTEAD_ERROR_EST                       0.0448
## HALSTEAD_LENGTH                          0.0425
## HALSTEAD_LEVEL                           0.0000
## HALSTEAD_PROG_TIME                       0.0375
## HALSTEAD_VOLUME                          0.0471
## MAINTENANCE_SEVERITY                     0.0000
## MODIFIED_CONDITION_COUNT                 0.0000
## MULTIPLE_CONDITION_COUNT                 0.0000
## NODE_COUNT                               0.0000
## NORMALIZED_CYLOMATIC_COMPLEXITY          0.0000
## NUM_OPERANDS                             0.0000
## NUM_OPERATORS                            0.0471
## NUM_UNIQUE_OPERANDS                      0.0589
## NUM_UNIQUE_OPERATORS                     0.0616
## NUMBER_OF_LINES                          0.0573
## PERCENT_COMMENTS                         0.0663
## LOC_TOTAL                                0.0763
```

```
cutoff.biggest.diff(cm1GRWeights)
```

```
##  [1] "LOC_EXECUTABLE"       "LOC_TOTAL"            "LOC_COMMENTS"
##  [4] "HALSTEAD_CONTENT"     "PERCENT_COMMENTS"     "NUM_UNIQUE_OPERATORS"
##  [7] "NUM_UNIQUE_OPERANDS"  "NUMBER_OF_LINES"      "HALSTEAD_VOLUME"
## [10] "NUM_OPERATORS"        "HALSTEAD_ERROR_EST"   "HALSTEAD_LENGTH"
## [13] "HALSTEAD_EFFORT"      "HALSTEAD_PROG_TIME"
```

```
# After assigning weights, we can select the statistaclly significant ones
cm1X2Weights <- chi.squared(Defective ~ ., cm1)
cutoff.biggest.diff(cm1X2Weights)
```

```
##  [1] "LOC_EXECUTABLE"       "LOC_COMMENTS"         "LOC_TOTAL"
##  [4] "NUM_UNIQUE_OPERATORS" "NUM_UNIQUE_OPERANDS"  "NUMBER_OF_LINES"
##  [7] "HALSTEAD_VOLUME"      "NUM_OPERATORS"        "HALSTEAD_ERROR_EST"
## [10] "HALSTEAD_CONTENT"     "HALSTEAD_EFFORT"      "HALSTEAD_PROG_TIME"
```

```
## [13] "HALSTEAD_LENGTH"      "PERCENT_COMMENTS"
```

Using CFS attribute selection

```
library(FSelector)
library(foreign)

cm1 <- read.arff("./datasets/defectPred/D1/CM1.arff")

result <- cfs(Defective ~ ., cm1)
f <- as.simple.formula(result, "Defective")
f
```

```
## Defective ~ LOC_COMMENTS + LOC_EXECUTABLE + HALSTEAD_CONTENT +
##     NUM_UNIQUE_OPERATORS + PERCENT_COMMENTS
## <environment: 0x4ec77c8>
```

Other packages for Feature selection in R include `FSelectorRccp` which re-implments the FSlector without WEKA dependencies.

Another popular package is `Boruta`, which is based on selection based on Random Forest.

8.6 Instance selection

Removal of samples (complementary to the removal of attributes) in order to scale down the dataset prior to learning a model so that there is (almost) no performance loss.

There are two types of processes:

- *Prototype Selection* (PS) (Garcia et al., 2012) when the subset is used with a distance based method (kNN)
- *Training Set Selection* (TSS) (Cano et al., 2007) in which an actual model is learned.

It is also a search problem as with *feature selection*. Garcia et al. (2012) provide a comprehensive overview of the topic.

8.7 Discretization

This process transforms continuous attributes into discrete ones, by associating categorical values to intervals and thus transforming quantitative data into qualitative data.

8.8 Correlation Coefficient and Covariance for Numeric Data

Two random variables x and y are called independent if the probability distribution of one variable is not affected by the presence of another.

$\tilde{\chi}^2 = \frac{1}{d}\sum_{k=1}^{n}\frac{(O_k - E_k)^2}{E_k}$

```
chisq.test(kc1$LOC_BLANK,kc1$BRANCH_TOTAL)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  kc1$LOC_BLANK
```

```
## X-squared = 20000, df = 2000, p-value <2e-16
```

```
chisq.test(kc1$DESIGN_COMPLEXITY,kc1$CYCLOMATIC_COMPLEXITY)
```

```
##
##  Pearson's Chi-squared test
##
## data:  kc1$DESIGN_COMPLEXITY and kc1$CYCLOMATIC_COMPLEXITY
## X-squared = 30000, df = 700, p-value <2e-16
```

8.9 Normalization

8.9.1 Min-Max Normalization

$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$

```
library(caret)
preObj <- preProcess(kc1[, -22], method=c("center", "scale"))
```

8.9.2 Z-score normalization

TBD

8.10 Transformations

8.10.1 Linear Transformations and Quadratic Trans formations

TBD

8.10.2 Box-cox transformation

TBD

8.10.3 Nominal to Binary tranformations

TBD

8.11 Preprocessing in R

8.11.1 The `dplyr` package

The *dplyr* package created by Hadley Wickham. Some functions are similar to SQL syntax and it key functions in dplyr include:

- select: select columns from a dataframe
- filter: select rows from a dataframe
- summarize: allows us to do summary stats based upon the grouped variable

- group_by: group by a factor variable
- arrange: order the dataset
- joins: as in sql left join

Tutorial: https://github.com/justmarkham/dplyr-tutorial

Examples

```
library(dplyr)
```

Describe the dataframe:

```
str(kc1)
```

```
## 'data.frame':    2096 obs. of  22 variables:
##  $ LOC_BLANK             : num  0 0 0 0 2 0 0 0 0 2 ...
##  $ BRANCH_COUNT          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ LOC_CODE_AND_COMMENT : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ LOC_COMMENTS          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CYCLOMATIC_COMPLEXITY: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ DESIGN_COMPLEXITY     : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ ESSENTIAL_COMPLEXITY : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ LOC_EXECUTABLE        : num  3 1 1 1 8 3 1 1 1 9 ...
##  $ HALSTEAD_CONTENT      : num  11.6 0 0 0 18 ...
##  $ HALSTEAD_DIFFICULTY   : num  2.67 0 0 0 3.5 2.67 0 0 0 3.75 ...
##  $ HALSTEAD_EFFORT       : num  82.3 0 0 0 220.9 ...
##  $ HALSTEAD_ERROR_EST    : num  0.01 0 0 0 0.02 0.01 0 0 0 0.04 ...
##  $ HALSTEAD_LENGTH       : num  11 1 1 1 19 11 1 1 1 29 ...
##  $ HALSTEAD_LEVEL        : num  0.38 0 0 0 0.29 0.38 0 0 0 0.27 ...
##  $ HALSTEAD_PROG_TIME    : num  4.57 0 0 0 12.27 ...
##  $ HALSTEAD_VOLUME       : num  30.9 0 0 0 63.1 ...
##  $ NUM_OPERANDS          : num  4 0 0 0 7 4 0 0 0 10 ...
##  $ NUM_OPERATORS         : num  7 1 1 1 12 7 1 1 1 19 ...
##  $ NUM_UNIQUE_OPERANDS   : num  3 0 0 0 5 3 0 0 0 8 ...
##  $ NUM_UNIQUE_OPERATORS : num  4 1 1 1 5 4 1 1 1 6 ...
##  $ LOC_TOTAL             : num  5 3 3 3 12 5 3 3 3 13 ...
##  $ Defective             : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
```

`tbl_df` creates a "local data frame" as a wrapper for better printing

```
kc1_tbl <- tbl_df(kc1)
```

Filter:

```
# Filter rows: use comma or & to represent AND condition
filter(kc1_tbl, Defective == "Y" & LOC_BLANK != 0)
```

```
## # A tibble: 251 x 22
##    LOC_BLANK BRANCH_COUNT LOC_CODE_AND_CO~ LOC_COMMENTS CYCLOMATIC_COMP~
##        <dbl>        <dbl>            <dbl>        <dbl>            <dbl>
##  1         6           21                0           10               11
##  2         5           15                0            2                8
##  3         2            5                0            0                3
##  4         4            5                0            2                3
##  5         2           11                0            2                6
##  6         2           23                0            3               12
##  7         1           11                0            2                6
##  8         1           13                0            2                7
##  9         2           17                0            2                9
```

```
## 10         3            1                 0            0                 1
## # ... with 241 more rows, and 17 more variables: DESIGN_COMPLEXITY <dbl>,
## #   ESSENTIAL_COMPLEXITY <dbl>, LOC_EXECUTABLE <dbl>,
## #   HALSTEAD_CONTENT <dbl>, HALSTEAD_DIFFICULTY <dbl>,
## #   HALSTEAD_EFFORT <dbl>, HALSTEAD_ERROR_EST <dbl>,
## #   HALSTEAD_LENGTH <dbl>, HALSTEAD_LEVEL <dbl>, HALSTEAD_PROG_TIME <dbl>,
## #   HALSTEAD_VOLUME <dbl>, NUM_OPERANDS <dbl>, NUM_OPERATORS <dbl>,
## #   NUM_UNIQUE_OPERANDS <dbl>, NUM_UNIQUE_OPERATORS <dbl>,
## #   LOC_TOTAL <dbl>, Defective <fct>
```

Another operator is `%in%`.

Select:

```
select(kc1_tbl, contains("LOC"), Defective)
```

```
## # A tibble: 2,096 x 6
##    LOC_BLANK LOC_CODE_AND_CO~ LOC_COMMENTS LOC_EXECUTABLE LOC_TOTAL
##        <dbl>            <dbl>        <dbl>          <dbl>     <dbl>
##  1         0                0            0              3         5
##  2         0                0            0              1         3
##  3         0                0            0              1         3
##  4         0                0            0              1         3
##  5         2                0            0              8        12
##  6         0                0            0              3         5
##  7         0                0            0              1         3
##  8         0                0            0              1         3
##  9         0                0            0              1         3
## 10         2                0            0              9        13
## # ... with 2,086 more rows, and 1 more variable: Defective <fct>
```

Now, `kc1_tbl` contains("LOC"), Defective

Filter and Select together:

```
# nesting method
filter(select(kc1_tbl, contains("LOC"), Defective), Defective !=0)
```

```
## # A tibble: 2,096 x 6
##    LOC_BLANK LOC_CODE_AND_CO~ LOC_COMMENTS LOC_EXECUTABLE LOC_TOTAL
##        <dbl>            <dbl>        <dbl>          <dbl>     <dbl>
##  1         0                0            0              3         5
##  2         0                0            0              1         3
##  3         0                0            0              1         3
##  4         0                0            0              1         3
##  5         2                0            0              8        12
##  6         0                0            0              3         5
##  7         0                0            0              1         3
##  8         0                0            0              1         3
##  9         0                0            0              1         3
## 10         2                0            0              9        13
## # ... with 2,086 more rows, and 1 more variable: Defective <fct>
```

It is easier usign the chaining method:

```
# chaining method
kc1_tbl %>%
    select(contains("LOC"), Defective) %>%
```

```
    filter(Defective !=0)
```

```
## # A tibble: 2,096 x 6
##    LOC_BLANK LOC_CODE_AND_CO~ LOC_COMMENTS LOC_EXECUTABLE LOC_TOTAL
##        <dbl>            <dbl>        <dbl>           <dbl>     <dbl>
##  1         0                0            0               3         5
##  2         0                0            0               1         3
##  3         0                0            0               1         3
##  4         0                0            0               1         3
##  5         2                0            0               8        12
##  6         0                0            0               3         5
##  7         0                0            0               1         3
##  8         0                0            0               1         3
##  9         0                0            0               1         3
## 10         2                0            0               9        13
## # ... with 2,086 more rows, and 1 more variable: Defective <fct>
```

Arrange ascending

```
#
kc1_tbl %>%
    select(LOC_TOTAL, Defective) %>%
    arrange(LOC_TOTAL)
```

```
## # A tibble: 2,096 x 2
##    LOC_TOTAL Defective
##        <dbl> <fct>
##  1         1 N
##  2         1 N
##  3         1 N
##  4         1 N
##  5         1 N
##  6         1 N
##  7         1 N
##  8         1 N
##  9         1 N
## 10         1 N
## # ... with 2,086 more rows
```

Arrange descending:

```
kc1_tbl %>%
    select(LOC_TOTAL, Defective) %>%
    arrange(desc(LOC_TOTAL))
```

```
## # A tibble: 2,096 x 2
##    LOC_TOTAL Defective
##        <dbl> <fct>
##  1       288 Y
##  2       286 Y
##  3       283 N
##  4       220 Y
##  5       217 Y
##  6       210 N
##  7       205 Y
##  8       184 Y
```

```
##  9       179 Y
## 10       176 Y
## # ... with 2,086 more rows
```

Mutate:

```
kc1_tbl %>%
    filter(Defective == "Y") %>%
    select(NUM_OPERANDS, NUM_OPERATORS, Defective) %>%
    mutate(HalsteadLength = NUM_OPERANDS + NUM_OPERATORS)
```

```
## # A tibble: 325 x 4
##    NUM_OPERANDS NUM_OPERATORS Defective HalsteadLength
##           <dbl>         <dbl> <fct>              <dbl>
##  1           64           107 Y                    171
##  2           52            89 Y                    141
##  3           17            41 Y                     58
##  4           41            74 Y                    115
##  5           54            95 Y                    149
##  6           75           156 Y                    231
##  7           54            95 Y                    149
##  8           56            99 Y                    155
##  9           69           124 Y                    193
## 10           44            60 Y                    104
## # ... with 315 more rows
```

summarise: Reduce variables to values

```
# Create a table grouped by Defective, and then summarise each group by taking the mean of loc
kc1_tbl %>%
    group_by(Defective) %>%
    summarise(avg_loc = mean(LOC_TOTAL, na.rm=TRUE))
```

```
## # A tibble: 2 x 2
##   Defective avg_loc
##   <fct>       <dbl>
## 1 N            15.9
## 2 Y            44.7
```

```
# Create a table grouped by Defective, and then summarise each group by taking the mean of loc
kc1_tbl %>%
    group_by(Defective) %>%
    summarise_each(funs(mean, min, max), BRANCH_COUNT, LOC_TOTAL)
```

```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over a selection of variables, use `summarise_at()`
```

```
## # A tibble: 2 x 7
##   Defective BRANCH_COUNT_me~ LOC_TOTAL_mean BRANCH_COUNT_min LOC_TOTAL_min
##   <fct>                <dbl>          <dbl>            <dbl>         <dbl>
## 1 N                     3.68           15.9                1             1
## 2 Y                    10.1            44.7                1             2
## # ... with 2 more variables: BRANCH_COUNT_max <dbl>, LOC_TOTAL_max <dbl>
```

It seems than the number of *Defective* modules is larger than the *Non-Defective* ones. We can count them with:

```
# n() or tally
kc1_tbl %>%
    group_by(Defective) %>%
    tally()
```

```
## # A tibble: 2 x 2
##   Defective     n
##   <fct>     <int>
## 1 N          1771
## 2 Y           325
```

It seems that it's an imbalanced dataset...

```
# randomly sample a fixed number of rows, without replacement
kc1_tbl %>% sample_n(2)
```

```
## # A tibble: 2 x 22
##   LOC_BLANK BRANCH_COUNT LOC_CODE_AND_CO~ LOC_COMMENTS CYCLOMATIC_COMP~
##       <dbl>        <dbl>            <dbl>        <dbl>            <dbl>
## 1         0            1                0            0                1
## 2         2            7                0            1                4
## # ... with 17 more variables: DESIGN_COMPLEXITY <dbl>,
## #   ESSENTIAL_COMPLEXITY <dbl>, LOC_EXECUTABLE <dbl>,
## #   HALSTEAD_CONTENT <dbl>, HALSTEAD_DIFFICULTY <dbl>,
## #   HALSTEAD_EFFORT <dbl>, HALSTEAD_ERROR_EST <dbl>,
## #   HALSTEAD_LENGTH <dbl>, HALSTEAD_LEVEL <dbl>, HALSTEAD_PROG_TIME <dbl>,
## #   HALSTEAD_VOLUME <dbl>, NUM_OPERANDS <dbl>, NUM_OPERATORS <dbl>,
## #   NUM_UNIQUE_OPERANDS <dbl>, NUM_UNIQUE_OPERATORS <dbl>,
## #   LOC_TOTAL <dbl>, Defective <fct>
```

```
# randomly sample a fraction of rows, with replacement
kc1_tbl %>% sample_frac(0.05, replace=TRUE)
```

```
## # A tibble: 105 x 22
##    LOC_BLANK BRANCH_COUNT LOC_CODE_AND_CO~ LOC_COMMENTS CYCLOMATIC_COMP~
##        <dbl>        <dbl>            <dbl>        <dbl>            <dbl>
##  1         1            1                0            0                1
##  2         0            1                0            0                1
##  3         5            7                0            0                4
##  4         0            1                0            0                1
##  5         0            1                0            0                1
##  6         2            3                0            0                2
##  7         1            3                1            1                2
##  8         1            3                0            0                2
##  9         0            3                0            0                2
## 10         2            5                0            0                3
## # ... with 95 more rows, and 17 more variables: DESIGN_COMPLEXITY <dbl>,
## #   ESSENTIAL_COMPLEXITY <dbl>, LOC_EXECUTABLE <dbl>,
## #   HALSTEAD_CONTENT <dbl>, HALSTEAD_DIFFICULTY <dbl>,
## #   HALSTEAD_EFFORT <dbl>, HALSTEAD_ERROR_EST <dbl>,
## #   HALSTEAD_LENGTH <dbl>, HALSTEAD_LEVEL <dbl>, HALSTEAD_PROG_TIME <dbl>,
## #   HALSTEAD_VOLUME <dbl>, NUM_OPERANDS <dbl>, NUM_OPERATORS <dbl>,
## #   NUM_UNIQUE_OPERANDS <dbl>, NUM_UNIQUE_OPERATORS <dbl>,
## #   LOC_TOTAL <dbl>, Defective <fct>
```

```
# Better formatting adapted to the screen width
glimpse(kc1_tbl)
```

```
## Observations: 2,096
## Variables: 22
## $ LOC_BLANK             <dbl> 0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 2, 0, 2, 1...
## $ BRANCH_COUNT          <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ LOC_CODE_AND_COMMENT  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ LOC_COMMENTS          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ CYCLOMATIC_COMPLEXITY <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ DESIGN_COMPLEXITY     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ ESSENTIAL_COMPLEXITY  <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ LOC_EXECUTABLE        <dbl> 3, 1, 1, 1, 8, 3, 1, 1, 1, 9, 8, 1, 8, 1...
## $ HALSTEAD_CONTENT      <dbl> 11.6, 0.0, 0.0, 0.0, 18.0, 11.6, 0.0, 0....
## $ HALSTEAD_DIFFICULTY   <dbl> 2.67, 0.00, 0.00, 0.00, 3.50, 2.67, 0.00...
## $ HALSTEAD_EFFORT       <dbl> 82.3, 0.0, 0.0, 0.0, 220.9, 82.3, 0.0, 0...
## $ HALSTEAD_ERROR_EST    <dbl> 0.01, 0.00, 0.00, 0.00, 0.02, 0.01, 0.00...
## $ HALSTEAD_LENGTH       <dbl> 11, 1, 1, 1, 19, 11, 1, 1, 1, 29, 19, 1,...
## $ HALSTEAD_LEVEL        <dbl> 0.38, 0.00, 0.00, 0.00, 0.29, 0.38, 0.00...
## $ HALSTEAD_PROG_TIME    <dbl> 4.57, 0.00, 0.00, 0.00, 12.27, 4.57, 0.0...
## $ HALSTEAD_VOLUME       <dbl> 30.9, 0.0, 0.0, 0.0, 63.1, 30.9, 0.0, 0....
## $ NUM_OPERANDS          <dbl> 4, 0, 0, 0, 7, 4, 0, 0, 0, 10, 7, 0, 7, ...
## $ NUM_OPERATORS         <dbl> 7, 1, 1, 1, 12, 7, 1, 1, 1, 19, 12, 1, 1...
## $ NUM_UNIQUE_OPERANDS   <dbl> 3, 0, 0, 0, 5, 3, 0, 0, 0, 8, 5, 0, 5, 0...
## $ NUM_UNIQUE_OPERATORS  <dbl> 4, 1, 1, 1, 5, 4, 1, 1, 1, 6, 5, 1, 5, 1...
## $ LOC_TOTAL             <dbl> 5, 3, 3, 3, 12, 5, 3, 3, 3, 13, 12, 3, 1...
## $ Defective             <fct> N, N, N, N, N, N, N, N, N, N, N, N, N, N...
```

8.12 Other libraries and tricks

The `lubridate` package contains a number of functions facilitating the conversion of text to POSIX dates. As an example, consider the following code. We may use this, for example, with time series.

For example https://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf

```
library(lubridate)
dates <- c("15/02/2013", "15 Feb 13", "It happened on 15 02 '13")
dmy(dates)
```

```
## [1] "2013-02-15" "2013-02-15" "2013-02-15"
```

Part VI

Supervised Models

Chapter 9

Supervised Classification

A classification problem can be defined as the induction, from a dataset $\mathcal{D}$, of a classification function ψ that, given the attribute vector of an instance/example, returns a class c. A regression problem, on the other hand, returns an numeric value.

Dataset, $\mathcal{D}$, is typically composed of n attributes and a class attribute C.

Att_1	...	Att_n	$Class$
a_{11}	...	a_{1n}	c_1
a_{21}	...	a_{2n}	c_2
...	...	...	...
a_{m1}	...	a_{mn}	c_m

Columns are usually called *attributes* or *features*. Typically, there is a *class* attribute, which can be numeric or discrete. When the class is numeric, it is a regression problem. With discrete values, we can talk about binary classification or multiclass (multinomial classification) when we have more than three values. There are variants such *multi-label* classification (we will cover these in the advanced models section).

Once we learn a model, new instances are classified. As shown in the next figure.

We have multiple types of models such as *classification trees*, *rules*, *neural networks*, and *probabilistic classifiers* that can be used to classify instances.

Fernandez et al provide an extensive comparison of 176 classifiers using the UCI dataset (Fernández-Delgado et al., 2014).

We will show the use of different classification techniques in the problem of defect prediction as running example. In this example,the different datasets are composed of classical metrics (*Halstead* or *McCabe* metrics) based on counts of operators/operands and like or object-oriented metrics (e.g. Chidamber and Kemerer) and the class attribute indicating whether the module or class was defective.

9.1 Classification Trees

There are several packages for inducing classification trees, for example with the party package (recursive partitioning):

```
library(foreign) # To load arff file
library(party) # Build a decision tree
library(caret)
```

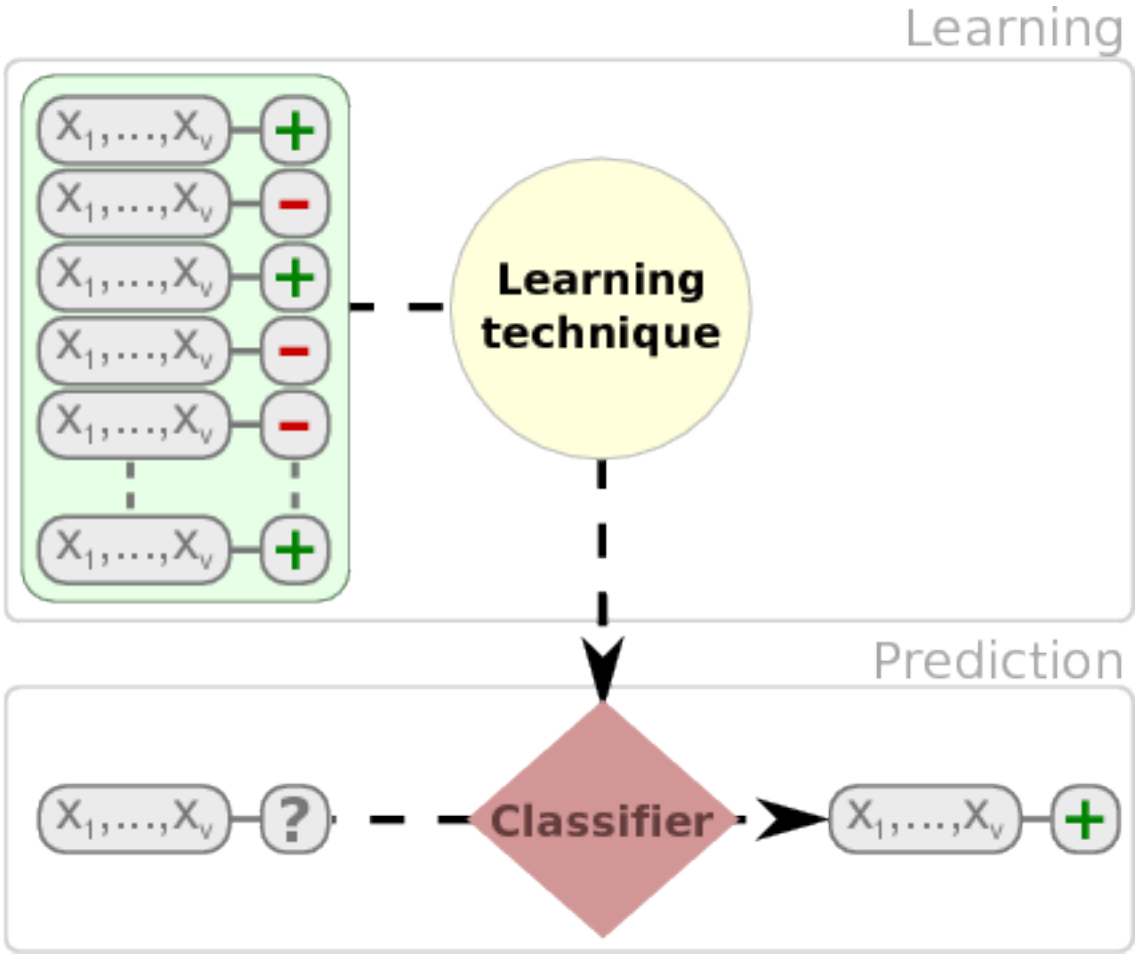


Figure 9.1: Supervised Classification

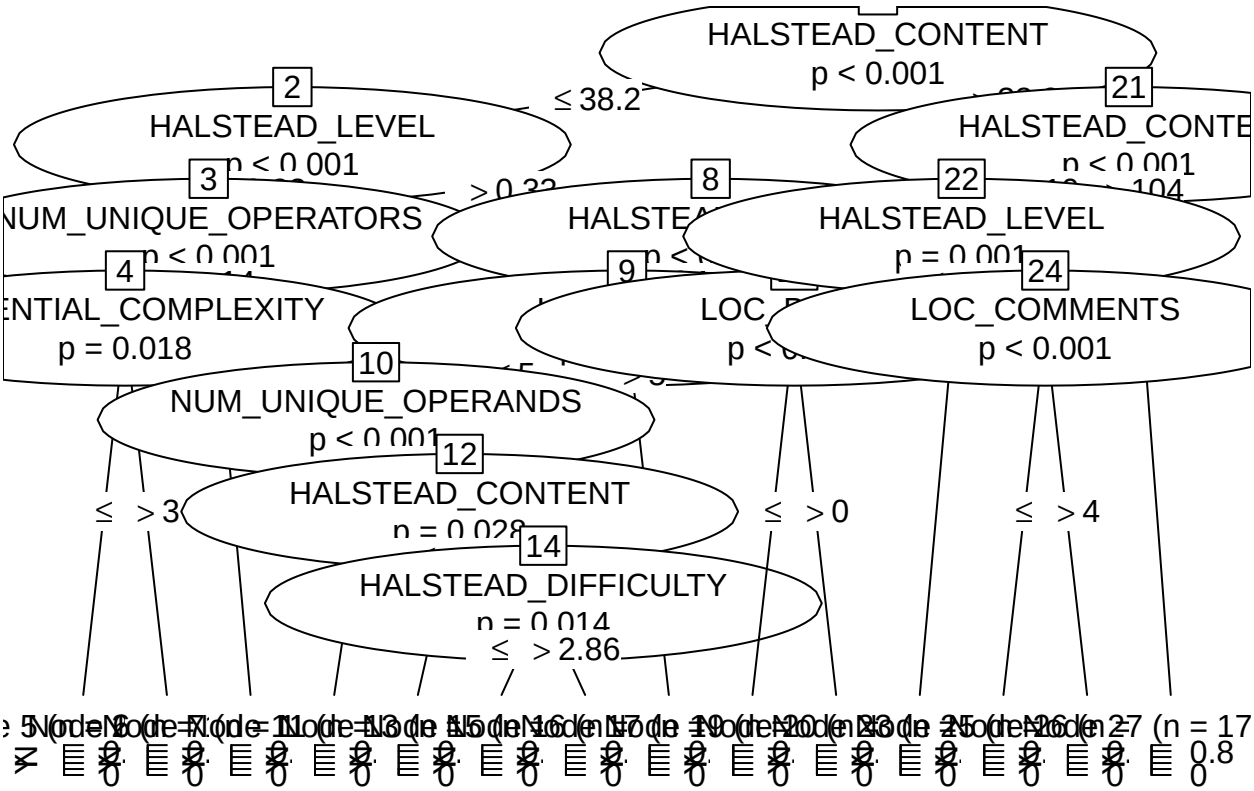
```
jm1 <- read.arff("./datasets/defectPred/D1/JM1.arff")
str(jm1)
```

```
## 'data.frame':    9593 obs. of  22 variables:
##  $ LOC_BLANK            : num  447 37 11 106 101 67 105 18 39 143 ...
##  $ BRANCH_COUNT         : num  826 29 405 240 464 187 344 47 163 67 ...
##  $ LOC_CODE_AND_COMMENT : num  12 8 0 7 11 4 9 0 1 7 ...
##  $ LOC_COMMENTS         : num  157 42 17 344 75 1 40 10 6 49 ...
##  $ CYCLOMATIC_COMPLEXITY: num  470 19 404 127 263 94 207 24 94 34 ...
##  $ DESIGN_COMPLEXITY    : num  385 19 2 105 256 63 171 13 67 25 ...
##  $ ESSENTIAL_COMPLEXITY : num  113 6 1 33 140 27 58 1 3 1 ...
##  $ LOC_EXECUTABLE       : num  2824 133 814 952 1339 ...
##  $ HALSTEAD_CONTENT     : num  210 108 101 218 106 ...
##  $ HALSTEAD_DIFFICULTY  : num  384.4 46.3 206 215.2 337.4 ...
##  $ HALSTEAD_EFFORT      : num  31079782 232044 4294926 10100867 12120796 ...
##  $ HALSTEAD_ERROR_EST   : num  26.95 1.67 6.95 15.65 11.98 ...
##  $ HALSTEAD_LENGTH      : num  8441 685 2033 5669 4308 ...
##  $ HALSTEAD_LEVEL       : num  0 0.02 0 0 0 0.02 0 0.03 0.01 0.02 ...
##  $ HALSTEAD_PROG_TIME   : num  1726655 12891 238607 561159 673378 ...
##  $ HALSTEAD_VOLUME      : num  80843 5009 20848 46944 35928 ...
##  $ NUM_OPERANDS         : num  3021 295 813 2301 1556 ...
##  $ NUM_OPERATORS        : num  5420 390 1220 3368 2752 ...
##  $ NUM_UNIQUE_OPERANDS  : num  609 121 811 262 226 167 279 47 117 355 ...
##  $ NUM_UNIQUE_OPERATORS : num  155 38 411 49 98 27 105 18 52 23 ...
##  $ LOC_TOTAL            : num  3442 222 844 1411 1532 ...
##  $ Defective            : Factor w/ 2 levels "N","Y": 2 2 2 2 2 2 2 2 1 2 ...
```

```
# Stratified partition (training and test sets)
set.seed(1234)
inTrain <- createDataPartition(y=jm1$Defective,p=.60,list=FALSE)
jm1.train <- jm1[inTrain,]
jm1.test <- jm1[-inTrain,]

jm1.formula <- jm1$Defective ~ . # formula approach: defect as dependent variable and the rest as indep
jm1.ctree <- ctree(jm1.formula, data = jm1.train)

# predict on test data
pred <- predict(jm1.ctree, newdata = jm1.test)
# check prediction result
table(pred, jm1.test$Defective)
```

```
##
## pred    N    Y
##    N   82    3
##    Y 3051  700
```

```
plot(jm1.ctree)
```

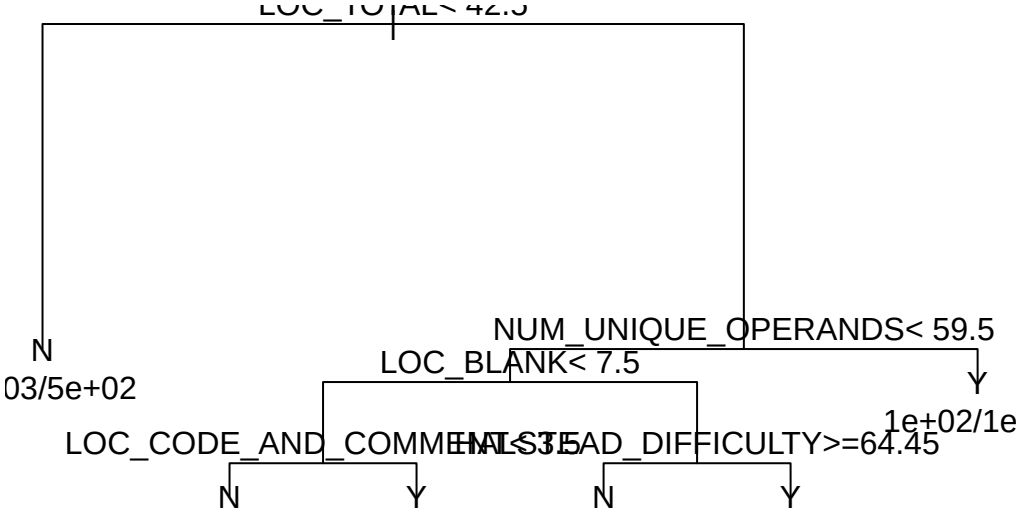


Using the C50 package, there are two ways, specifying train and testing

```
library(C50)
require(utils)
# c50t <- C5.0(jm1.train[,-ncol(jm1.train)], jm1.train[,ncol(jm1.train)])
c50t2 <- C5.0(Defective ~ ., jm1.train)
summary(c50t)
plot(c50t)
c50tPred <- predict(c50t, jm1.train)
# table(c50tPred, jm1.train$Defective)
```

Using the 'rpart' package

```
# Using the 'rpart' package
library(rpart)
jm1.rpart <- rpart(Defective ~ ., data=jm1.train, parms = list(prior = c(.65,.35), split = "information
# par(mfrow = c(1,2), xpd = NA)
plot(jm1.rpart)
text(jm1.rpart, use.n = TRUE)
```




```
jm1.rpart
```

```
## n= 5757
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 5757 2010.0 N (0.650 0.350)
##    2) LOC_TOTAL< 42.5 4309 1030.0 N (0.744 0.256) *
##    3) LOC_TOTAL>=42.5 1448  743.0 Y (0.430 0.570)
##      6) NUM_UNIQUE_OPERANDS< 59.5 1206  655.0 Y (0.473 0.527)
##       12) LOC_BLANK< 7.5 449  210.0 N (0.562 0.438)
##         24) LOC_CODE_AND_COMMENT< 3.5 412  172.0 N (0.599 0.401) *
##         25) LOC_CODE_AND_COMMENT>=3.5 37   13.5 Y (0.262 0.738) *
##       13) LOC_BLANK>=7.5 757  385.0 Y (0.425 0.575)
##         26) HALSTEAD_DIFFICULTY>=64.5 55   13.4 N (0.741 0.259) *
##         27) HALSTEAD_DIFFICULTY< 64.5 702  347.0 Y (0.406 0.594) *
##      7) NUM_UNIQUE_OPERANDS>=59.5 242   87.6 Y (0.258 0.742) *
```

```
library(rpart.plot)
# asRules(jm1.rpart)
# fancyRpartPlot(jm1.rpart)
```

9.2 Rules

C5 Rules

```
library(C50)
c50r <- C5.0(jm1.train[,-ncol(jm1.train)], jm1.train[,ncol(jm1.train)], rules = TRUE)
summary(c50r)
```

```
##
## Call:
## C5.0.default(x = jm1.train[, -ncol(jm1.train)], y =
##  jm1.train[, ncol(jm1.train)], rules = TRUE)
##
##
```

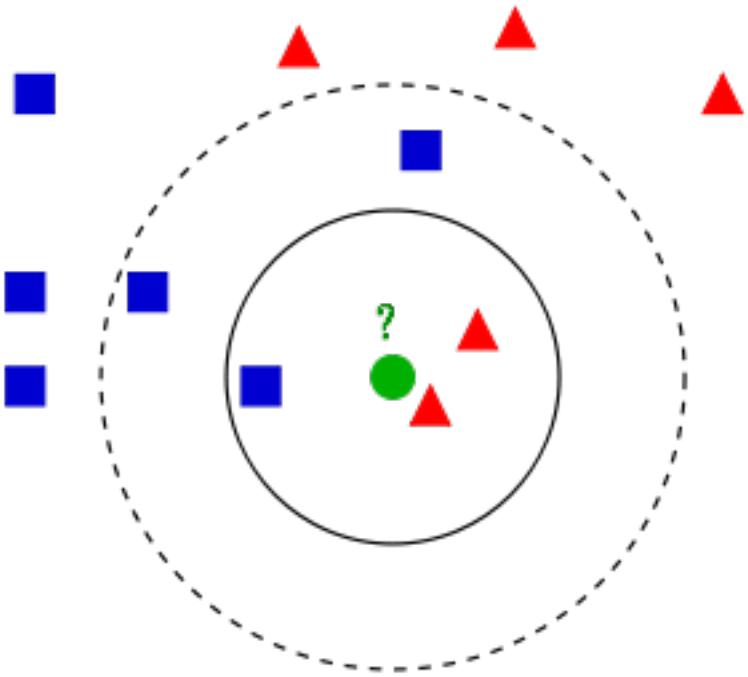
```
## C5.0 [Release 2.07 GPL Edition]      Wed Dec 26 21:32:44 2018
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 5757 cases (22 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (5512/923, lift 1.0)
##  CYCLOMATIC_COMPLEXITY <= 67
##  NUM_UNIQUE_OPERANDS <= 59
##  ->  class N  [0.832]
##
## Rule 2: (11/1, lift 4.6)
##  LOC_BLANK > 3
##  DESIGN_COMPLEXITY > 3
##  LOC_EXECUTABLE > 31
##  LOC_EXECUTABLE <= 33
##  LOC_TOTAL <= 42
##  ->  class Y  [0.846]
##
## Rule 3: (25/5, lift 4.2)
##  CYCLOMATIC_COMPLEXITY > 67
##  ->  class Y  [0.778]
##
## Rule 4: (242/110, lift 3.0)
##  NUM_UNIQUE_OPERANDS > 59
##  LOC_TOTAL > 42
##  ->  class Y  [0.545]
##
## Default class: N
##
##
## Evaluation on training data (5757 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       4 1025(17.8%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##    4589   112    (a): class N
##     913   143    (b): class Y
##
##
##  Attribute usage:
##
##   99.95% NUM_UNIQUE_OPERANDS
##   96.18% CYCLOMATIC_COMPLEXITY
##    4.39% LOC_TOTAL
```

```
##    0.19% LOC_BLANK
##    0.19% DESIGN_COMPLEXITY
##    0.19% LOC_EXECUTABLE
##
##
## Time: 0.1 secs
```

```
# c50rPred <- predict(c50r, jm1.train)
# table(c50rPred, jm1.train$Defective)
```

9.3 Distanced-based Methods

In this case, there is no model as such. Given a new instance to classify, this approach finds the closest k-neighbours to the given instance.



(Source: Wikipedia - https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)

```
library(class)
m1 <- knn(train=jm1.train[,-22], test=jm1.test[,-22], cl=jm1.train[,22], k=3)

table(jm1.test[,22],m1)
```

```
##    m1
##        N    Y
##   N 2827  306
##   Y  553  150
```

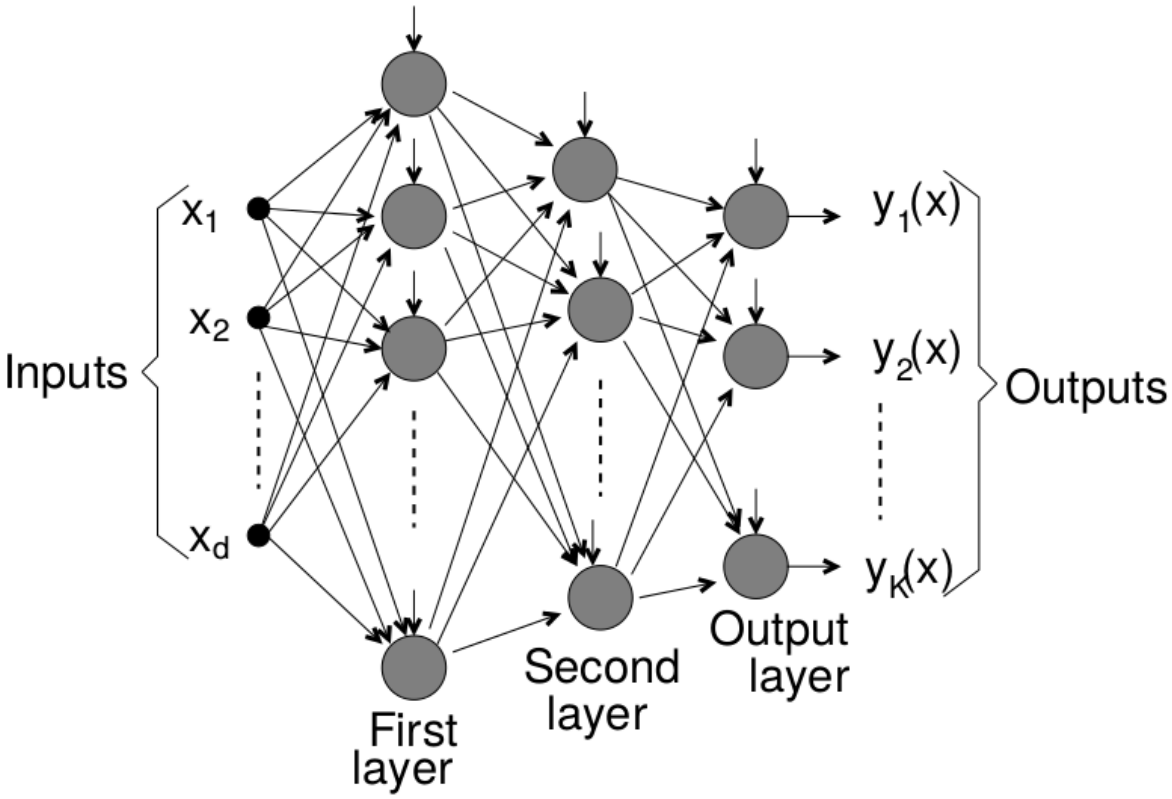


Figure 9.2: Neural Networks

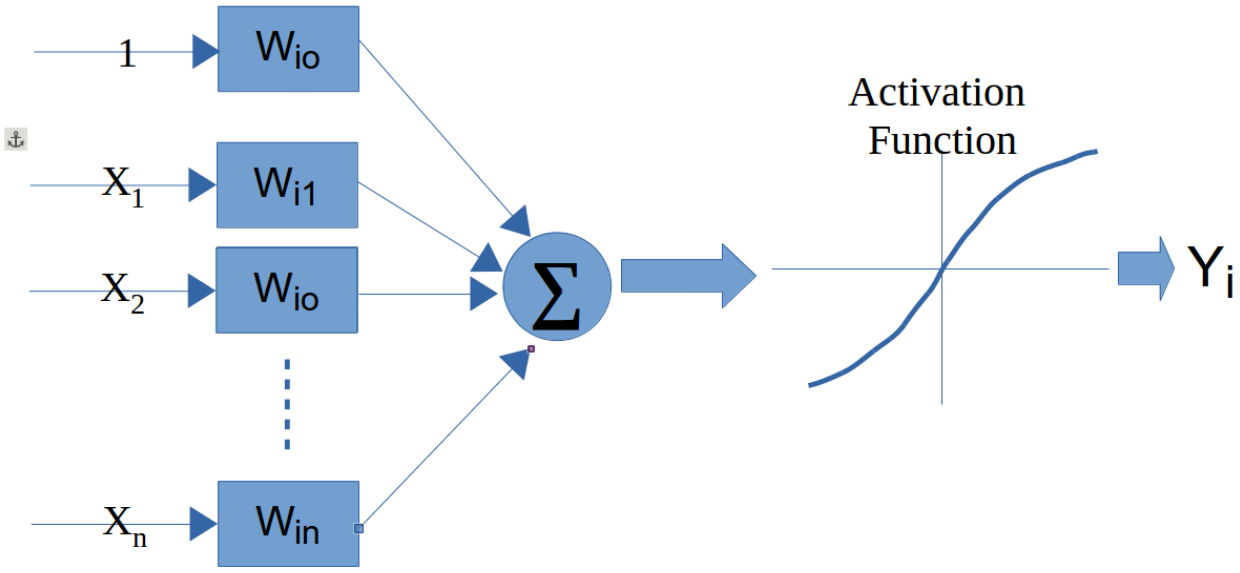


Figure 9.3: Neural Networks

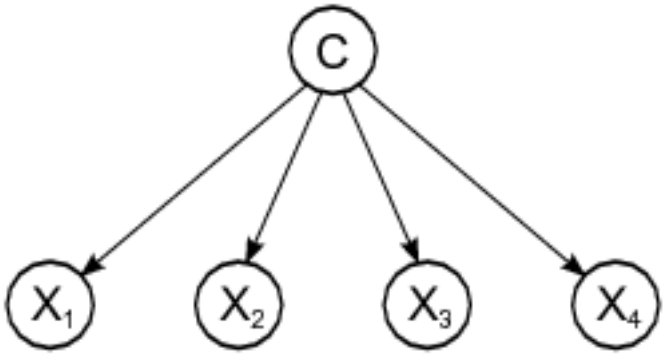
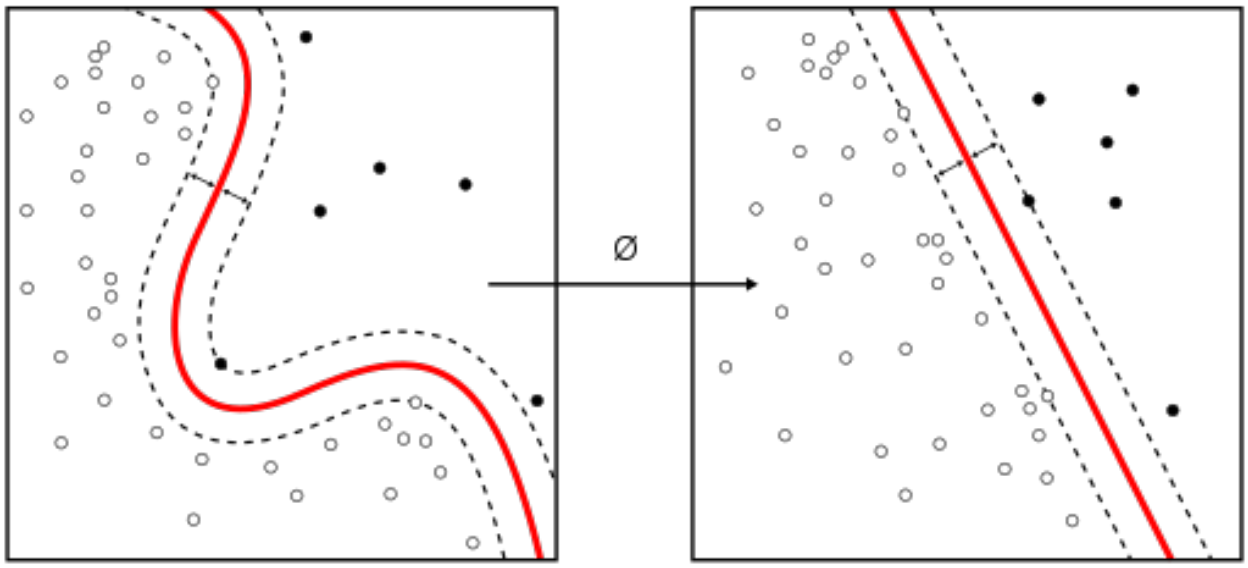


Figure 9.4: Naive Bayes

9.4 Neural Networks

9.5 Support Vector Machine



(Source: wikipedia https://en.wikipedia.org/wiki/Support_vector_machine)

9.6 Probabilistic Methods

9.6.1 Naive Bayes

Probabilistic graphical model assigning a probability to each possible outcome $p(C_k, x_1, \ldots, x_n)$

Using the `klaR` package with `caret`:

```
library(caret)
library(klaR)
```

```
## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

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

```
model <- NaiveBayes(Defective ~ ., data = jm1.train)
predictions <- predict(model, jm1.test[,-22])
confusionMatrix(predictions$class, jm1.test$Defective)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    N    Y
##          N 2990  548
##          Y  143  155
##
##                Accuracy : 0.82
##                  95% CI : (0.807, 0.832)
##     No Information Rate : 0.817
##     P-Value [Acc > NIR] : 0.317
##
##                   Kappa : 0.225
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.954
##             Specificity : 0.220
##          Pos Pred Value : 0.845
##          Neg Pred Value : 0.520
##              Prevalence : 0.817
##          Detection Rate : 0.779
##    Detection Prevalence : 0.922
##       Balanced Accuracy : 0.587
##
##        'Positive' Class : N
##
```

Using the `e1071` package:

```
library (e1071)
n1 <-naiveBayes(jm1.train$Defective ~ ., data=jm1.train)

# Show first 3 results using 'class'
head(predict(n1,jm1.test, type = c("class")),3) # class by default
```

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

```
# Show first 3 results using 'raw'
head(predict(n1,jm1.test, type = c("raw")),3)
```

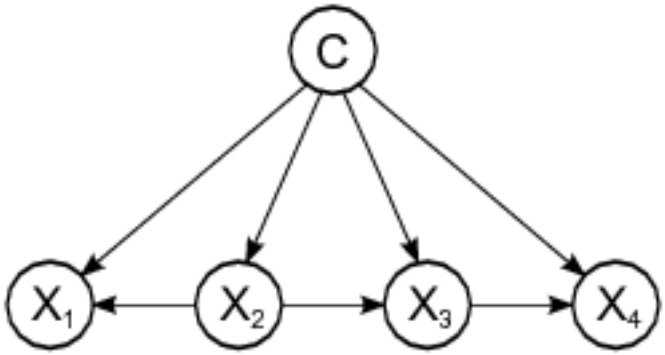


Figure 9.5: Naive Bayes

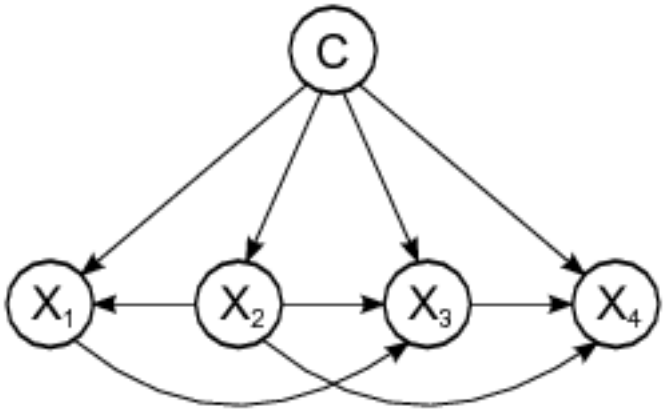


Figure 9.6: Naive Bayes

```
##             N     Y
## [1,] 1.000000 0.000
## [2,] 0.000000 1.000
## [3,] 0.000754 0.999
```

There are other variants such as TAN and KDB that do not assume the independece condition allowin us more complex structures.

A comprehensice comparison of

9.7 Linear Discriminant Analysis (LDA)

One classical approach to classification is Linear Discriminant Analysis (LDA), a generalization of Fisher's linear discriminant, as a method used to find a linear combination of features to separate two or more classes.

```
ldaModel <- train (Defective ~ ., data=jm1.train, method="lda", preProc=c("center","scale"))
ldaModel
```

```
## Linear Discriminant Analysis
##
## 5757 samples
##   21 predictors
```

```
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 5757, 5757, 5757, 5757, 5757, 5757, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.817     0.135
```

We can observe that we are training our model using `Defective ~ .` as a formula were `Defective` is the class variable separed by `~` and the ´.´ means the rest of the variables. Also, we are using a filter for the training data to (preProc) to center and scale.

Also, as stated in the documentation about the `train` method : > http://topepo.github.io/caret/training.html

```
ctrl <- trainControl(method = "repeatedcv",repeats=3)
ldaModel <- train (Defective ~ ., data=jm1.train, method="lda", trControl=ctrl, preProc=c("center","scal

ldaModel
```

```
## Linear Discriminant Analysis
##
## 5757 samples
##   21 predictors
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 5182, 5181, 5182, 5182, 5180, 5181, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.816     0.129
```

Instead of accuracy we can activate other metrics using `summaryFunction=twoClassSummary` such as `ROC`, `sensitivity` and `specificity`. To do so, we also need to speficy `classProbs=TRUE`.

```
ctrl <- trainControl(method = "repeatedcv",repeats=3, classProbs=TRUE, summaryFunction=twoClassSummary)
ldaModel3xcv10 <- train (Defective ~ ., data=jm1.train, method="lda", trControl=ctrl, preProc=c("center

ldaModel3xcv10
```

```
## Linear Discriminant Analysis
##
## 5757 samples
##   21 predictors
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 5181, 5181, 5181, 5181, 5182, 5182, ...
## Resampling results:
##
##   ROC    Sens   Spec
##   0.706  0.974  0.121
```

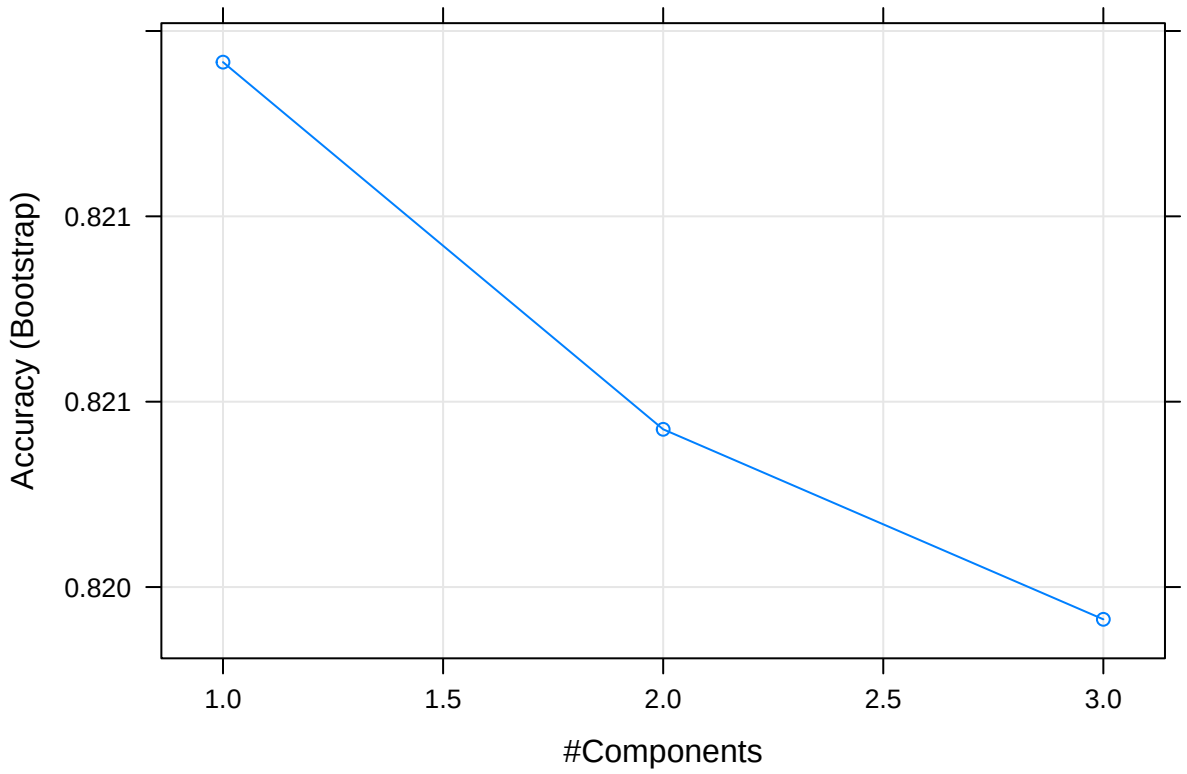
Most methods have parameters that need to be optimised and that is one of the

```
plsFit3x10cv <- train (Defective ~ ., data=jm1.train, method="pls", trControl=trainControl(classProbs=T
```

```
plsFit3x10cv
```

```
## Partial Least Squares
##
## 5757 samples
##   21 predictors
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 5757, 5757, 5757, 5757, 5757, 5757, ...
## Resampling results across tuning parameters:
##
##   ncomp  Accuracy  Kappa
##   1      0.821     0.0548
##   2      0.820     0.0768
##   3      0.820     0.0762
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 1.
```

```
plot(plsFit3x10cv)
```



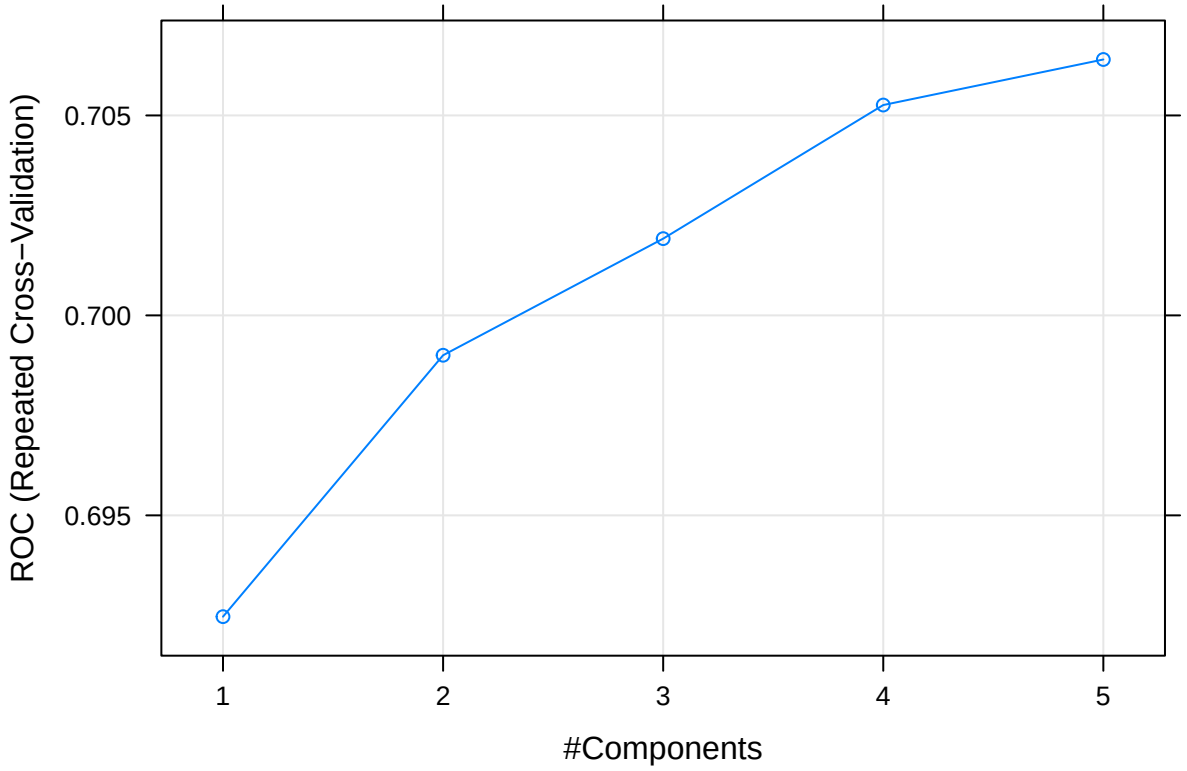
The parameter `tuneLength` allow us to specify the number values per parameter to consider.

```
plsFit3x10cv <- train (Defective ~ ., data=jm1.train, method="pls", trControl=ctrl, metric="ROC", tuneL
```

```
plsFit3x10cv
```

```
## Partial Least Squares
##
## 5757 samples
##   21 predictors
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 5181, 5181, 5181, 5182, 5181, 5181, ...
## Resampling results across tuning parameters:
##
##   ncomp  ROC    Sens   Spec
##   1      0.692  0.995  0.0454
##   2      0.699  0.991  0.0634
##   3      0.702  0.990  0.0641
##   4      0.705  0.989  0.0656
##   5      0.706  0.989  0.0663
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 5.
```

```
plot(plsFit3x10cv)
```



Finally to predict new cases, `caret` will use the best classfier obtained for prediction.

```
plsProbs <- predict(plsFit3x10cv, newdata = jm1.test, type = "prob")
```

```
plsClasses <- predict(plsFit3x10cv, newdata = jm1.test, type = "raw")
```

```
confusionMatrix(data=plsClasses,jm1.test$Defective)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    N    Y
##          N 3108  643
##          Y   25   60
##
##                Accuracy : 0.826
##                  95% CI : (0.813, 0.838)
##     No Information Rate : 0.817
##     P-Value [Acc > NIR] : 0.0743
##
##                   Kappa : 0.117
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9920
##             Specificity : 0.0853
##          Pos Pred Value : 0.8286
##          Neg Pred Value : 0.7059
##              Prevalence : 0.8167
##          Detection Rate : 0.8102
##    Detection Prevalence : 0.9778
##       Balanced Accuracy : 0.5387
##
##        'Positive' Class : N
##
```

9.7.1 Predicting the number of defects (numerical class)

From the Bug Prediction Repository (BPR) http://bug.inf.usi.ch/download.php

Some datasets contain CK and other 11 object-oriented metrics for the last version of the system plus categorized (with severity and priority) post-release defects. Using such dataset:

```
jdt <- read.csv("./datasets/defectPred/BPD/single-version-ck-oo-EclipseJDTCore.csv", sep=";")

# We just use the number of bugs, so we removed others
jdt$classname <- NULL
jdt$nonTrivialBugs <- NULL
jdt$majorBugs <- NULL
jdt$minorBugs <- NULL
jdt$criticalBugs <- NULL
jdt$highPriorityBugs <- NULL
jdt$X <- NULL

# Caret
library(caret)

# Split data into training and test datasets
set.seed(1)
inTrain <- createDataPartition(y=jdt$bugs,p=.8,list=FALSE)
```

```
jdt.train <- jdt[inTrain,]
jdt.test <- jdt[-inTrain,]
```

```
ctrl <- trainControl(method = "repeatedcv",repeats=3)
glmModel <- train (bugs ~ ., data=jdt.train, method="glm", trControl=ctrl, preProc=c("center","scale"))
glmModel
```

```
## Generalized Linear Model
##
## 798 samples
##  17 predictors
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 719, 718, 719, 718, 718, 718, ...
## Resampling results:
##
##   RMSE   Rsquared  MAE
##   0.786  0.397     0.417
```

Others such as Elasticnet:

```
glmnetModel <- train (bugs ~ ., data=jdt.train, method="glmnet", trControl=ctrl, preProc=c("center","sca
glmnetModel
```

```
## glmnet
##
## 798 samples
##  17 predictors
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 718, 718, 719, 718, 718, 718, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda  RMSE   Rsquared  MAE
##   0.10   0.0012  0.774  0.397     0.417
##   0.10   0.0120  0.788  0.384     0.421
##   0.10   0.1202  0.794  0.369     0.424
##   0.55   0.0012  0.773  0.397     0.416
##   0.55   0.0120  0.794  0.375     0.422
##   0.55   0.1202  0.805  0.371     0.435
##   1.00   0.0012  0.773  0.397     0.416
##   1.00   0.0120  0.794  0.373     0.423
##   1.00   0.1202  0.812  0.365     0.449
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.0012.
```

9.8 Binary Logistic Regression (BLR)

Binary Logistic Regression (BLR) can models fault-proneness as follows

$$fp(X) = \frac{e^{logit()}}{1 + e^{logit(X)}}$$

where the simplest form for logit is:

$logit(X) = c_0 + c_1 X$

```
jdt <- read.csv("./datasets/defectPred/BPD/single-version-ck-oo-EclipseJDTCore.csv", sep=";")

# Caret
library(caret)

# Convert the response variable into a boolean variable (0/1)
jdt$bugs[jdt$bugs>=1]<-1

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

# 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 $logit(X) = c_0 + c_1 X$

```
# 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
## -3.439  -0.601  -0.531  -0.492   2.106
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.10194    0.13457  -15.62  < 2e-16 ***
## cbo          0.05342    0.00721    7.41  1.3e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 796.98  on 797  degrees of freedom
## Residual deviance: 711.81  on 796  degrees of freedom
## AIC: 715.8
##
## Number of Fisher Scoring iterations: 5
```

Predict a single point:

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

```
##     1
## 0.125
```

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

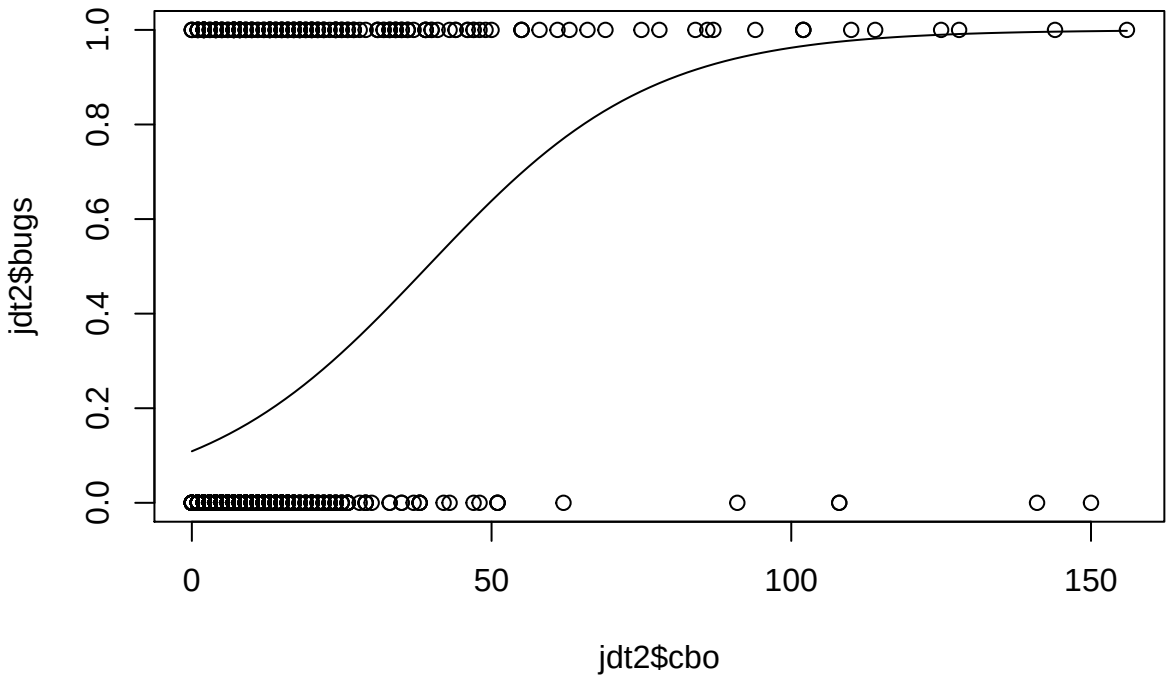
```
results <- predict(glmLogit, jdtTest, type = "response")

range(jdtTrain$cbo)
```

```
## [1]   0 150
```

```
range(results)
```

```
## [1] 0.109 0.998
```

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




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

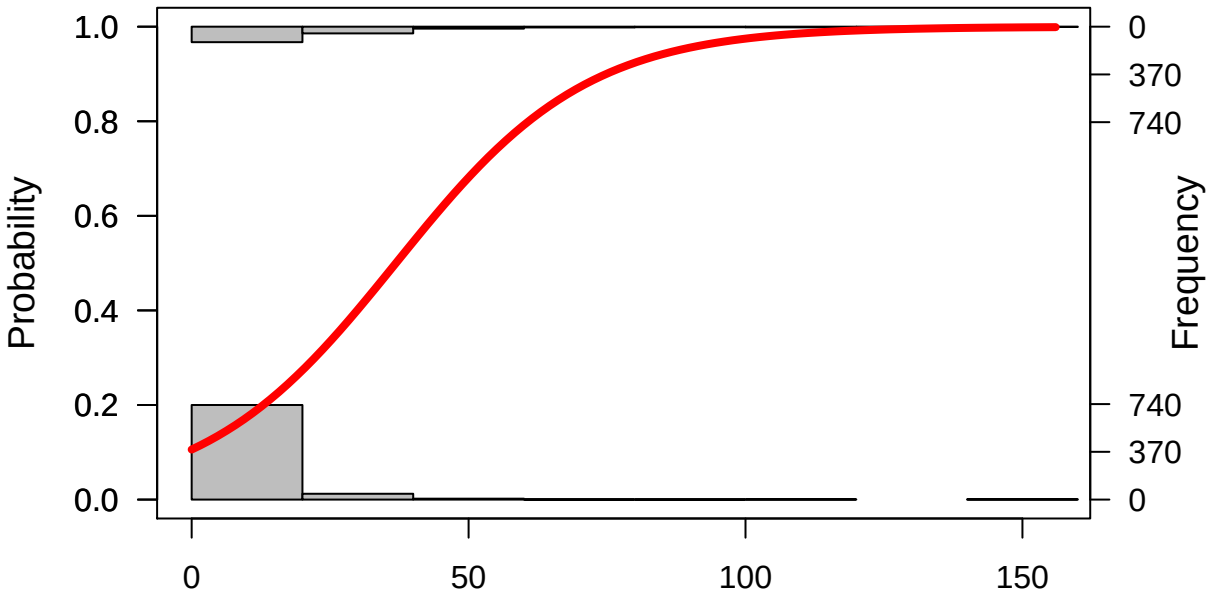
Another type of graph:

```
library(popbio)
```

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

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

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



9.9 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 10

Regression

10.1 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.

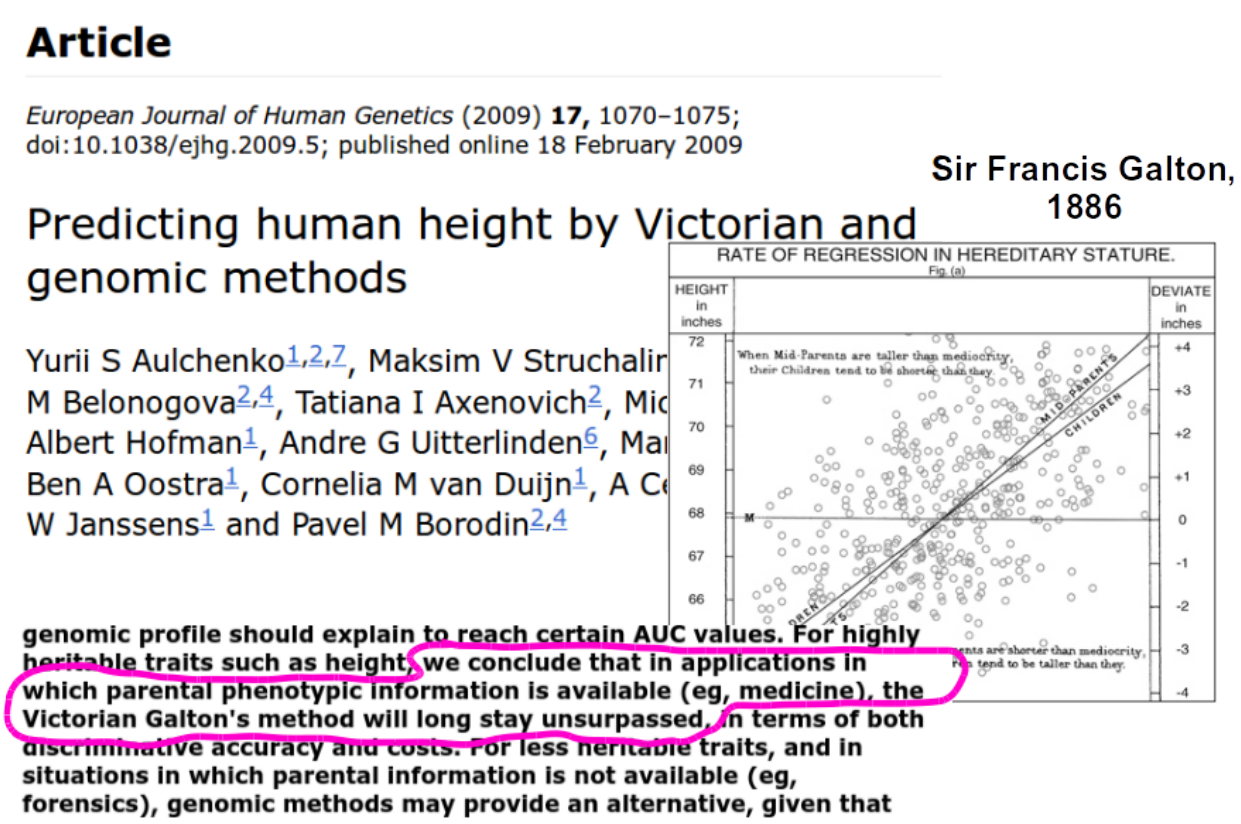
- First proposed many years ago but still very useful...

- 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.

10.1.1 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

Sir Francis Galton, 1886

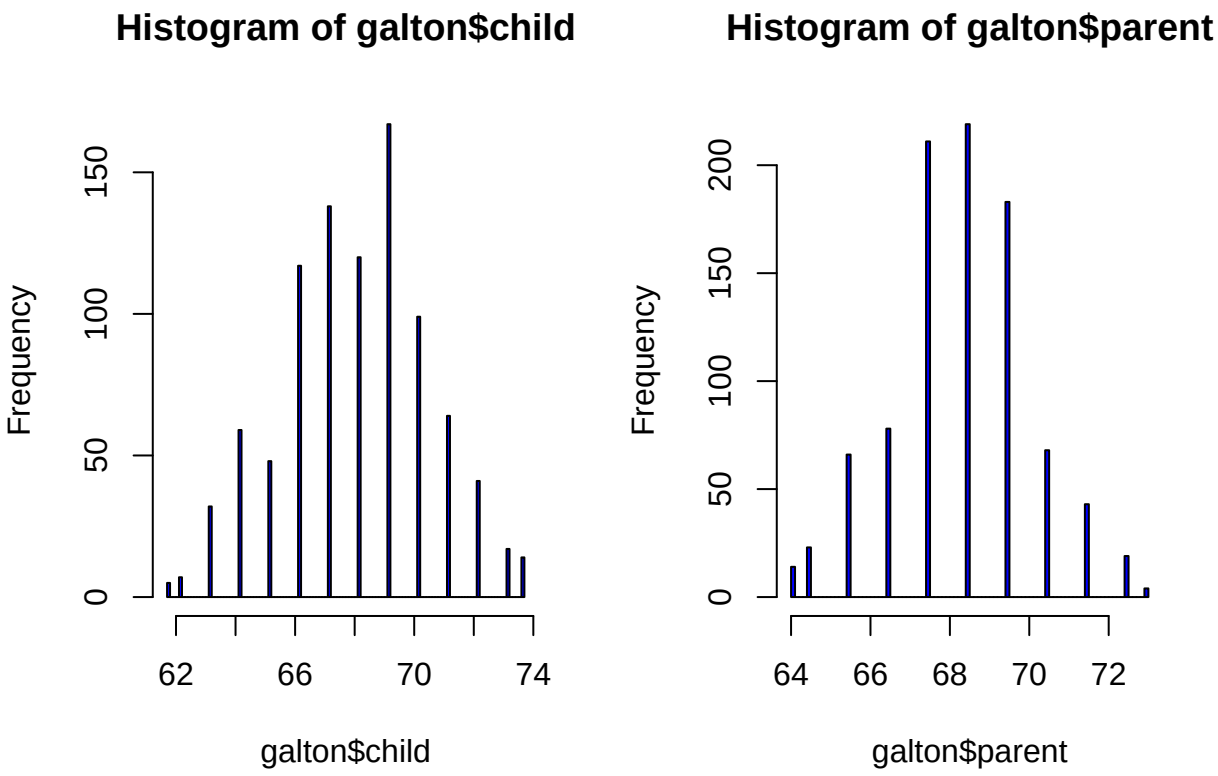
Predicting human height by Victorian and genomic methods

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

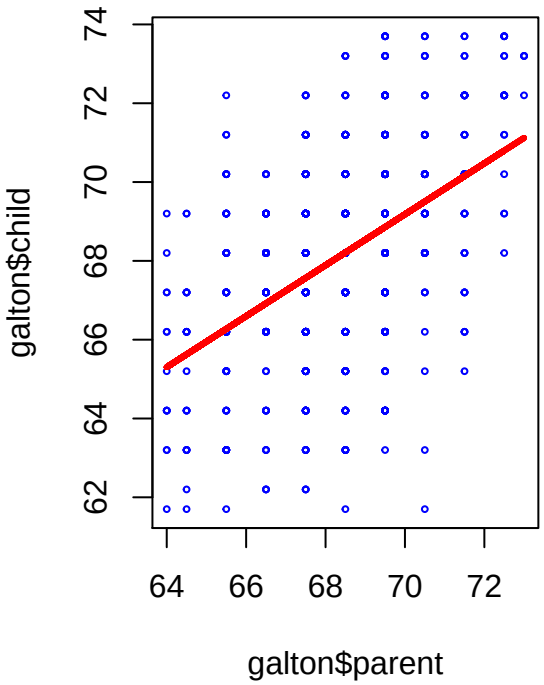
RATE OF REGRESSION IN HEREDITARY STATURE.
Fig. (a)
HEIGHT in inches
DEVIATE in inches
When Mid-Parents are taller than mediocrity, their Children tend to be shorter than they.
MID-PARENTS
CHILDREN
M

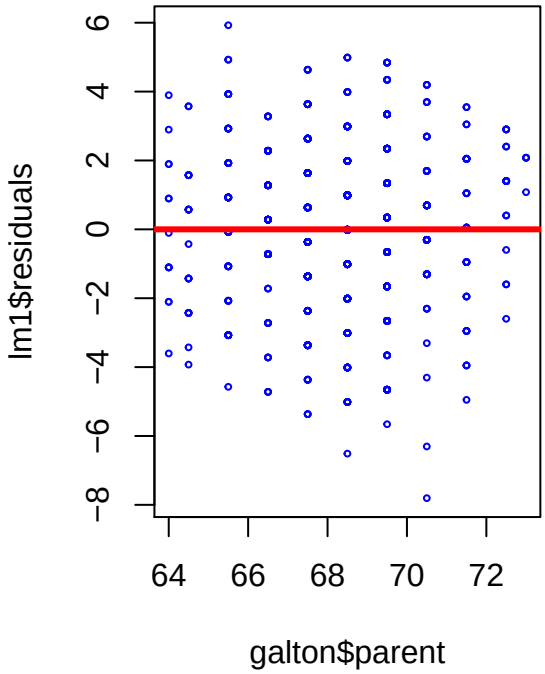
genomic profile should explain to reach certain AUC values. For highly heritable traits such as height, we conclude that in applications in which parental phenotypic information is available (eg, medicine), the Victorian Galton's method will long stay unsurpassed, in terms of both discriminative accuracy 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

Figure 10.1: Galton Data

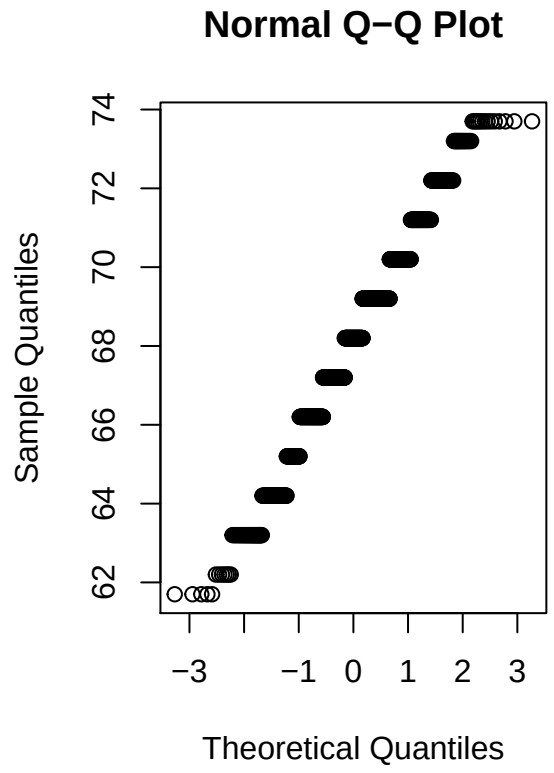


```
plot(galton$parent,galton$child,pch=1,col="blue", cex=0.4)
lm1 <- lm(galton$child ~ galton$parent)
lines(galton$parent,lm1$fitted,col="red",lwd=3)
plot(galton$parent,lm1$residuals,col="blue",pch=1, cex=0.4)
abline(c(0,0),col="red",lwd=3)
```






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



10.1.2 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.

10.1.3 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

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

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

10.1.4 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.

10.2 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.

10.2.1 Simulation example

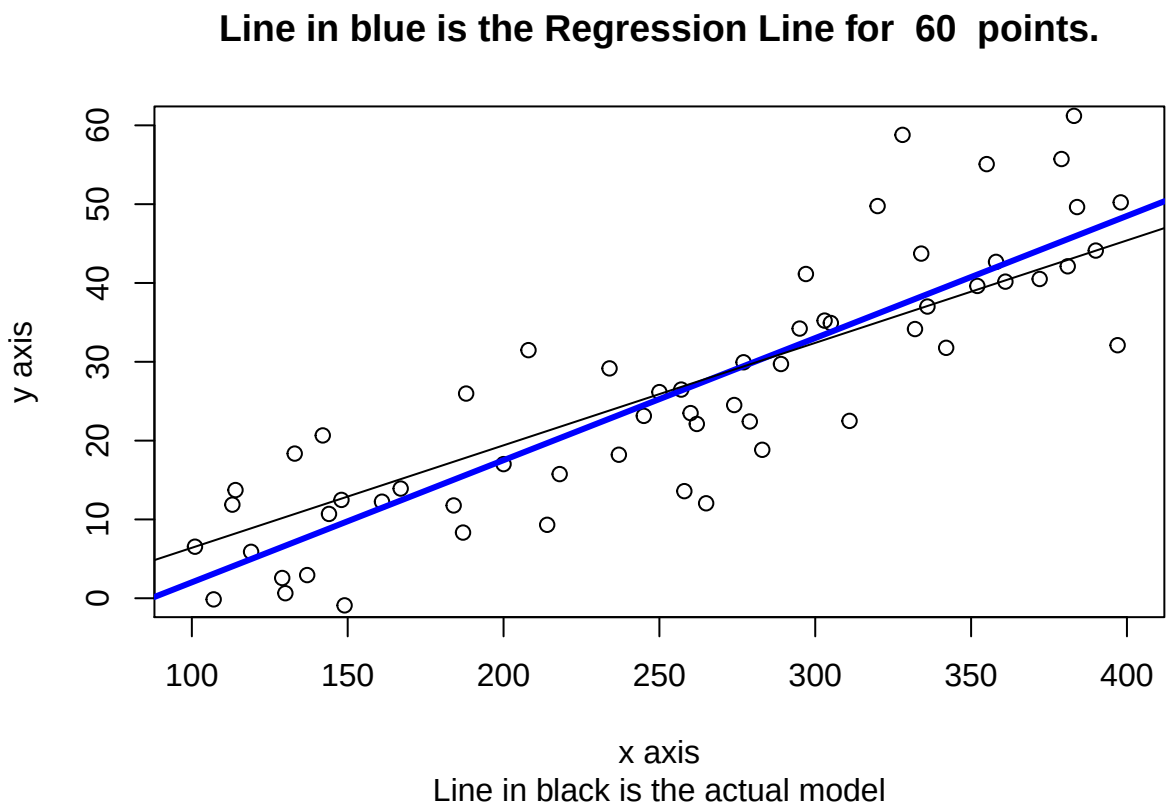
10.2.1.1 Simulate a dataset

```
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)
```

```r
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
## -15.937  -4.617  -0.923   3.797  21.442
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -13.4765     3.0320   -4.44    4e-05 ***
## x             0.1550     0.0113   13.75   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.99 on 58 degrees of freedom
## Multiple R-squared:  0.765,  Adjusted R-squared:  0.761
## F-statistic:  189 on 1 and 58 DF,  p-value: <2e-16
```

```r
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 black is
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
```



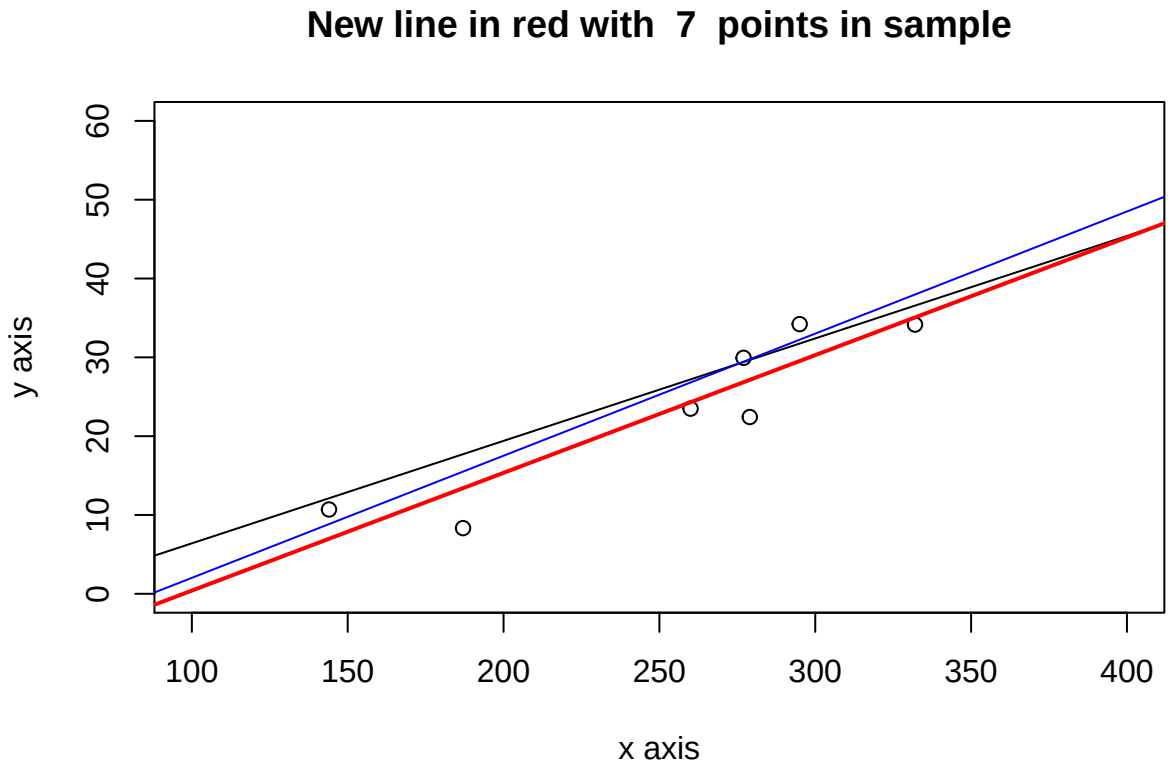
10.2.1.1.1 Subset a set of points from the same sample

```
# 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)
```

```
##
## Call:
## lm(formula = y1 ~ x1)
##
## Residuals:
##      1      2      3      4      5      6      7
##  3.722 -5.067  4.683 -4.716  3.095 -0.813 -0.904
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -14.5356     7.0962   -2.05   0.0958 .
## x1            0.1494     0.0272    5.48   0.0027 **
```

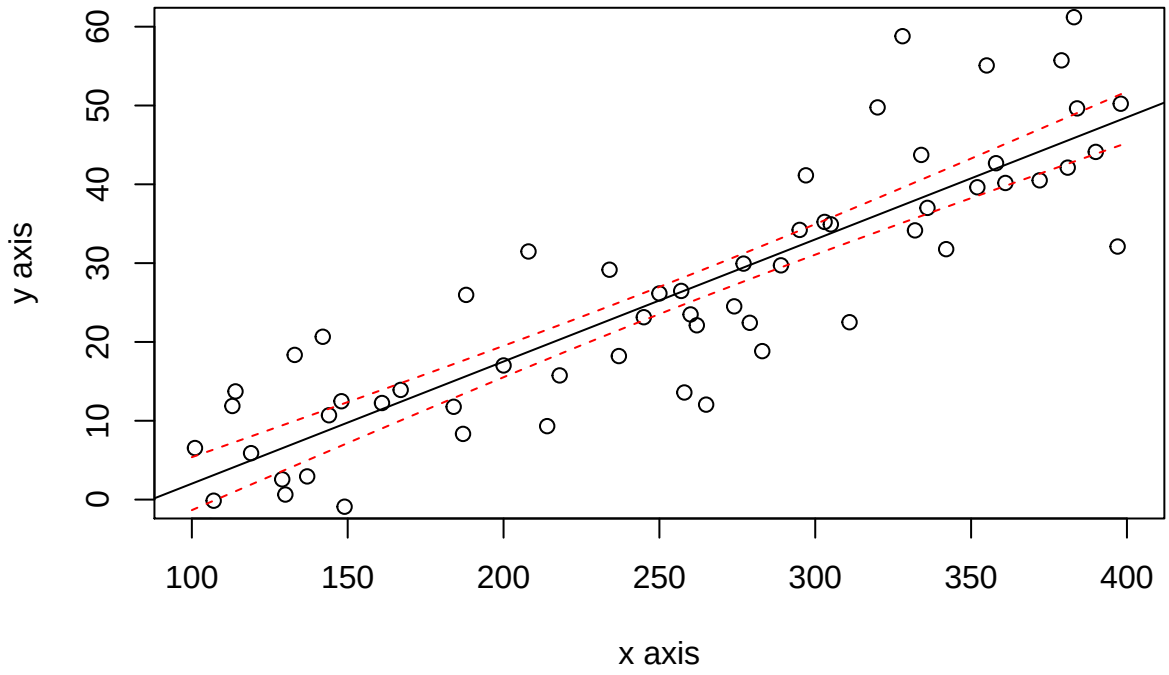
```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.35 on 5 degrees of freedom
## Multiple R-squared:  0.857,  Adjusted R-squared:  0.829
## F-statistic: 30.1 on 1 and 5 DF,  p-value: 0.00275
```

```
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
```

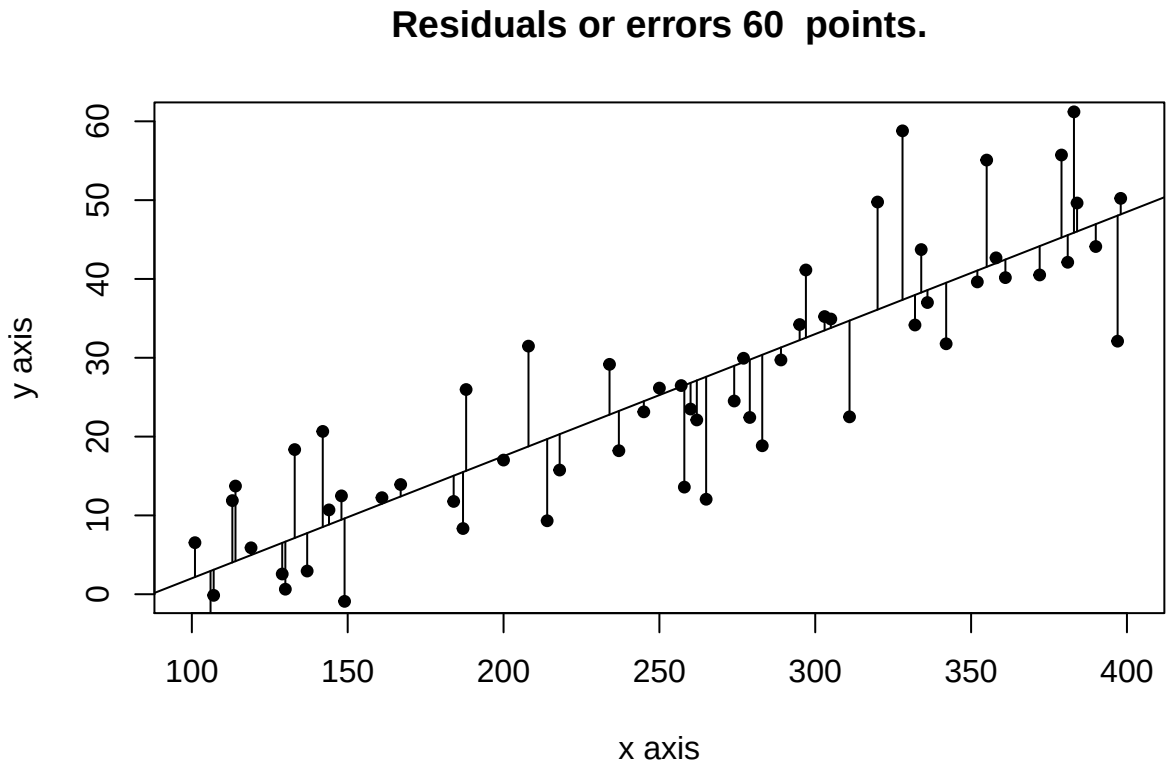


10.2.1.1.2 Compute a confidence interval on the original sample regression line

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

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)
```




```
# 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 = "", pch=19, cex=0.
title(main = paste("Residuals or errors", num_obs, " points."))
abline(newlm)
segments(x, y, x, ypredicted_x)
```

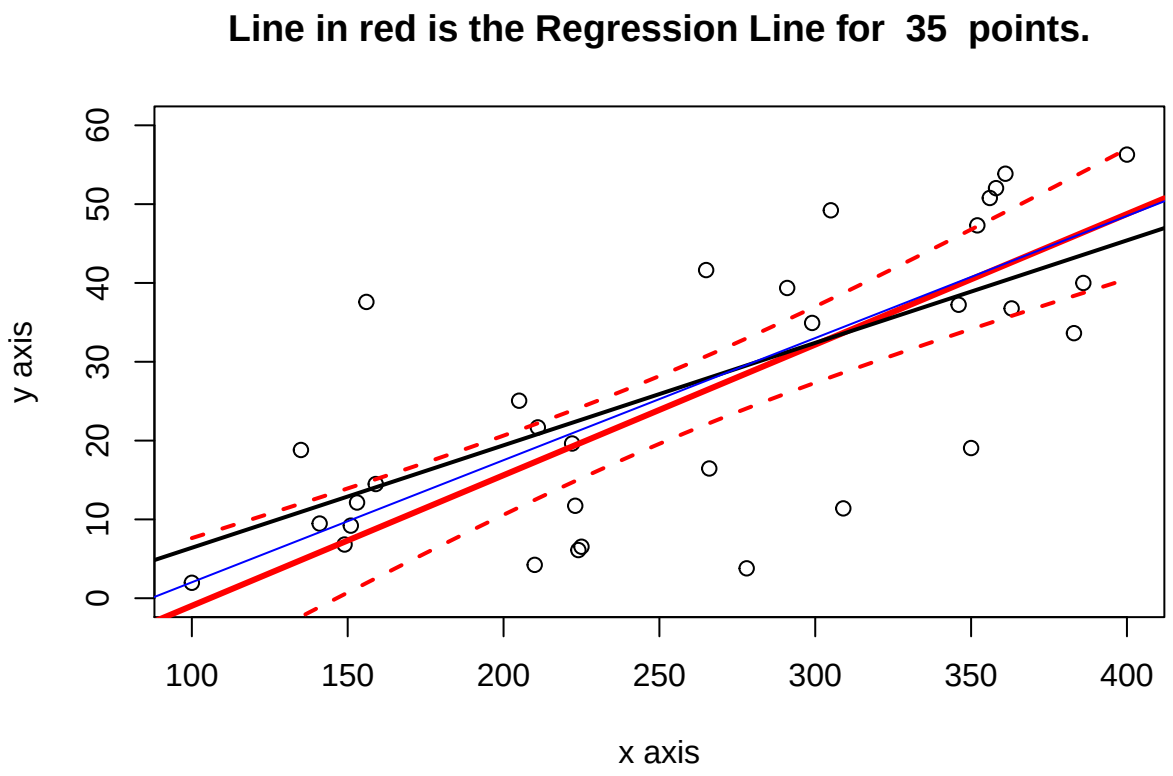


10.2.1.1.3 Take another sample from the model and explore

```
# 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)
```

```
##
## Call:
## lm(formula = y3 ~ x3)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -25.59 -11.19   2.92   8.65  39.16
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5335     7.6813   -2.28    0.029 *
## x3            0.1657     0.0285    5.80  1.7e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15 on 33 degrees of freedom
## Multiple R-squared:  0.505,  Adjusted R-squared:  0.49
## F-statistic: 33.7 on 1 and 33 DF,  p-value: 1.72e-06
```

```
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", level= 0.90, se = TRU

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



10.2.2 Diagnostics fro assessing the regression line

10.2.2.1 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.

10.2.2.2 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.

10.3 Multiple Linear Regression

10.3.1 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.

10.4 Linear regression in Software Effort estimation

Fitting a linear model to log-log - the predictive power equation is $y = e^{b_0+b_1 log(x)}$, ignoring the bias corrections - First, we are fitting the model to the whole dataset. But it is not the right way to do it, because of overfitting.

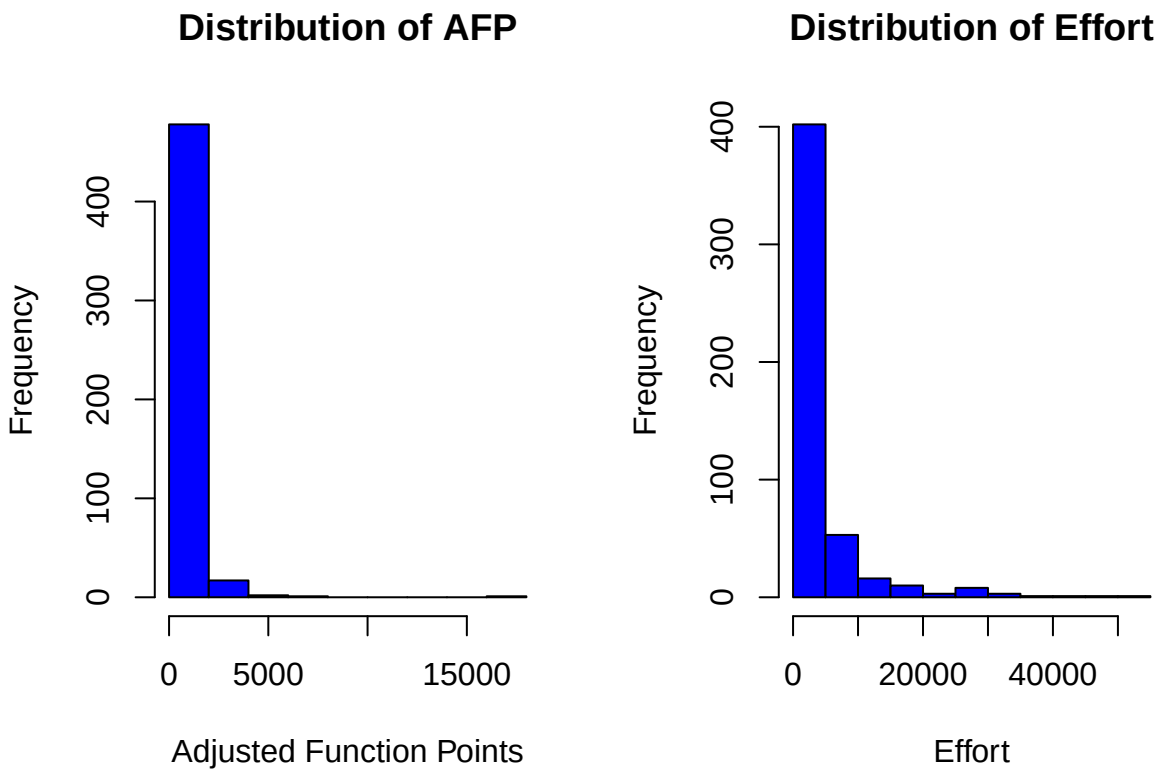
```
library(foreign)
china <- read.arff("./datasets/effortEstimation/china.arff")
china_size <- china$AFP
summary(china_size)
```

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

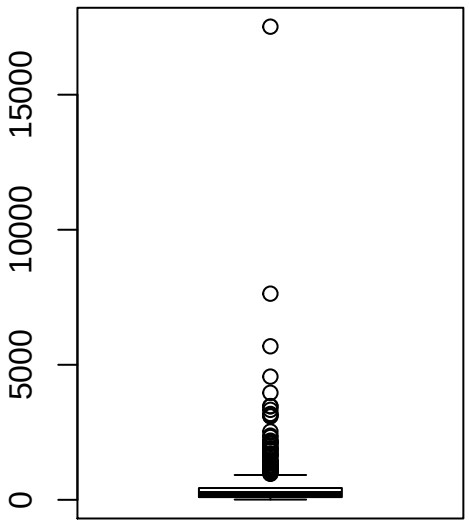
```
china_effort <- china$Effort
summary(china_effort)
```

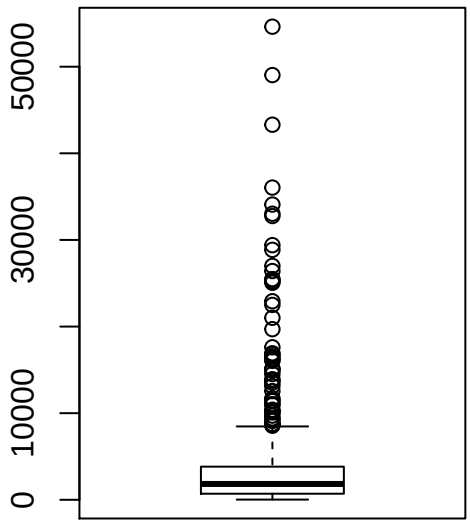
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      26     704    1829    3921    3826   54620
```

```
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")
```

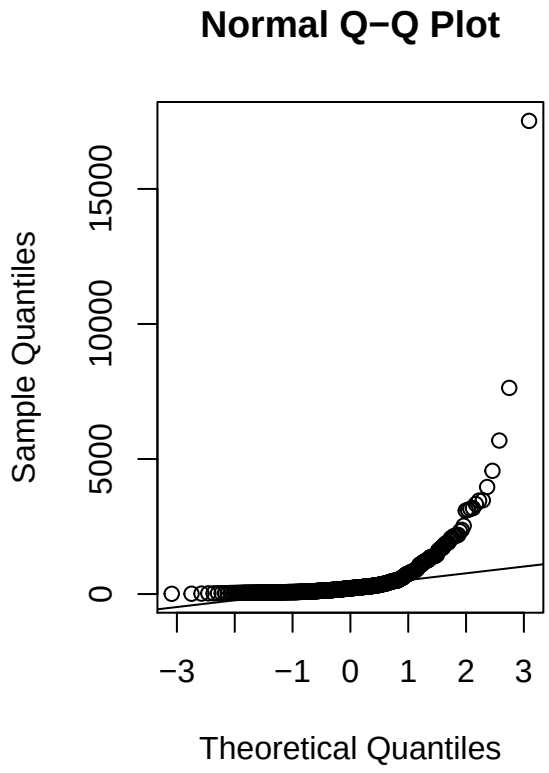


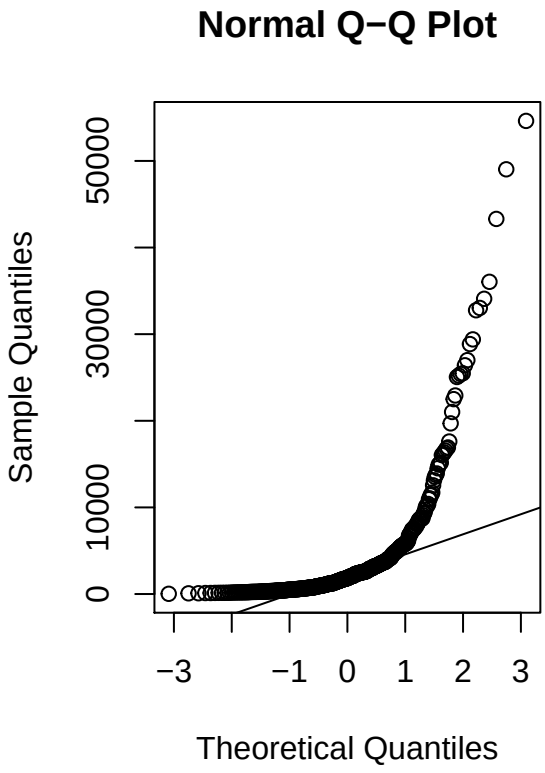

```
boxplot(china_size)
boxplot(china_effort)
```



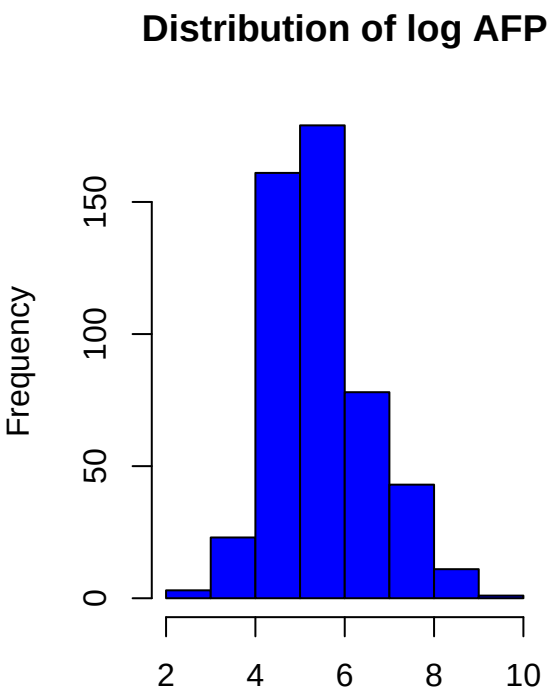


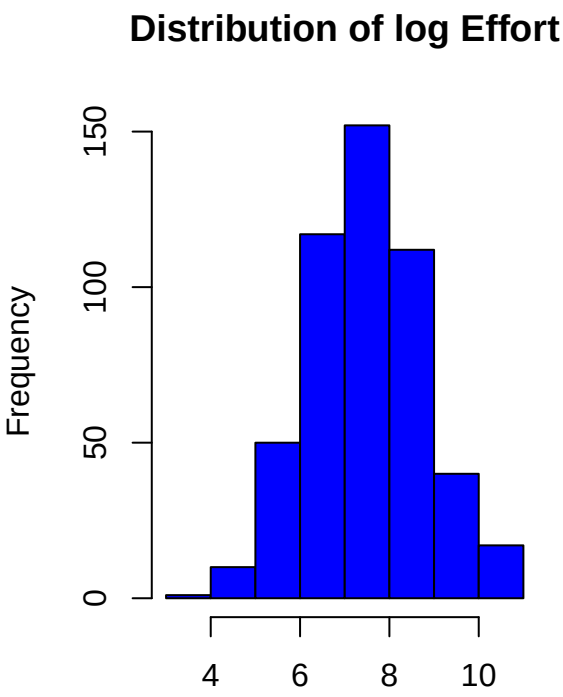

```
qqnorm(china_size)
qqline(china_size)
qqnorm(china_effort)
qqline(china_effort)
```

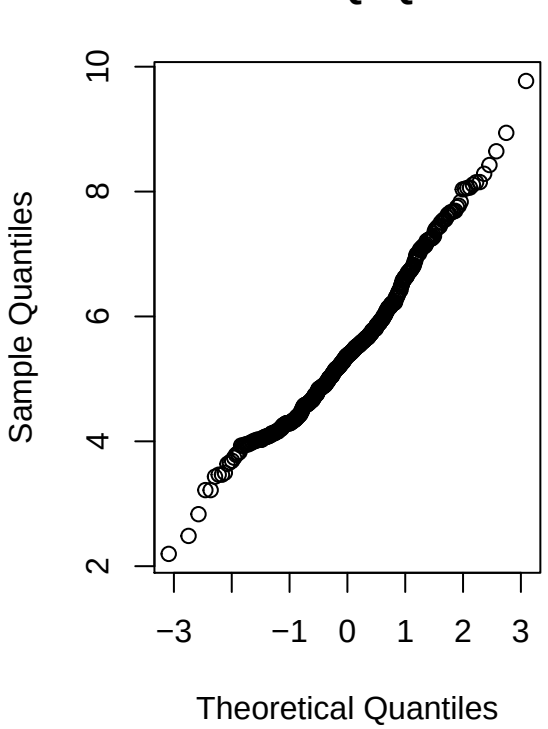


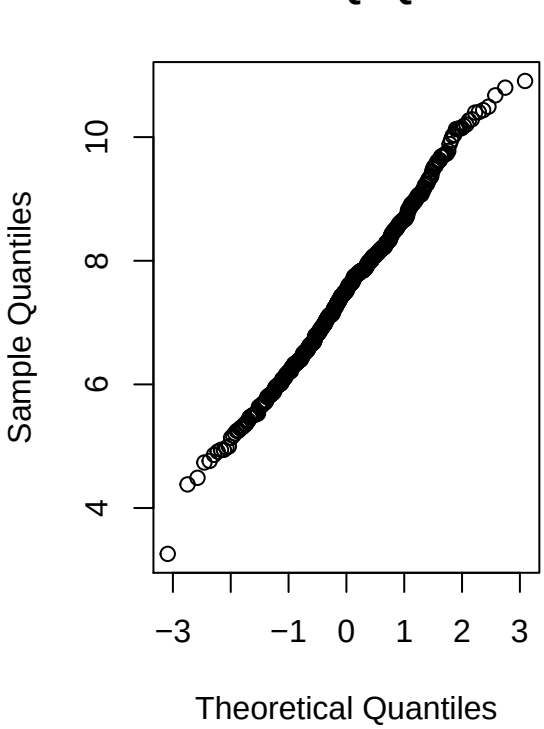


Applying the `log` function

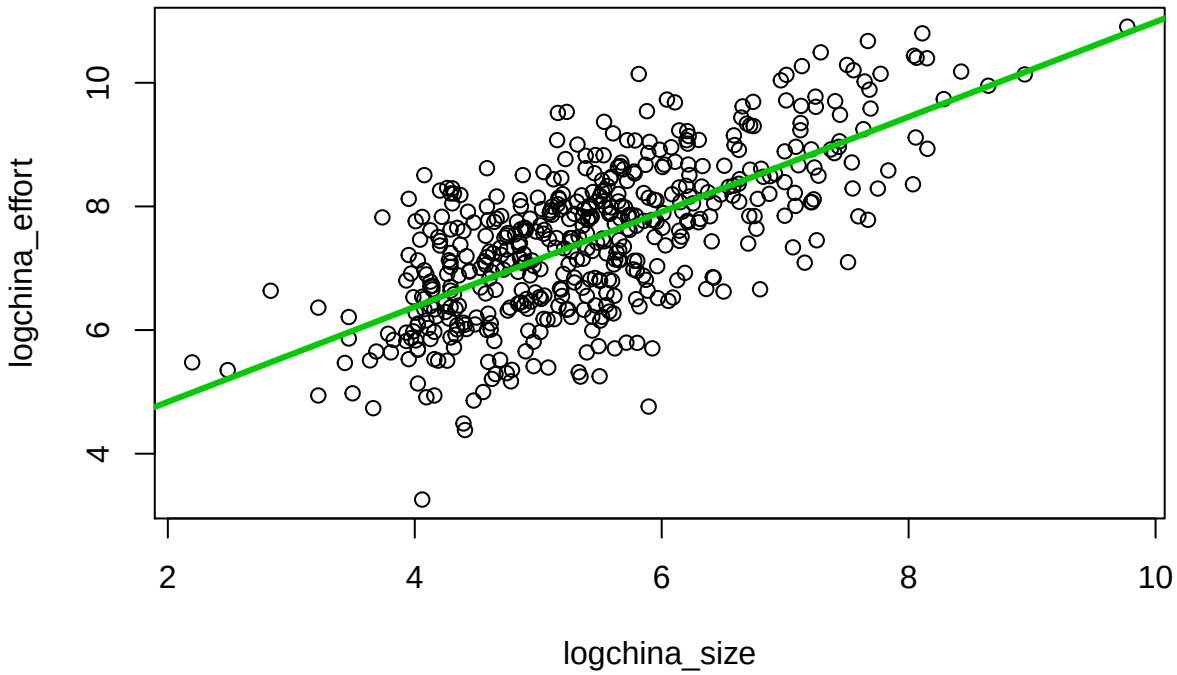




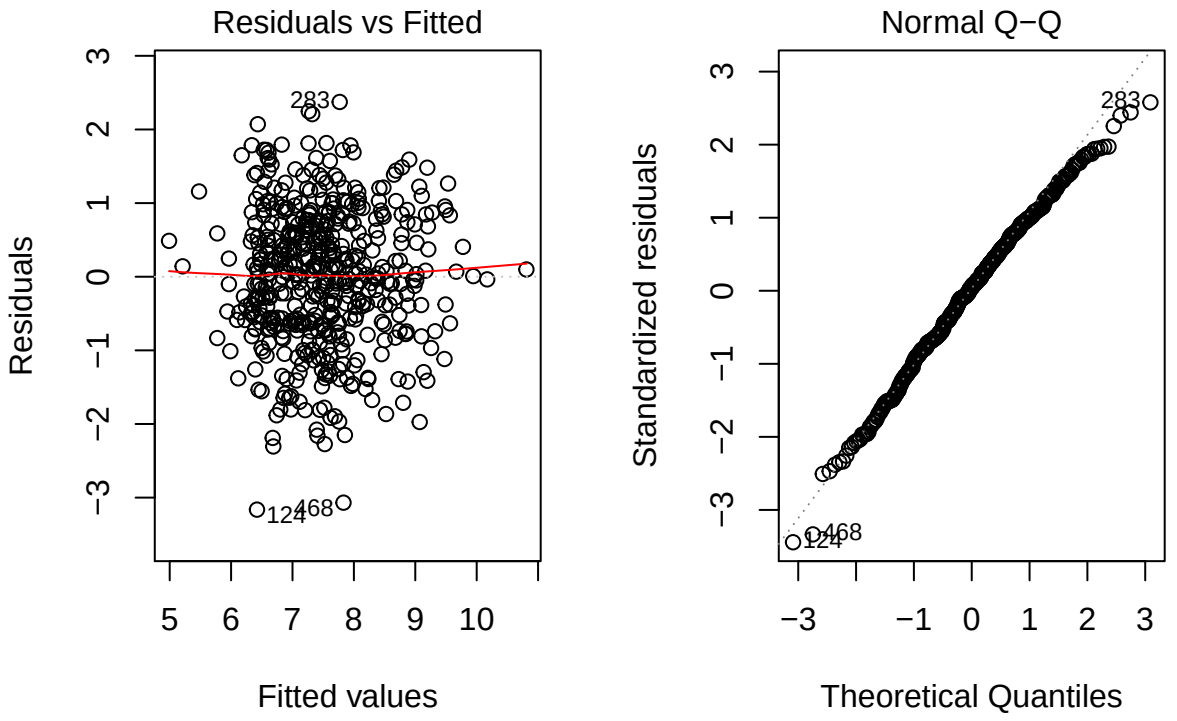


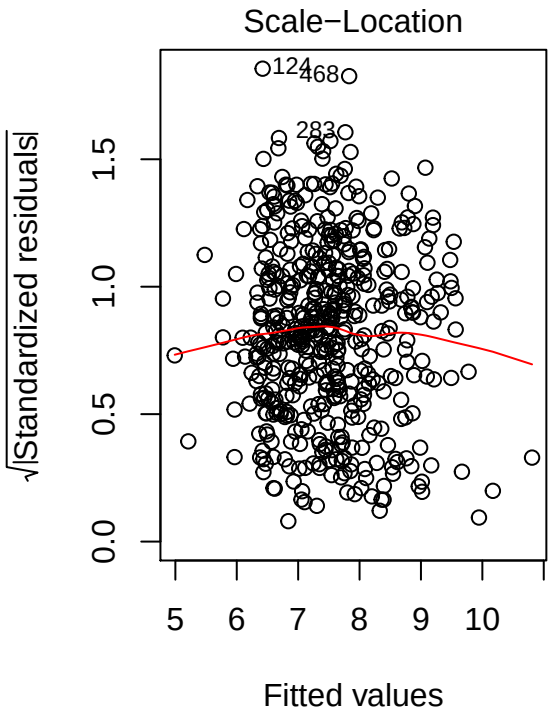


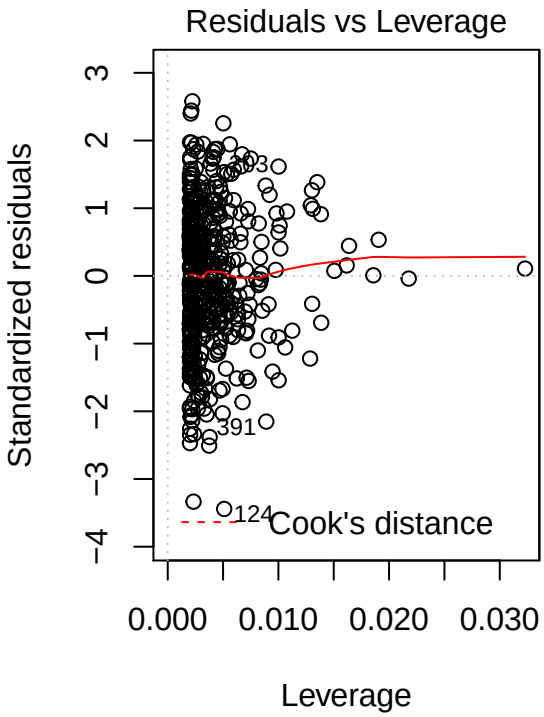

```
linmodel_logchina <- lm(logchina_effort ~ logchina_size)
par(mfrow=c(1,1))
plot(logchina_size, logchina_effort)
abline(linmodel_logchina, lwd=3, col=3)
```




```
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
```

10.5 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 VII

Unsupervised Models

Chapter 11

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}

11.1 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)
```

11.1.1 k-Means

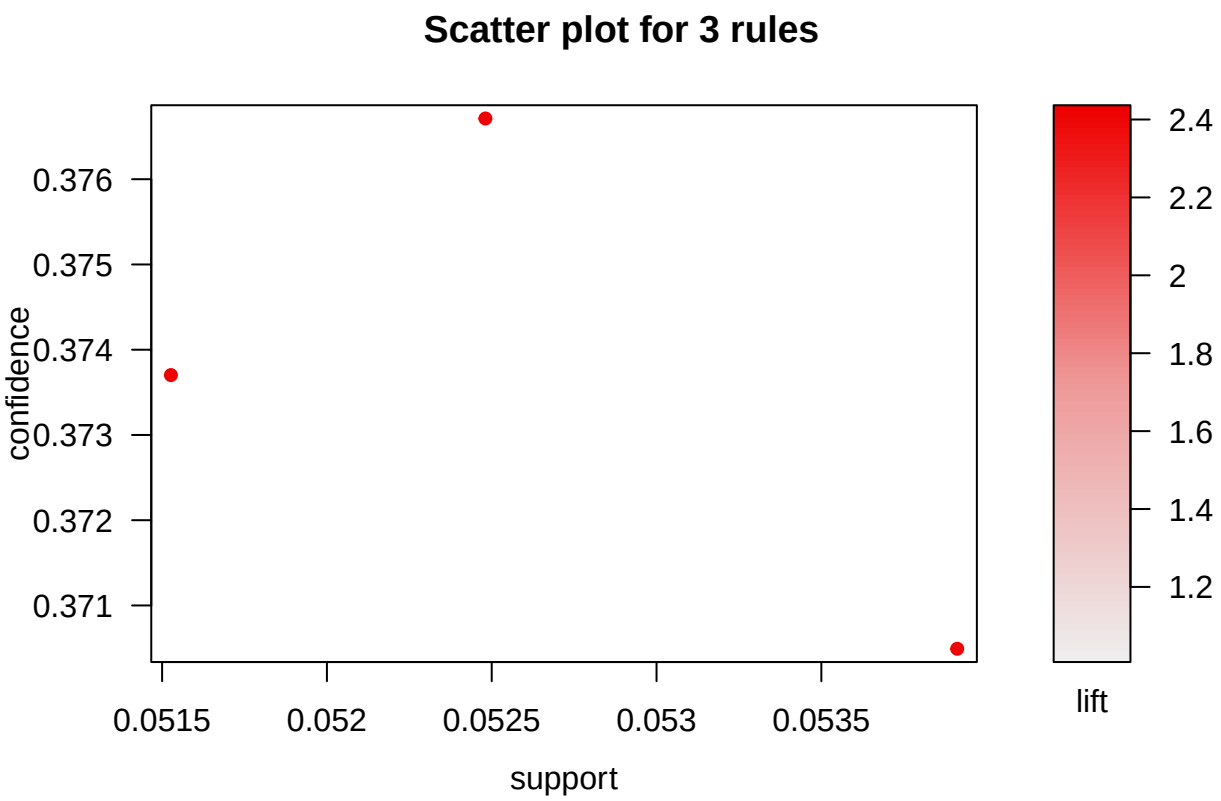
```
library(reshape, quietly=TRUE)
library(graphics)
kc1kmeans <- kmeans(sapply(na.omit(kc1.train), rescaler, "range"), 10)
#plot(kc1kmeans, col = kc1kmeans$cluster)
#points(kc1kmeans$centers, col = 1:5, pch = 8)
```

11.2 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 lift count
## [1] {HALSTEAD_CONTENT=[38.6,77.2),
##      HALSTEAD_LEVEL=[0,0.4)}       => {Defective=Y}  0.0539      0.370 2.39   113
## [2] {LOC_CODE_AND_COMMENT=[0,2.4),
##      HALSTEAD_CONTENT=[38.6,77.2)} => {Defective=Y}  0.0525      0.377 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 2.41   108
```

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


Scatter plot for 3 rules
confidence
0.376
0.375
0.374
0.373
0.372
0.371
0.0515
0.052
0.0525
0.053
0.0535
support
2.4
2.2
2
1.8
1.6
1.4
1.2
lift

Part VIII

Evaluation

Chapter 12

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

12.1 Building and Validating a Model

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

Therefore, we should, at a minimun, 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.

12.1.1 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 aprox 45% of positive cases, the training and testing datasets should mantain 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)

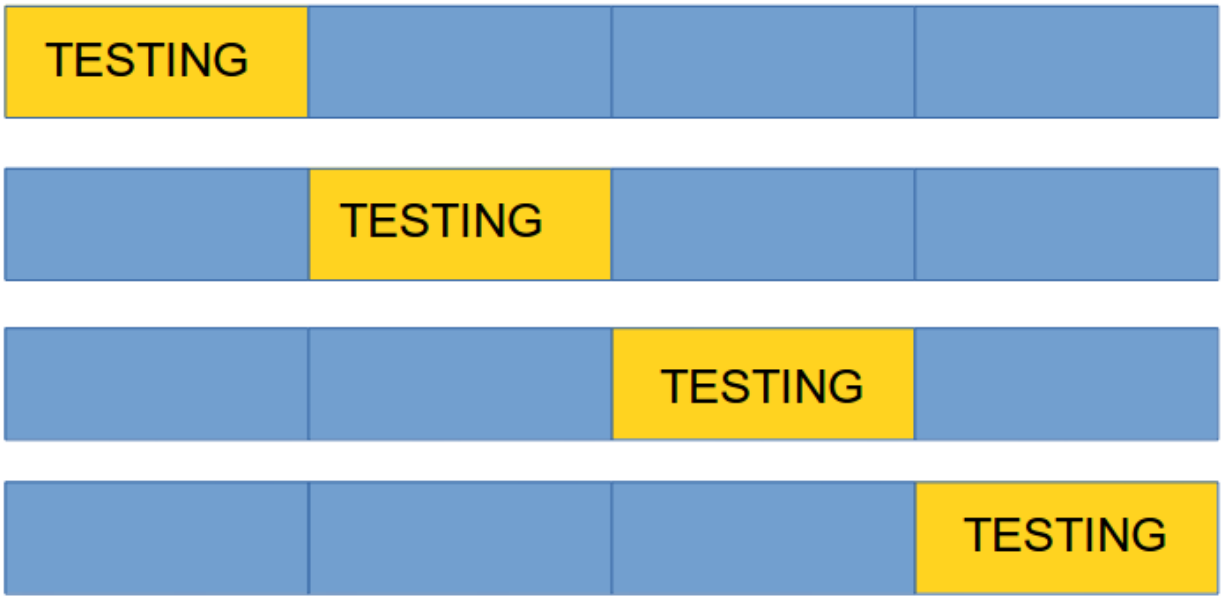
- Usually, part of the data points are used for building the model and the remaining points are used for validating the model. There are several approaches to this process.
- *Validation Set approach*: it is the simplest method. It consists of randomly dividing the available set of oservations into two 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.

12.1.2 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 $k-1$

Validation

Figure 12.1: Hold out validation



K-fold

Figure 12.2: k-fold

folds. This procedure is repeated k times. If k is equal to n we are in the previous method.

- 1st step: split dataset ($\mathcal{D}$) into k subsets of approximatelly equal size $C_1, \ldots, C_k$
- 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

- *Leave-One-Out Cross-Validation*: 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.

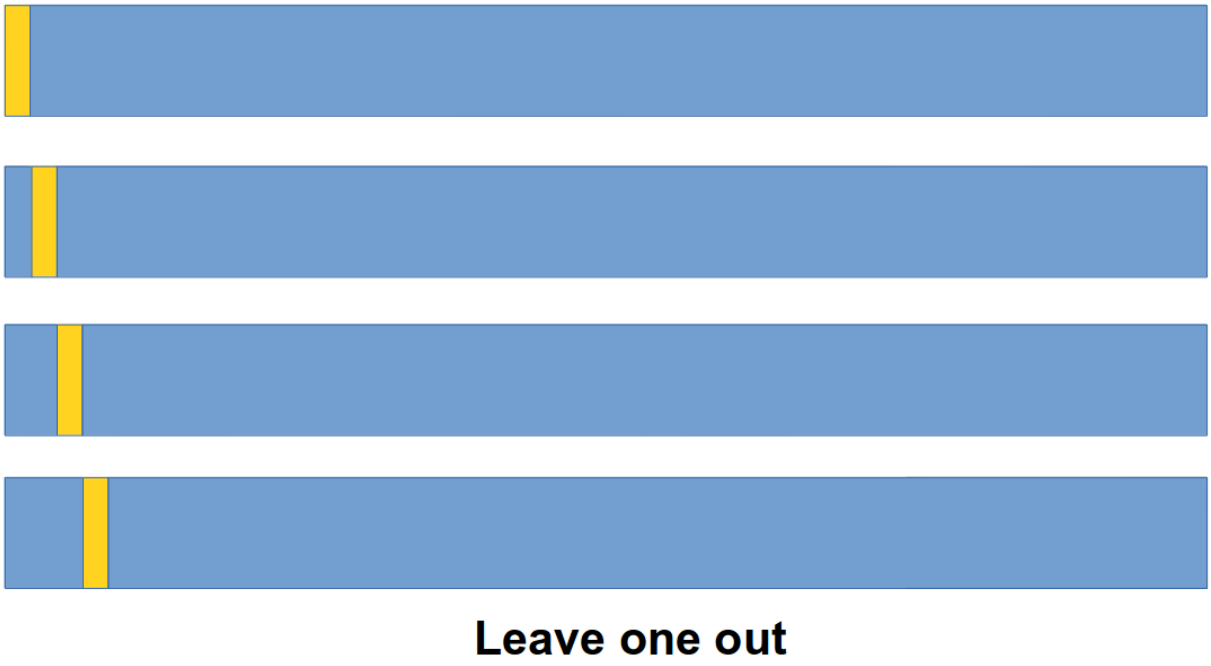


Figure 12.3: Leave One Out

12.2 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:

	$ActPos$	$ActNeg$
$PredPos$	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* ($TP_r = recall = r = TP/TP + FN$) 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 tradeoff 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*:

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

- 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.

12.2.1 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):

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

12.3 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.

12.4 Graphical Evaluation

12.4.1 Receiver Operating Characteristic (ROC)

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

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: $AUC = \frac{1+TP_r-FP_r}{2}$

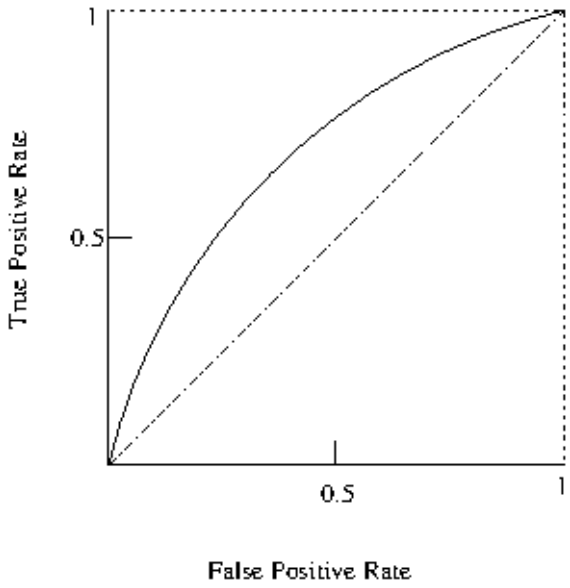


Figure 12.4: Receiver Operating Characteristic

12.4.2 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.

12.5 Numeric Prediction Evaluation

RSME

Mean Square Error = $MSE = \frac{(p_1-a_1)^2+...+(p_n-a_n)^2}{n}$

$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$

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

Mean-absolute error MAE

$\frac{|p_1-a_1|+...+|p_n-a_n|}{n}$

Relative absolute error:

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

Root relative-squared error:

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

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

Relative-squared error $\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{(a_1-\hat{a})^2+\ldots+(a_n-\hat{a})^2}$ ($\hat{a}$ is the mean value over the training data)

Relative Absolut Error

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 13

Measures of Evaluation in Software Engineering

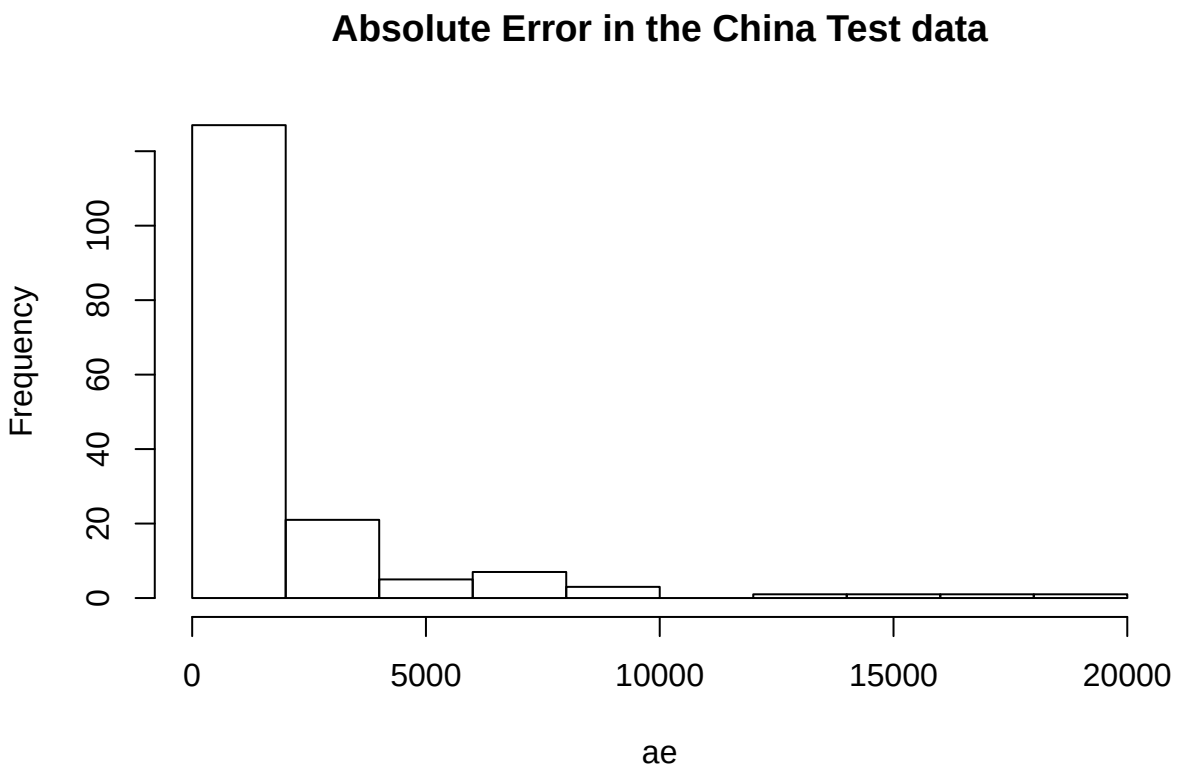
There are several measures typically used in software engieering. In particular for effort estimation, the following metrics are extensively used in addition or instead of statistical measures.

- *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 insted 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&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.

13.1 Evaluation of the model in the Testing data

```
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))

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
```

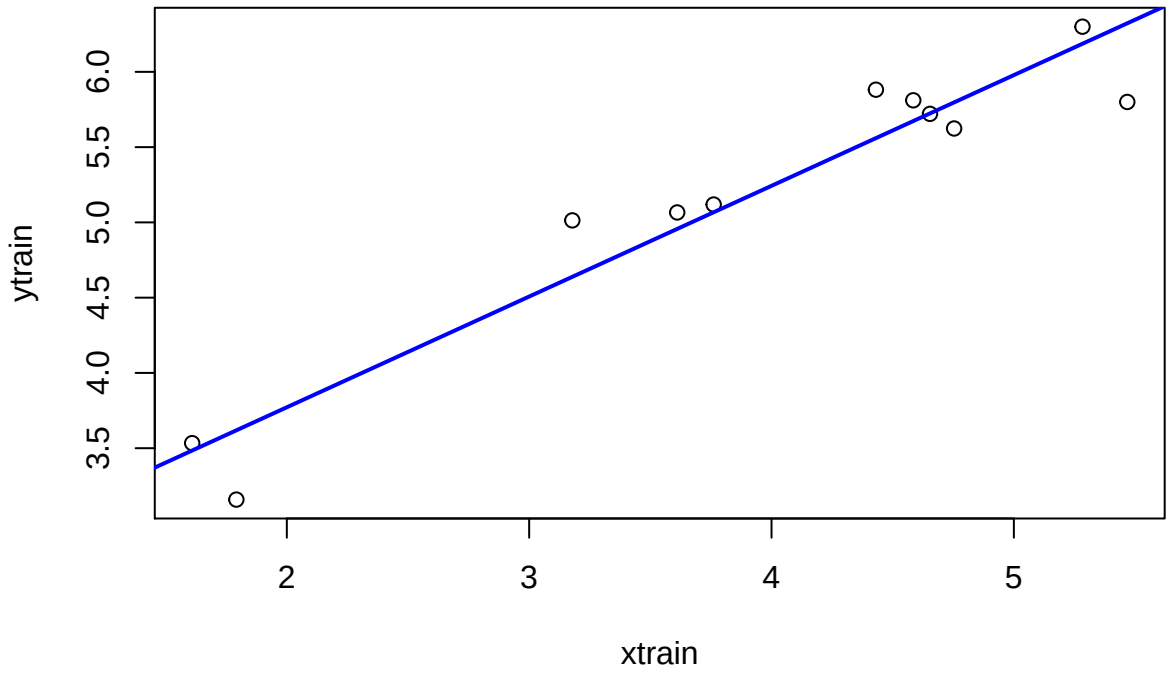
```
level_pred <- 0.25 #below and above (both)
lowpred <- actualEffort*(1-level_pred)
uppred <-  actualEffort*(1+level_pred)
pred  <-  predEffort <= uppred & predEffort >= lowpred  #pred is a vector with logical values
Lpred <- sum(pred)/length(pred)
Lpred
```

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

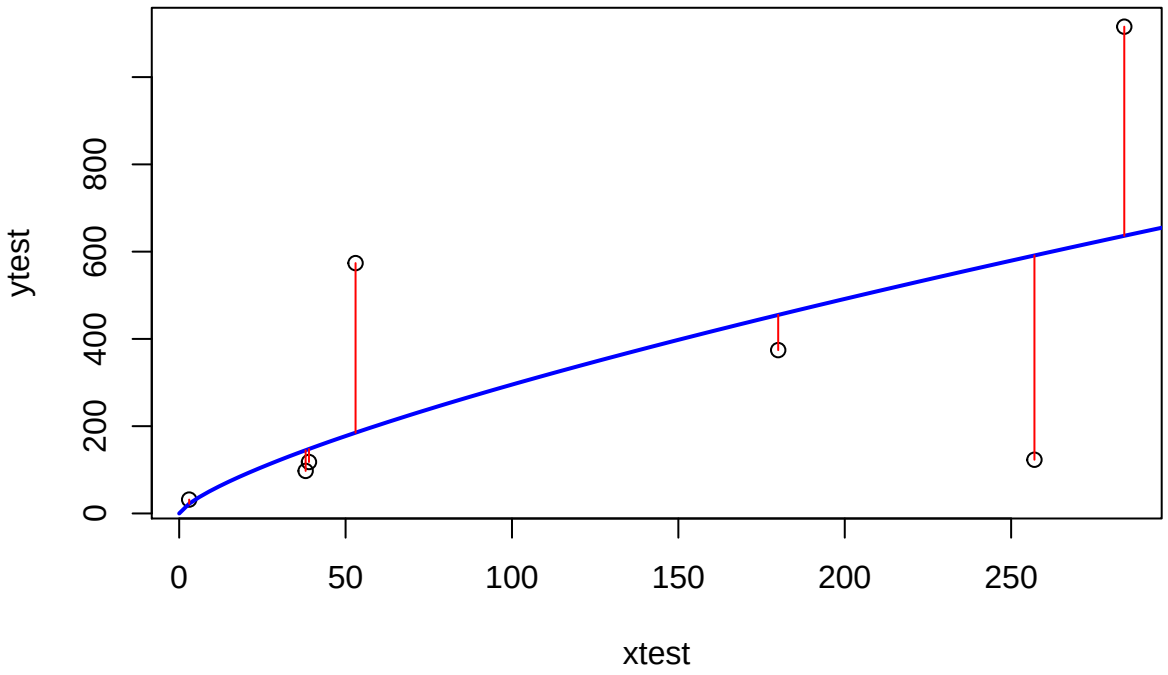
13.2 Building a Linear Model on the Telecom1 dataset

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

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

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

xtest <- telecom1_test$size
ytest <- telecom1_test$effort
pre_tel1 <- exp(b0_tel1+b1_tel1*log(xtest))
# 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")
```



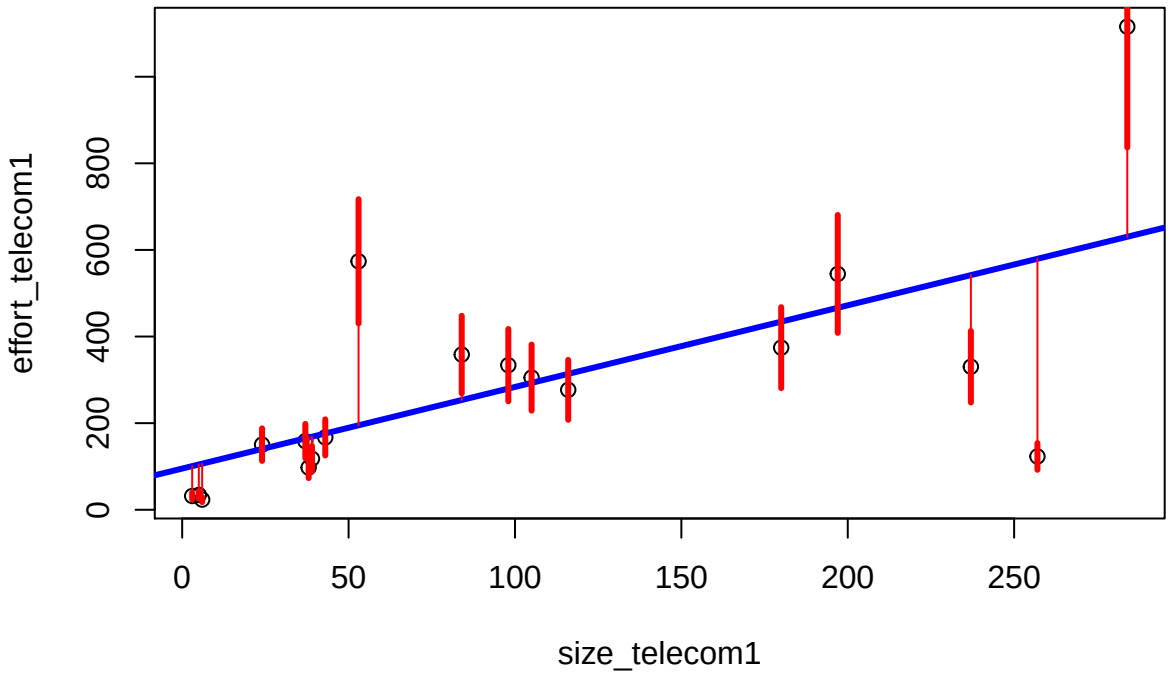
13.3 Building a Linear Model on the Telecom1 dataset with all observations

- Just to visualize results

```
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 with logical values
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
```

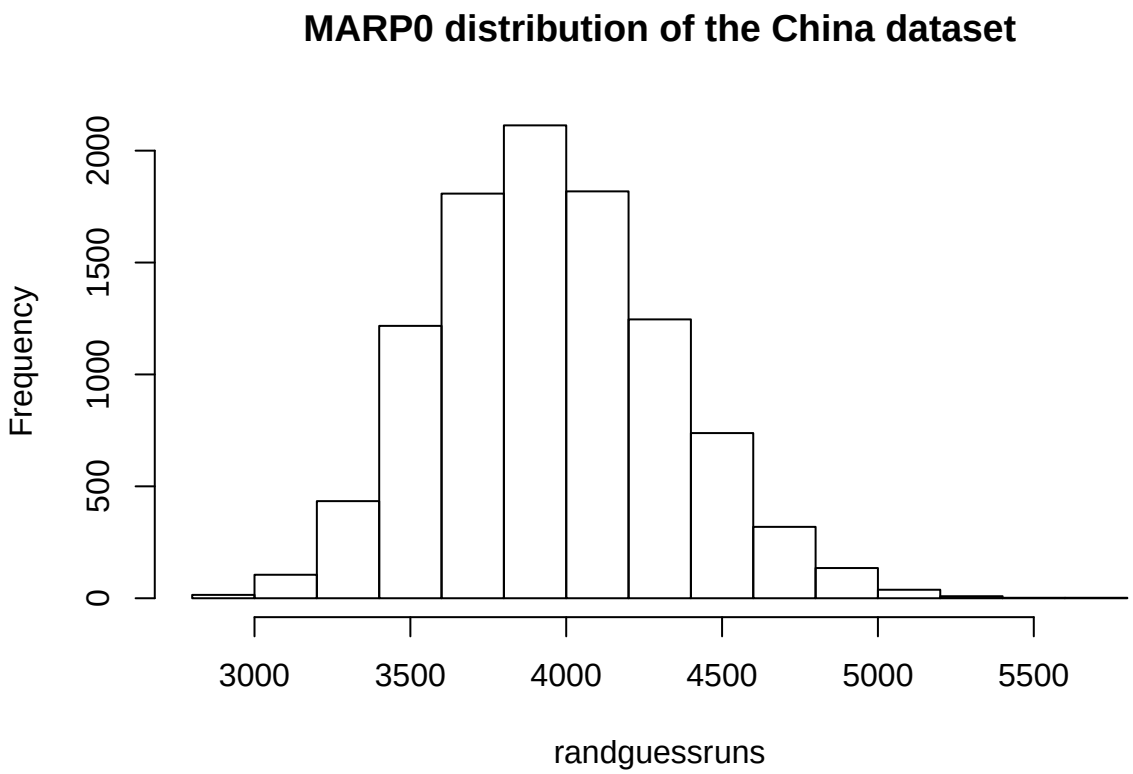
13.4 Standardised Accuracy. MARP0. ChinaTest

- 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] 3956
```

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




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

```
## [1] 52.8
```

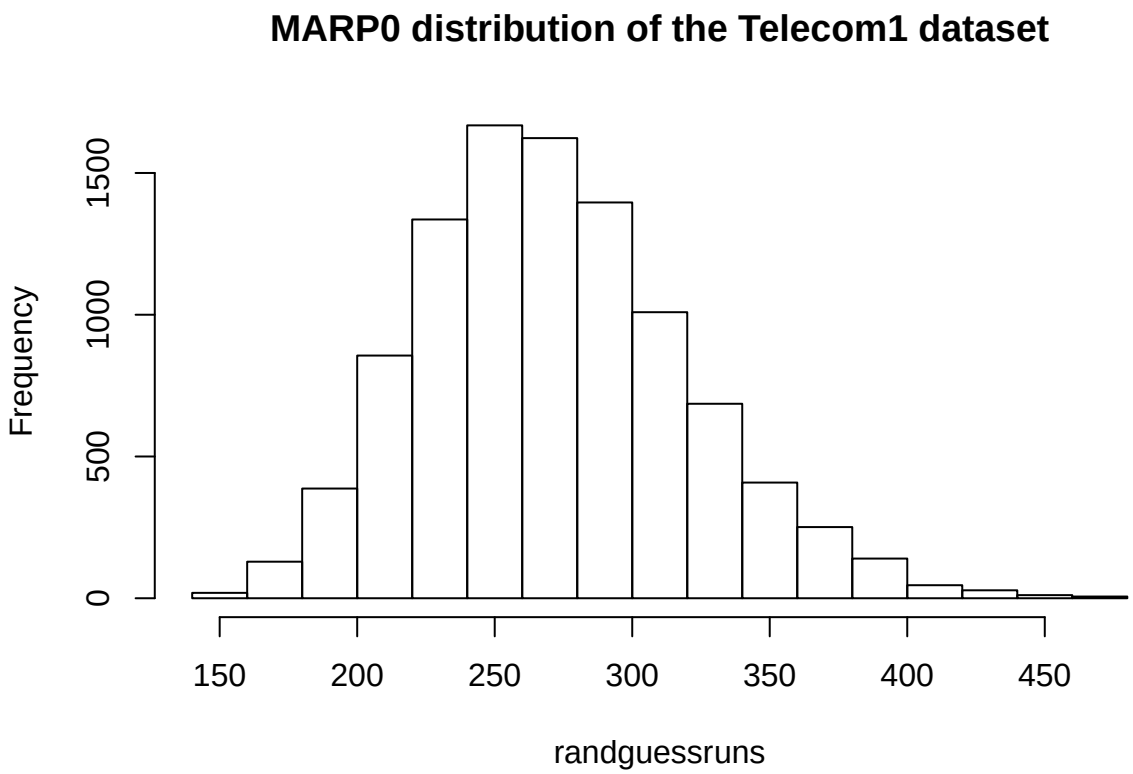
13.5 Standardised Accuracy. MARP0. Telecom1

- Computing $MARP_0$

```
telecom1 <- read.table("./datasets/effortEstimation/Telecom1.csv", sep=",",header=TRUE, stringsAsFactors
#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 guessingt
  randguessruns[i] <- mean(abs(estimEffTelecom1-actualEffTelecom1))
  }
marp0telecom1 <- mean(randguessruns)
marp0telecom1
```

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

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



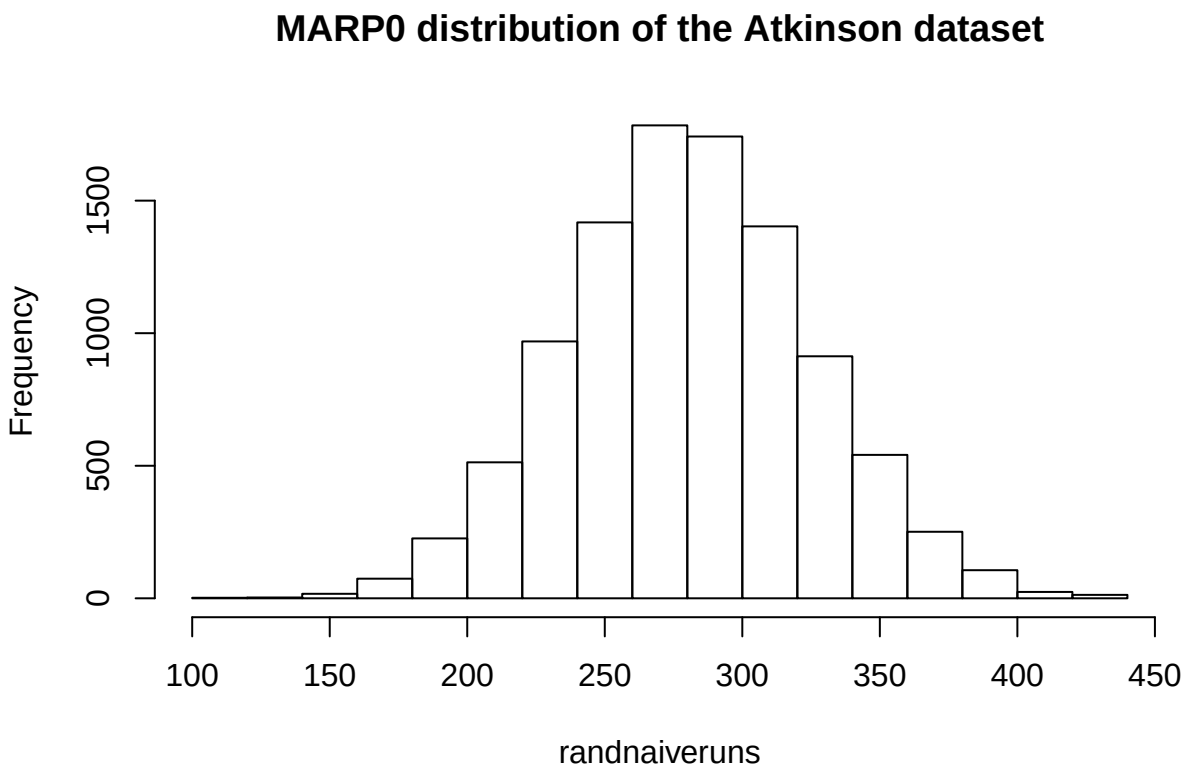

```
saTelecom1 <- (1- mar_tel1/marp0telecom1)*100
saTelecom1
```

```
## [1] 54
```

13.5.1 MARP0 in the Atkinson dataset

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

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



13.6 Exact MARP0

Chapter 14

WBL simple R code to calculate Shepperd and MacDonell's marp0 exactly

```
#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
```

```
#### 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
```

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

```
library(boot)
```

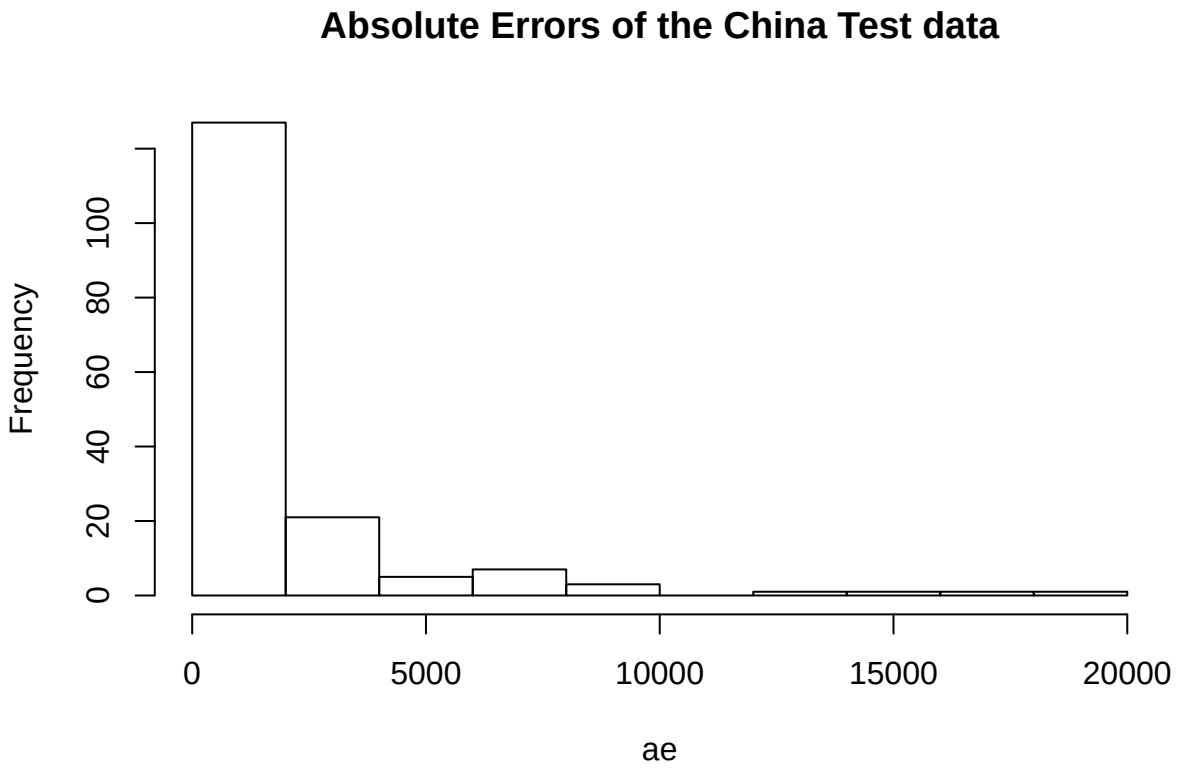
```
##
## Attaching package: 'boot'
```

```
## 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%   (1412, 2315 )   (1384, 2283 )
##
## Level     Percentile            BCa
## 95%   (1451, 2350 )   (1486, 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 calculating the mean, just
  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")$bca #following 10.9 of Ugarte et al.'s book
  conf_int[5] <- exp(conf_int[5]) # the boot was computed with log. Now take the measure back to its pr
  conf_int[4] <- exp(conf_int[4])
  return (conf_int)
}
# this is a boot object
b_gm <- boot_geom_mean(ae) #"ae" is the absolute error in the China Test data
print(paste0("Geometric Mean of the China Test data: ", round(exp(b_gm$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[5], digits=3)))
```

```
## [1] "Confidence Interval: 673.125 - 1009.427"
```

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

Part IX

Advanced Topics

Chapter 15

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 [21, 14, 13, 3, 1, 12, 5]:

- 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].

15.1 Instance Selection

15.2 Missing Data Imputation

Chapter 16

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 {r} testPred <- predict(plsFit, testing) postResample(testPred, testing$Defective) sensitivity(testPred, testing$Defective) confusionMatrix(testPred, testing$Defective)

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

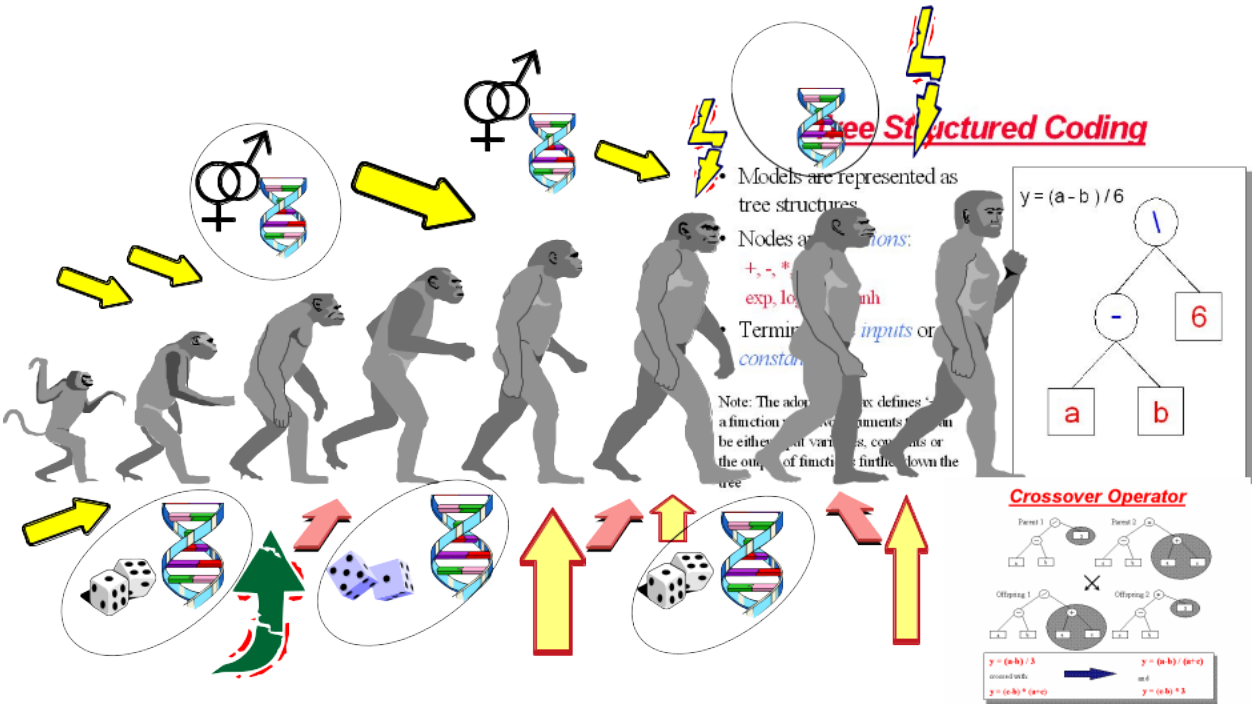
Chapter 17

Advanced Models

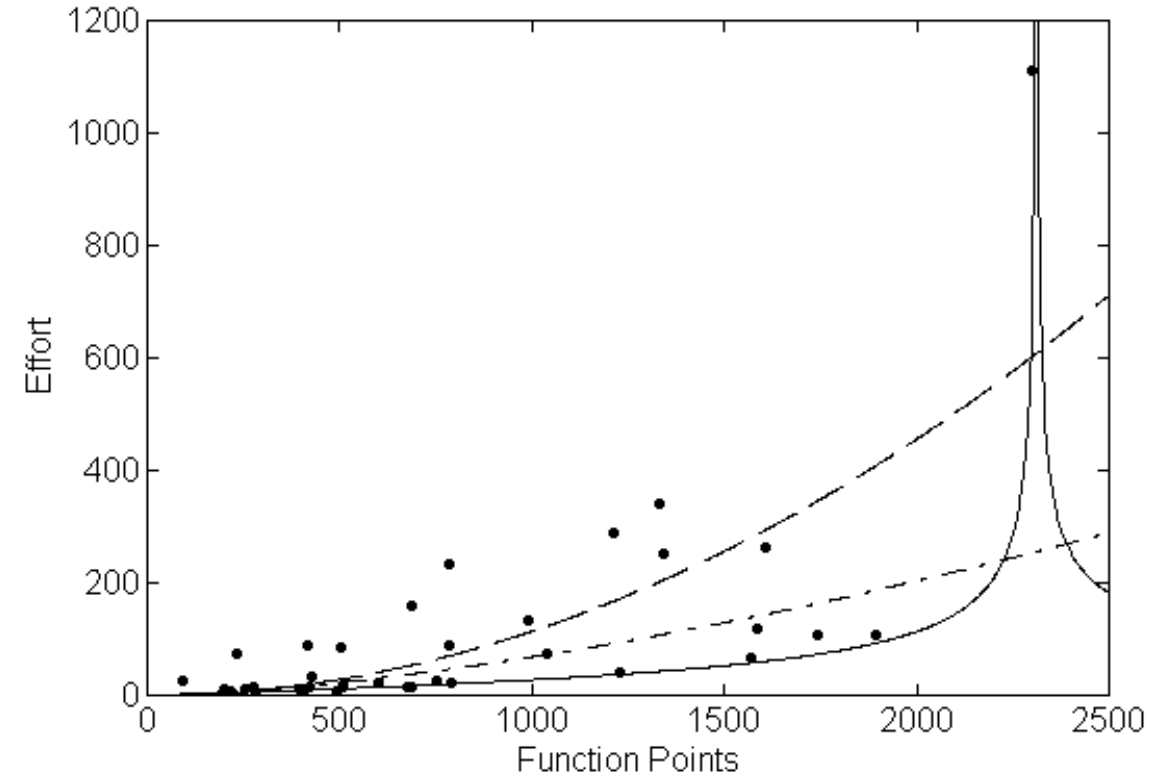
17.1 Genetic Programming for Symbolic Regression

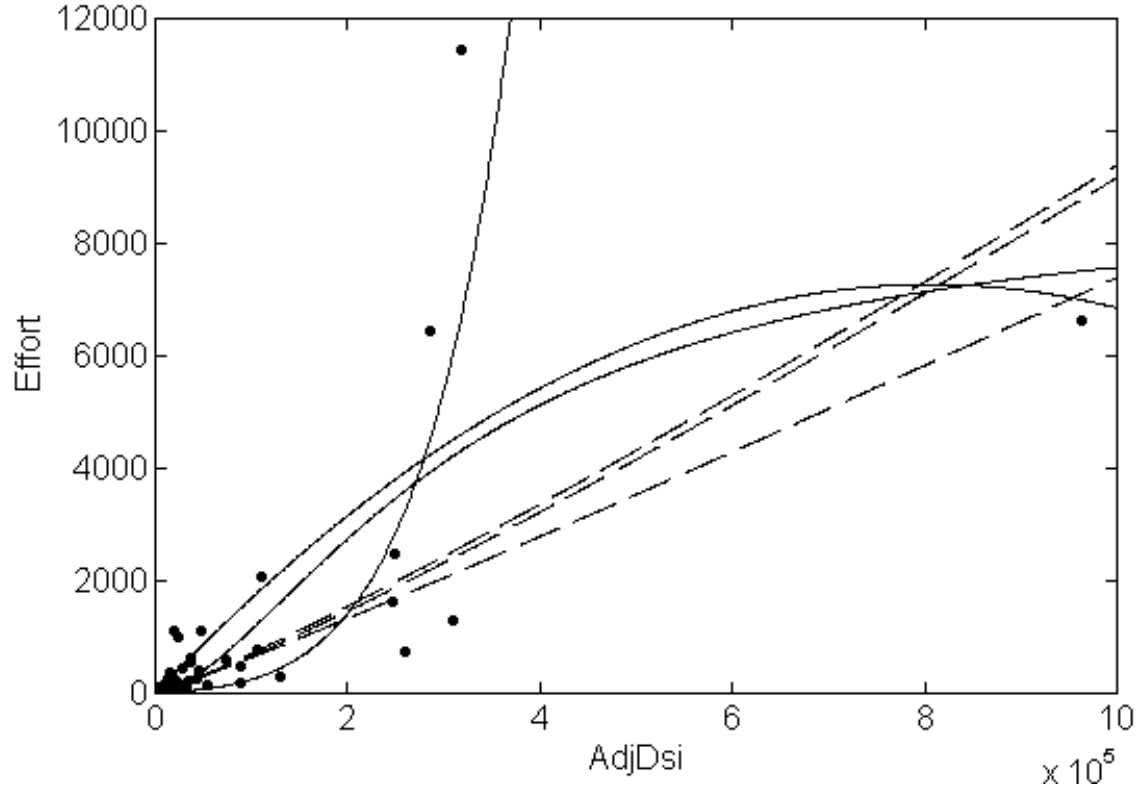
This technique is inspired by Darwin's evolution theory. + 1960s by I. Rechenberg in his work "Evolution strategies" + 1975 Genetic Algorithms (GAs) invented by J Holland and published in his book"Adaption in Natural and Artificial Systems" + 1992 J. Koza has used genetic algorithm to evolve programs to perform certain tasks. He called his method "genetic programming"

Other reference for GP: Langdon WB, Poli R (2001) Foundations of Genetic Programming. Springer.



- Depending on the function set used and the function to be minimised, GP can generate almost any type of curve





In R, we can use the "rgp" package

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Figure 17.1: Evolutionary Algorithms

17.2 Genetic Programming Example

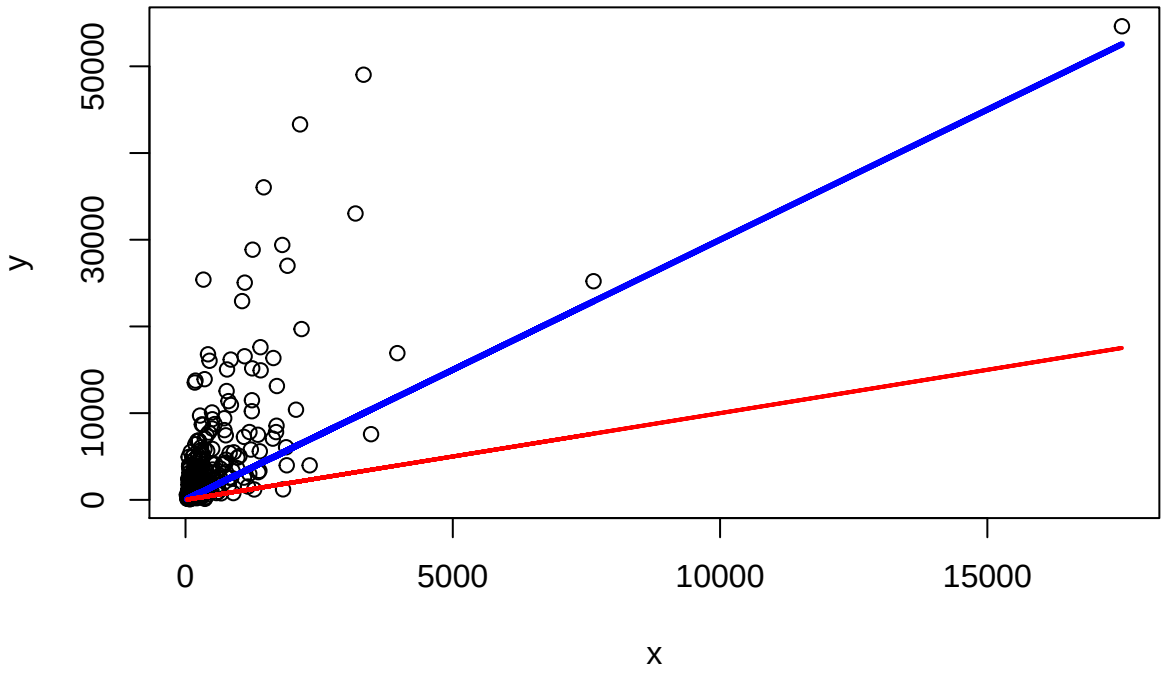
17.2.1 Load Data

```
library(foreign)

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

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

chinaTrain <- read.arff("./datasets/effortEstimation/china3AttSelectedAFPTrain.arff")
china_train_size <- chinaTrain$AFP
china_train_effort <- chinaTrain$Effort
chinaTest <- read.arff("./datasets/effortEstimation/china3AttSelectedAFPTest.arff")
china_size_test <- chinaTest$AFP
actualEffort <- chinaTest$Effort
```

17.2.2 Genetic Programming for Symbolic Regression: China dataset.

```
library("rgp")
options(digits = 5)
stepsGenerations <- 100
initialPopulation <- 100
Steps <- c(10)
y <- china_train_effort   #
x <- china_train_size  #
```

```
data2 <- data.frame(y, x)  # create a data frame with effort, size
# newFuncSet <- mathFunctionSet
# alternatives to mathFunctionSet
# newFuncSet <- expLogFunctionSet # sqrt", "exp", and "ln"
# newFuncSet <- trigonometricFunctionSet
# newFuncSet <- arithmeticFunctionSet
newFuncSet <- functionSet("+","-","*", "/","sqrt", "log", "exp") # ,, )

gpresult <- symbolicRegression(y ~ x,
                               data=data2, functionSet=newFuncSet,
                               populationSize=initialPopulation,
                               stopCondition=makeStepsStopCondition(stepsGenerations))

bf <- gpresult$population[[which.min(sapply(gpresult$population, gpresult$fitnessFunction))]]
wf <- gpresult$population[[which.max(sapply(gpresult$population, gpresult$fitnessFunction))]]

bf1 <- gpresult$population[[which.min((gpresult$fitnessValues))]]
plot(x,y)
lines(x, bf(x), type = "l", col="blue", lwd=3)
lines(x,wf(x), type = "l", col="red", lwd=2)
```




```
x_test <- china_size_test
estim_by_gp <- bf(x_test)
ae_gp <- abs(actualEffort - estim_by_gp)
mean(ae_gp)
```

```
## [1] 2135.4
```

17.2.3 Genetic Programming for Symbolic Regression. Telecom1 dataset.

- For illustration purposes only. We use all data points.

```
# y <- effort_telecom1   # all data points
# x <- size_telecom1   #
#
# data2 <- data.frame(y, x)  # create a data frame with effort, size
# # newFuncSet <- mathFunctionSet
# # alternatives to mathFunctionSet
# newFuncSet <- expLogFunctionSet # sqrt", "exp", and "ln"
# # newFuncSet <- trigonometricFunctionSet
# # newFuncSet <- arithmeticFunctionSet
# # newFuncSet <- functionSet("+","-","*", "/","sqrt", "log", "exp") # ,, )
#
# gpresult <- symbolicRegression(y ~ x,
#                                data=data2, functionSet=newFuncSet,
#                                populationSize=initialPopulation,
#                                stopCondition=makeStepsStopCondition(stepsGenerations))
#
# bf <- gpresult$population[[which.min(sapply(gpresult$population, gpresult$fitnessFunction))]]
# wf <- gpresult$population[[which.max(sapply(gpresult$population, gpresult$fitnessFunction))]]
#
# bf1 <- gpresult$population[[which.min((gpresult$fitnessValues))]]
# plot(x,y)
# lines(x, bf(x), type = "l", col="blue", lwd=3)
# lines(x,wf(x), type = "l", col="red", lwd=2)
```

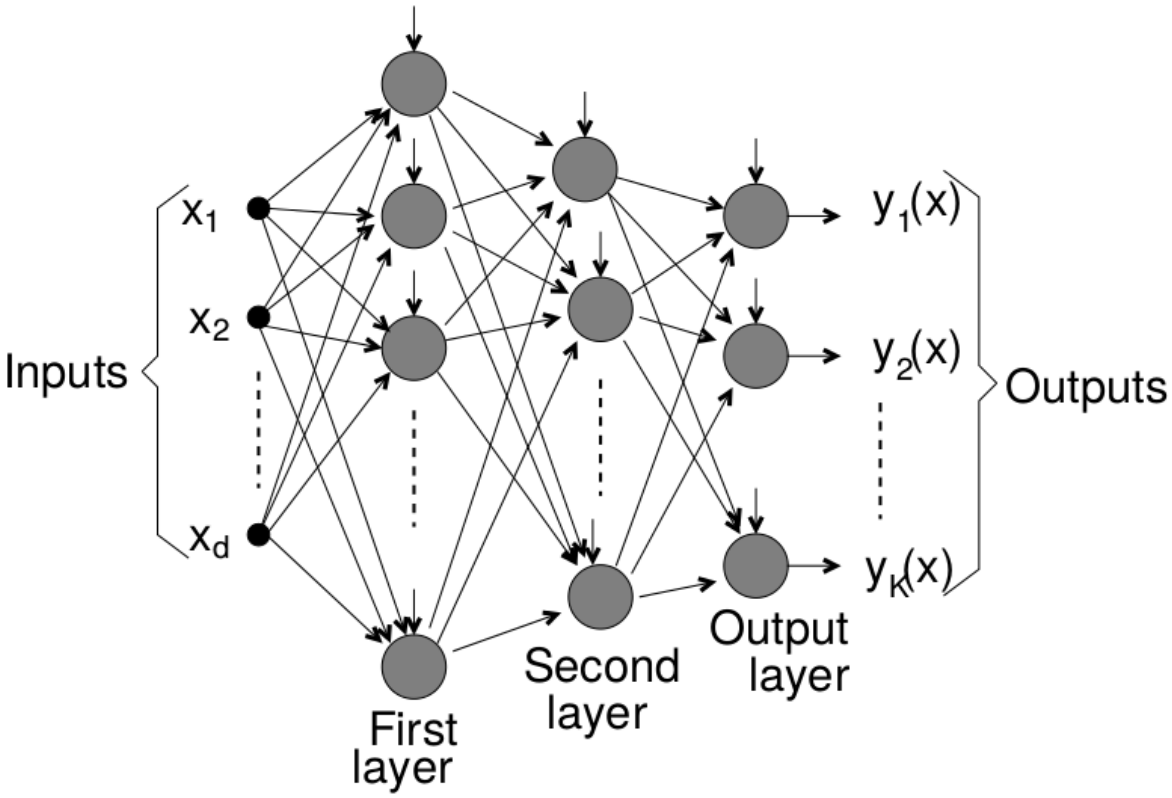
17.3 Neural Networks

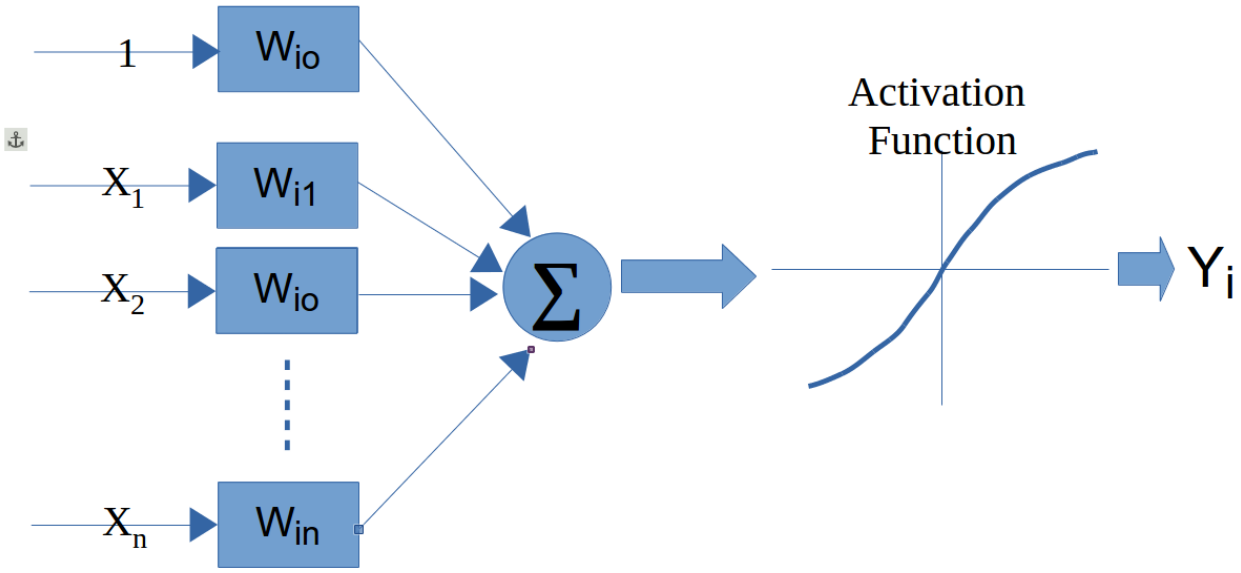
A neural network (NN) simulates some of the learning functions of the human brain.

It can recognize patterns and “learn” . Through the use of a trial and error method the system “learns” to become an “expert” in the field.

A NN is composed of a set of nodes (units, neurons, processing elements) + Each node has input and output + Each node performs a simple computation by its node function

Weighted connections between nodes + Connectivity gives the structure/architecture of the net + What can be computed by a NN is primarily determined by the connections and their weights





There are several packages in R to work with NNs + neuralnet + nnet + RSNNS

TO BE FIXED!!!: The following is an example with the neuralnet package (TO DO, denormalize!). Neural nets need scaling of variables to work properly.

```
library(foreign)
library(neuralnet)

chinaTrain <- read.arff("datasets/effortEstimation/china3AttSelectedAFPTrain.arff")
```

```
afpsize <- chinaTrain$AFP
effort_china <- chinaTrain$Effort

chinaTest <- read.arff("datasets/effortEstimation/china3AttSelectedAFPTest.arff")
AFPTest <- chinaTest$AFP
actualEffort <- chinaTest$Effort

trainingdata <- cbind(afpsize,effort_china)
colnames(trainingdata) <- c("Input","Output")

testingdata <- cbind(afpsize,effort_china)
colnames(trainingdata) <- c("Input","Output")

#Normalize data
norm.fun = function(x){(x - min(x))/(max(x) - min(x))}
data.norm = apply(trainingdata, 2, norm.fun)
#data.norm

testdata.norm <- apply(trainingdata, 2, norm.fun)
#testdata.norm


#Train the neural network
#Going to have 10 hidden layers
#Threshold is a numeric value specifying the threshold for the partial
#derivatives of the error function as stopping criteria.
#net_eff <- neuralnet(Output~Input,trainingdata, hidden=5, threshold=0.25)
net_eff <- neuralnet(Output~Input, data.norm, hidden=10, threshold=0.01)

# Print the network
# print(net_eff)

#Plot the neural network
plot(net_eff)

#Test the neural network on some training data
#testdata.norm<-data.frame((testdata[,1] - min(data[, 'displ']))/(max(data[, 'displ'])-min(data[, 'disp

# Run them through the neural network
# net.results <- compute(net_eff, testdata.norm[,2])


#net.results <- compute(net_eff, dataTest.norm) # With normalized data

#Lets see what properties net.sqrt has
#ls(net.results)
#Lets see the results
#print(net.results$net.result)

#Lets display a better version of the results
#cleanoutput <- cbind(testdata.norm[,2],actualEffort,
#                     as.data.frame(net.results$net.result))
#colnames(cleanoutput) <- c("Input","Expected Output","Neural Net Output")
```

```
#print(cleanoutput)
```

17.4 Support Vector Machines

SVM

17.5 Ensembles

Ensembles or meta-learners combine multiple models to obtain better predictions i.e., this technique consists in combining single classifiers (sometimes are also called weak classifiers).

A problem with ensembles is that their models are difficult to interpret (they behave as blackboxes) in comparison to decision trees or rules which provide an explanation of their decision making process.

They are typically classified as Bagging, Boosting and Stacking (Stacked generalization).

17.5.1 Bagging

Bagging (also known as Bootstrap aggregating) is an ensemble technique in which a base learner is applied to multiple equal size datasets created from the original data using bootstraping. Predictions are based on voting of the individual predictions. An advantage of bagging is that it does not require any modification to the learning algorithm and takes advantage of the instability of the base classifier to create diversity among individual ensembles so that individual members of the ensemble perform well in different regions of the data. Bagging does not perform well with classifiers if their output is robust to perturbation of the data such as nearest-neighbour (NN) classifiers.

17.5.2 Boosting

Boosting techniques generate multiple models that complement each other inducing models that improve regions of the data where previous induced models preformed poorly. This is achieved by increasing the weights of instances wrongly classified, so new learners focus on those instances. Finally, classification is based on a weighted voted among all members of the ensemble.

In particular, AdaBoost.M1 [15] is a popular boosting algorithm for classification. The set of training examples is assigned an equal weight at the beginning and the weight of instances is either increased or decreased depending on whether the learner classified that instance incorrectly or not. The following iterations focus on those instances with higher weights. AdaBoost.M1 can be applied to any base learner.

17.5.3 Rotation Forests

Rotation Forests [40] combine randomly chosen subsets of attributes (random subspaces) and bagging approaches with principal components feature generation to construct an ensemble of decision trees. Principal Component Analysis is used as a feature selection technique combining subsets of attributes which are used with a bootstrapped subset of the training data by the base classifier.

17.5.4 Boosting in R

In R, there are three packages to deal with Boosting: gmb, ada and the mboost packages. An example of gbm using the caret package.

```
# load libraries
library(caret)
library(pROC)

##################################################
# model it
##################################################

# Get names of caret supported models (just a few - head)
head(names(getModelInfo()))
```

```
## [1] "ada"         "AdaBag"      "AdaBoost.M1" "adaboost"    "amdai"
## [6] "ANFIS"
```

```
# Show model info and find out what type of model it is
getModelInfo()$gbm$tags
```

```
## [1] "Tree-Based Model"           "Boosting"
## [3] "Ensemble Model"             "Implicit Feature Selection"
## [5] "Accepts Case Weights"
```

```
getModelInfo()$gbm$type
```

```
## [1] "Regression"     "Classification"
```

```
library(foreign)
library(caret)
library(pROC)

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

# Split data into training and test datasets
# TODO: Improve this with createDataPartition from Caret
set.seed(1234)
ind <- sample(2, nrow(kc1), replace = TRUE, prob = c(0.7, 0.3))
kc1.train <- kc1[ind==1, ]
kc1.test <- kc1[ind==2, ]


# create caret trainControl object to control the number of cross-validations performed
objControl <- trainControl(method='cv', number=3, returnResamp='none', summaryFunction = twoClassSummary


# run model
objModel <- train(Defective ~ .,
                  data = kc1.train,
                  method = 'gbm',
                  trControl = objControl,
                  metric = "ROC" #,
                  #preProc = c("center", "scale")
                  )
```

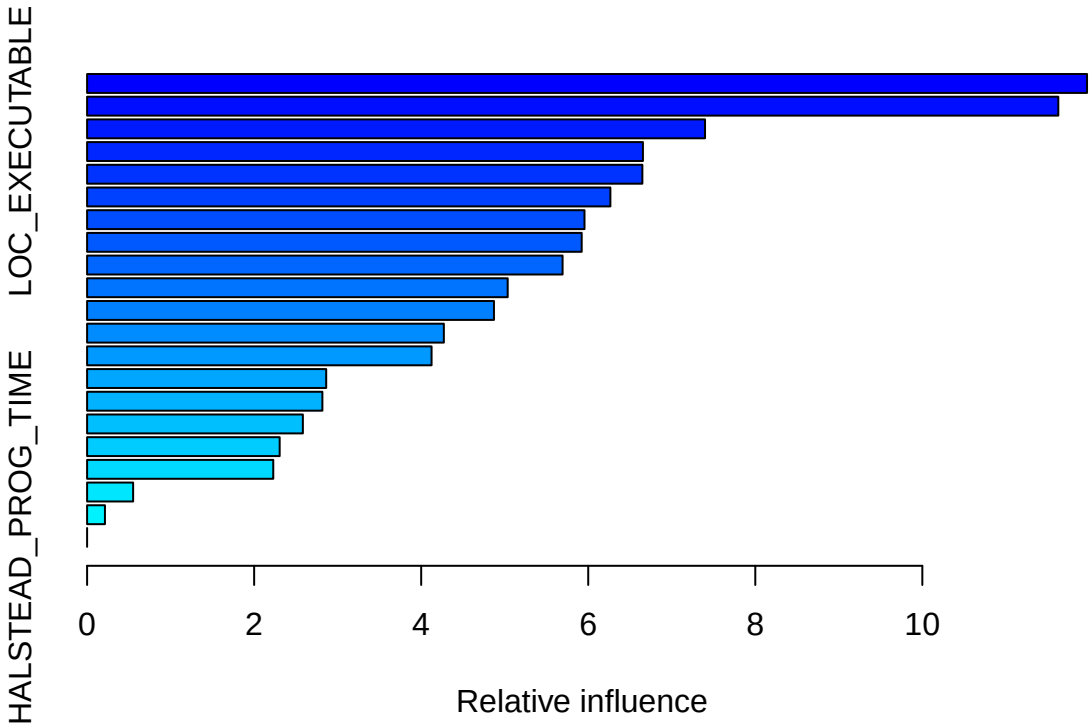
```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8424            -nan     0.1000    0.0102
##      2        0.8182            -nan     0.1000    0.0089
##      3        0.7997            -nan     0.1000    0.0067
##      4        0.7873            -nan     0.1000    0.0055
##      5        0.7786            -nan     0.1000    0.0030
##      6        0.7714            -nan     0.1000    0.0020
##      7        0.7640            -nan     0.1000    0.0024
##      8        0.7544            -nan     0.1000    0.0048
##      9        0.7463            -nan     0.1000    0.0032
##     10        0.7410            -nan     0.1000    0.0015
##     20        0.7080            -nan     0.1000    0.0006
##     40        0.6832            -nan     0.1000   -0.0002
##     60        0.6699            -nan     0.1000   -0.0003
##     80        0.6563            -nan     0.1000   -0.0004
##    100        0.6472            -nan     0.1000    0.0001
##    120        0.6391            -nan     0.1000   -0.0003
##    140        0.6308            -nan     0.1000   -0.0010
##    150        0.6260            -nan     0.1000   -0.0003
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8310            -nan     0.1000    0.0133
##      2        0.8072            -nan     0.1000    0.0089
##      3        0.7864            -nan     0.1000    0.0081
##      4        0.7689            -nan     0.1000    0.0064
##      5        0.7575            -nan     0.1000    0.0047
##      6        0.7422            -nan     0.1000    0.0042
##      7        0.7320            -nan     0.1000    0.0038
##      8        0.7230            -nan     0.1000    0.0017
##      9        0.7120            -nan     0.1000    0.0029
##     10        0.7058            -nan     0.1000    0.0013
##     20        0.6618            -nan     0.1000    0.0015
##     40        0.6252            -nan     0.1000   -0.0012
##     60        0.5999            -nan     0.1000   -0.0010
##     80        0.5761            -nan     0.1000   -0.0004
##    100        0.5596            -nan     0.1000    0.0000
##    120        0.5385            -nan     0.1000   -0.0004
##    140        0.5240            -nan     0.1000   -0.0011
##    150        0.5158            -nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8345            -nan     0.1000    0.0118
##      2        0.8100            -nan     0.1000    0.0114
##      3        0.7857            -nan     0.1000    0.0083
##      4        0.7714            -nan     0.1000    0.0058
##      5        0.7544            -nan     0.1000    0.0049
##      6        0.7401            -nan     0.1000    0.0067
##      7        0.7312            -nan     0.1000    0.0022
##      8        0.7198            -nan     0.1000    0.0033
##      9        0.7109            -nan     0.1000    0.0014
##     10        0.7021            -nan     0.1000    0.0028
##     20        0.6524            -nan     0.1000   -0.0004
##     40        0.6004            -nan     0.1000   -0.0004
##     60        0.5621            -nan     0.1000    0.0000
```

```
##     80        0.5350             -nan     0.1000   -0.0017
##    100        0.5077             -nan     0.1000   -0.0004
##    120        0.4899             -nan     0.1000   -0.0018
##    140        0.4690             -nan     0.1000   -0.0007
##    150        0.4577             -nan     0.1000   -0.0014
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8434             -nan     0.1000    0.0111
##      2        0.8213             -nan     0.1000    0.0092
##      3        0.8043             -nan     0.1000    0.0075
##      4        0.7867             -nan     0.1000    0.0050
##      5        0.7735             -nan     0.1000    0.0050
##      6        0.7650             -nan     0.1000    0.0033
##      7        0.7572             -nan     0.1000    0.0022
##      8        0.7500             -nan     0.1000    0.0010
##      9        0.7425             -nan     0.1000    0.0032
##     10        0.7395             -nan     0.1000    0.0007
##     20        0.7068             -nan     0.1000    0.0002
##     40        0.6827             -nan     0.1000   -0.0005
##     60        0.6672             -nan     0.1000   -0.0006
##     80        0.6570             -nan     0.1000   -0.0005
##    100        0.6460             -nan     0.1000    0.0004
##    120        0.6387             -nan     0.1000   -0.0005
##    140        0.6309             -nan     0.1000   -0.0003
##    150        0.6273             -nan     0.1000   -0.0003
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8252             -nan     0.1000    0.0149
##      2        0.8040             -nan     0.1000    0.0074
##      3        0.7862             -nan     0.1000    0.0052
##      4        0.7709             -nan     0.1000    0.0062
##      5        0.7549             -nan     0.1000    0.0062
##      6        0.7423             -nan     0.1000    0.0033
##      7        0.7322             -nan     0.1000    0.0024
##      8        0.7246             -nan     0.1000    0.0019
##      9        0.7188             -nan     0.1000    0.0006
##     10        0.7103             -nan     0.1000    0.0030
##     20        0.6715             -nan     0.1000    0.0003
##     40        0.6339             -nan     0.1000   -0.0015
##     60        0.6057             -nan     0.1000   -0.0002
##     80        0.5866             -nan     0.1000   -0.0012
##    100        0.5689             -nan     0.1000   -0.0003
##    120        0.5520             -nan     0.1000   -0.0004
##    140        0.5372             -nan     0.1000   -0.0007
##    150        0.5295             -nan     0.1000   -0.0007
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8347             -nan     0.1000    0.0162
##      2        0.8090             -nan     0.1000    0.0107
##      3        0.7864             -nan     0.1000    0.0085
##      4        0.7725             -nan     0.1000    0.0028
##      5        0.7588             -nan     0.1000    0.0026
##      6        0.7448             -nan     0.1000    0.0041
##      7        0.7348             -nan     0.1000    0.0024
```

```
##      8        0.7237             -nan     0.1000    0.0025
##      9        0.7178             -nan     0.1000    0.0012
##     10        0.7049             -nan     0.1000    0.0041
##     20        0.6472             -nan     0.1000   -0.0004
##     40        0.6046             -nan     0.1000   -0.0008
##     60        0.5684             -nan     0.1000   -0.0014
##     80        0.5432             -nan     0.1000   -0.0011
##    100        0.5152             -nan     0.1000   -0.0010
##    120        0.4900             -nan     0.1000   -0.0009
##    140        0.4707             -nan     0.1000   -0.0008
##    150        0.4625             -nan     0.1000   -0.0013
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8337             -nan     0.1000    0.0127
##      2        0.8137             -nan     0.1000    0.0085
##      3        0.7882             -nan     0.1000    0.0102
##      4        0.7683             -nan     0.1000    0.0089
##      5        0.7520             -nan     0.1000    0.0071
##      6        0.7408             -nan     0.1000    0.0030
##      7        0.7295             -nan     0.1000    0.0052
##      8        0.7242             -nan     0.1000    0.0006
##      9        0.7140             -nan     0.1000    0.0037
##     10        0.7086             -nan     0.1000    0.0016
##     20        0.6677             -nan     0.1000   -0.0001
##     40        0.6394             -nan     0.1000   -0.0009
##     60        0.6252             -nan     0.1000   -0.0002
##     80        0.6171             -nan     0.1000   -0.0003
##    100        0.6073             -nan     0.1000   -0.0003
##    120        0.6011             -nan     0.1000   -0.0002
##    140        0.5909             -nan     0.1000   -0.0007
##    150        0.5874             -nan     0.1000   -0.0006
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8310             -nan     0.1000    0.0169
##      2        0.8017             -nan     0.1000    0.0147
##      3        0.7752             -nan     0.1000    0.0120
##      4        0.7533             -nan     0.1000    0.0070
##      5        0.7366             -nan     0.1000    0.0088
##      6        0.7254             -nan     0.1000    0.0029
##      7        0.7137             -nan     0.1000    0.0047
##      8        0.7040             -nan     0.1000    0.0045
##      9        0.6986             -nan     0.1000   -0.0004
##     10        0.6945             -nan     0.1000    0.0005
##     20        0.6445             -nan     0.1000   -0.0006
##     40        0.5992             -nan     0.1000   -0.0003
##     60        0.5740             -nan     0.1000   -0.0007
##     80        0.5518             -nan     0.1000   -0.0002
##    100        0.5276             -nan     0.1000   -0.0005
##    120        0.5105             -nan     0.1000   -0.0009
##    140        0.4939             -nan     0.1000   -0.0003
##    150        0.4839             -nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8219             -nan     0.1000    0.0177
```

```
##      2        0.7959             -nan     0.1000    0.0110
##      3        0.7663             -nan     0.1000    0.0124
##      4        0.7469             -nan     0.1000    0.0052
##      5        0.7293             -nan     0.1000    0.0076
##      6        0.7138             -nan     0.1000    0.0062
##      7        0.7004             -nan     0.1000    0.0052
##      8        0.6883             -nan     0.1000    0.0043
##      9        0.6803             -nan     0.1000    0.0023
##     10        0.6712             -nan     0.1000    0.0005
##     20        0.6192             -nan     0.1000   -0.0007
##     40        0.5594             -nan     0.1000   -0.0000
##     60        0.5264             -nan     0.1000   -0.0015
##     80        0.4934             -nan     0.1000   -0.0003
##    100        0.4707             -nan     0.1000   -0.0009
##    120        0.4484             -nan     0.1000   -0.0007
##    140        0.4306             -nan     0.1000   -0.0006
##    150        0.4218             -nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8323             -nan     0.1000    0.0146
##      2        0.8053             -nan     0.1000    0.0105
##      3        0.7798             -nan     0.1000    0.0119
##      4        0.7633             -nan     0.1000    0.0072
##      5        0.7492             -nan     0.1000    0.0080
##      6        0.7378             -nan     0.1000    0.0033
##      7        0.7258             -nan     0.1000    0.0053
##      8        0.7136             -nan     0.1000    0.0048
##      9        0.7027             -nan     0.1000    0.0040
##     10        0.6940             -nan     0.1000    0.0022
##     20        0.6475             -nan     0.1000   -0.0009
##     40        0.6085             -nan     0.1000    0.0001
##     60        0.5769             -nan     0.1000   -0.0008
##     80        0.5556             -nan     0.1000   -0.0010
##    100        0.5358             -nan     0.1000    0.0001
##    120        0.5147             -nan     0.1000   -0.0010
##    140        0.4989             -nan     0.1000   -0.0005
##    150        0.4924             -nan     0.1000   -0.0006
```

```
# Find out variable importance
summary(objModel)
```




```
##                                       var       rel.inf
## HALSTEAD_CONTENT           HALSTEAD_CONTENT 11.9719209365
## HALSTEAD_DIFFICULTY     HALSTEAD_DIFFICULTY 11.6281222487
## HALSTEAD_EFFORT             HALSTEAD_EFFORT  7.3982708323
## LOC_EXECUTABLE               LOC_EXECUTABLE  6.6547461836
## HALSTEAD_LENGTH             HALSTEAD_LENGTH  6.6481194643
## HALSTEAD_VOLUME             HALSTEAD_VOLUME  6.2651829677
## NUM_OPERANDS                   NUM_OPERANDS  5.9550331042
## LOC_TOTAL                         LOC_TOTAL  5.9217452855
## NUM_UNIQUE_OPERATORS   NUM_UNIQUE_OPERATORS  5.6928518350
## LOC_COMMENTS                 LOC_COMMENTS  5.0355938149
## NUM_UNIQUE_OPERANDS     NUM_UNIQUE_OPERANDS  4.8712914173
## NUM_OPERATORS                 NUM_OPERATORS  4.2714353987
## LOC_BLANK                         LOC_BLANK  4.1242127346
## LOC_CODE_AND_COMMENT   LOC_CODE_AND_COMMENT  2.8625349112
## ESSENTIAL_COMPLEXITY   ESSENTIAL_COMPLEXITY  2.8170600164
## DESIGN_COMPLEXITY         DESIGN_COMPLEXITY  2.5833211480
## CYCLOMATIC_COMPLEXITY CYCLOMATIC_COMPLEXITY  2.3053896008
## BRANCH_COUNT                   BRANCH_COUNT  2.2287410148
## HALSTEAD_ERROR_EST       HALSTEAD_ERROR_EST  0.5509628978
## HALSTEAD_LEVEL               HALSTEAD_LEVEL  0.2134641879
## HALSTEAD_PROG_TIME       HALSTEAD_PROG_TIME  0.0000000000
```

```
# find out model details
objModel
```

```
## Stochastic Gradient Boosting
##
## 1500 samples
##   21 predictors
##    2 classes: 'N', 'Y'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 1000, 1000, 1000
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  ROC           Sens          Spec
##   1                   50      0.8064922834  0.9826224329  0.1495726496
##   1                  100      0.8098239964  0.9802527646  0.1923076923
##   1                  150      0.8110442743  0.9794628752  0.2051282051
##   2                   50      0.8137379998  0.9731437599  0.1965811966
##   2                  100      0.8153785393  0.9644549763  0.2393162393
##   2                  150      0.8135405274  0.9644549763  0.2606837607
##   3                   50      0.8161532385  0.9715639810  0.2136752137
##   3                  100      0.8155608215  0.9644549763  0.2905982906
##   3                  150      0.8164165350  0.9668246445  0.3034188034
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
################################################
# evalutate model
################################################
# get predictions on your testing data

# class prediction
predictions <- predict(object=objModel, kc1.test[,-22], type='raw')
head(predictions)
```

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

```
postResample(pred=predictions, obs=as.factor(kc1.test[,22]))
```

```
##     Accuracy        Kappa
## 0.8657718121 0.2891009393
```

```
# probabilities
predictions <- predict(object=objModel, kc1.test[,-22], type='prob')
head(predictions)
```

```
##              N            Y
## 1 0.9185156057 0.08148439428
## 2 0.9797772040 0.02022279595
## 3 0.8470294127 0.15297058732
## 4 0.8799954090 0.12000459096
## 5 0.8298472743 0.17015272568
## 6 0.9578031188 0.04219688123
```

```
postResample(pred=predictions[[2]], obs=ifelse(kc1.test[,22]=='yes',1,0))
```

```
##         RMSE     Rsquared          MAE
## 0.2165564399           NA 0.1347441023
```

```
auc <- roc(ifelse(kc1.test[,22]=="Y",1,0), predictions[[2]])
print(auc$auc)
```

```
## Area under the curve: 0.8031553
```

Chapter 18

Further Classification Models

18.1 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.)

18.2 Semi-supervised Learning

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

```
library(DMwR)

## 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)
```

```
##
##              setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0         16         2
##   virginica       0          1        17
```

```
## 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)
```

```
##
##              setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0         17         2
##   virginica       0          0        17
```

```
## 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)
```

```
##
##              setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0         17         2
##   virginica       0          0        17
```

Chapter 19

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 objects are masked from 'package:rgp':
##
##     %->%, normalize

## 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:

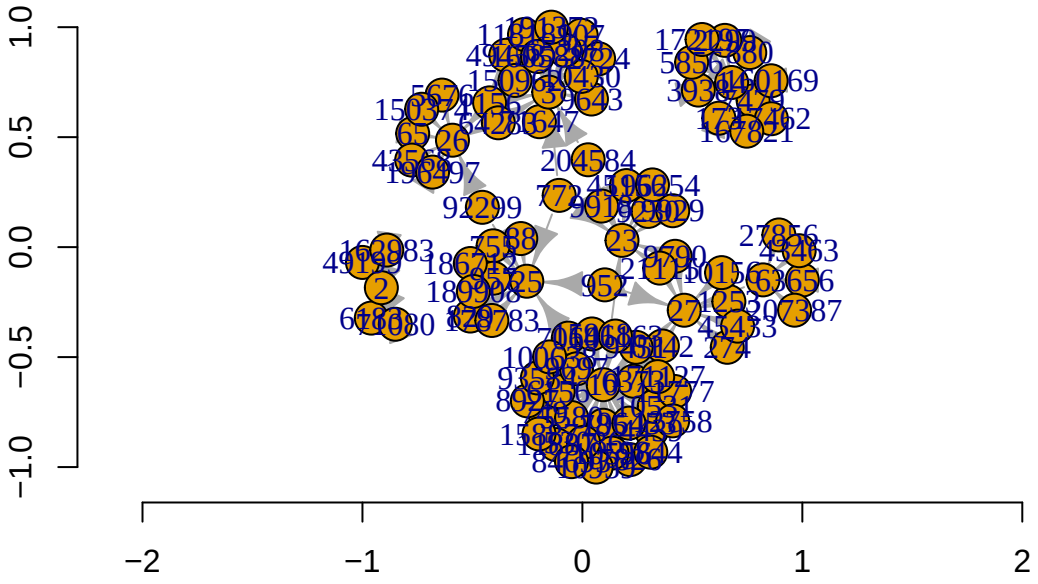
```
summary(g);
```

```
## IGRAPH 18b8578 DN-- 95 100 --
## + attr: name (v/c)
```

Plotting the graph:

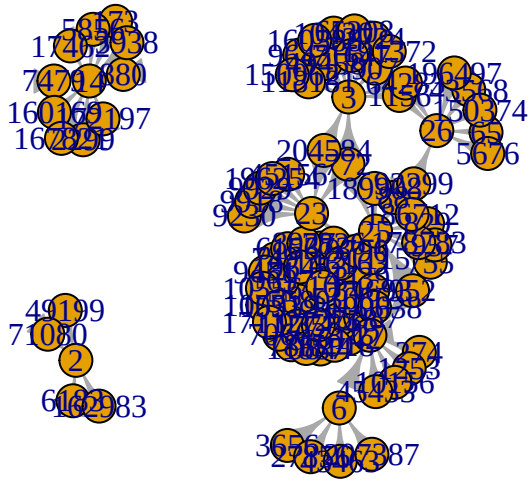
```
layout1 <-  layout.fruchterman.reingold(g)
plot(g, layout1)
```

```
## Warning in if (axes) {: the condition has length > 1 and only the first
## element will be used
```



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)
)
```

```
## Warning in closeness(g): At centrality.c:2784 :closeness centrality is not
## well-defined for disconnected graphs
```

```
#
head(metrics)
```

```
##        deg bet             clo             eig cor
## 6183     1   0 0.0001131733816 0.000000000000   1
## 49199    1   0 0.0001131733816 0.000000000000   1
## 71080    1   0 0.0001131733816 0.000000000000   1
## 162983   1   0 0.0001131733816 0.000000000000   1
## 772      3   0 0.0001156336725 0.104085365595   2
## 907      1   0 0.0001131733816 0.008141832109   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 20

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.

The main package for *text mining* is `tm` (Feinerer and Hornik, 2015, Feinerer et al. (2008)). Another popular package is `wordcloud`.

20.1 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

20.2 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 the traffic green/red/yel
##  $ 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 - I have to pul
## 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
```

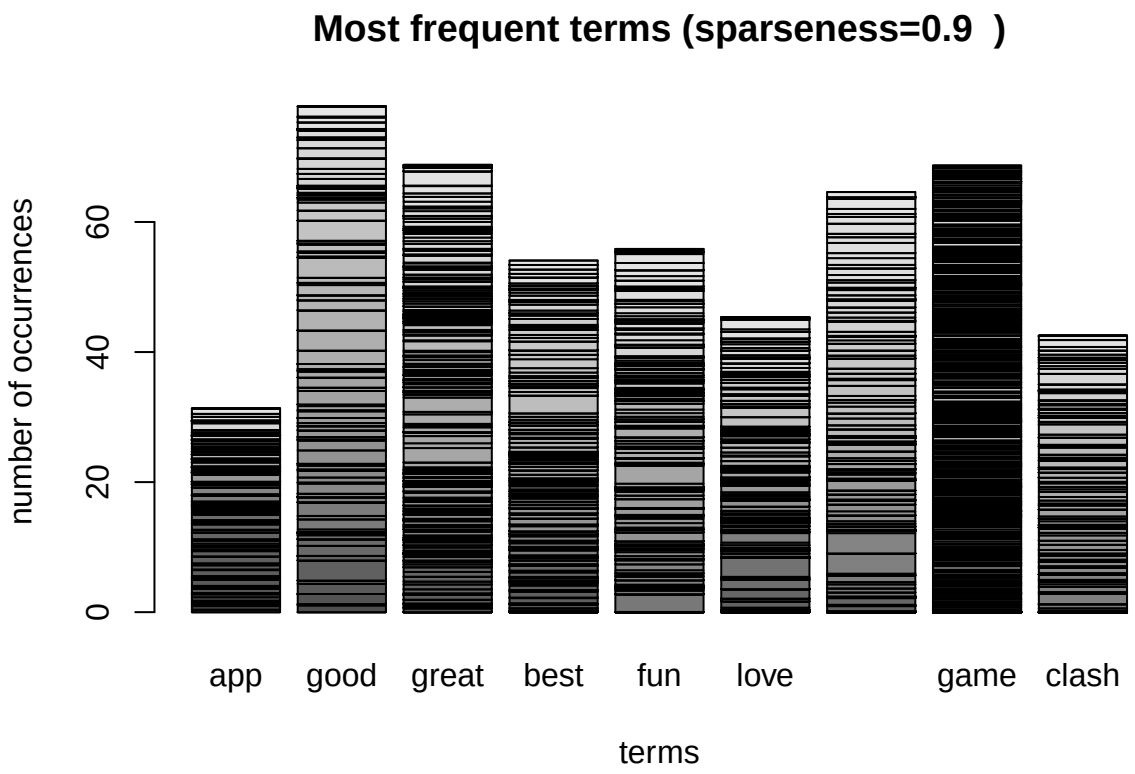
```
##  [1] "google"    "map"       "like"      "app"       "just"
##  [6] "good"      "crashes"   "maps"      "time"      "get"
## [11] "much"      "really"    "update"    "great"     "nice"
## [16] "best"      "ever"      "fun"       "review"    "love"
## [21] "awesome"   "cool"      "amazing"   "game"      "clans"
## [26] "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
##   159   0 0.000000000    0 1.598808368   0    0 0.000000000     0
##   163   0 3.116086850    0 0.000000000   0    0 0.000000000     0
##   178   0 3.116086850    0 0.000000000   0    0 0.000000000     0
##   400   0 0.000000000    0 0.000000000   0    0 3.100319534     0
##   421   0 0.000000000    0 0.000000000   0    0 3.100319534     0
##   472   0 0.000000000    0 0.000000000   0    0 3.100319534     0
##   50    0 0.000000000    0 0.000000000   0    0 3.100319534     0
##   525   0 1.558043425    0 0.000000000   0    0 0.000000000     0
##   527   0 0.000000000    0 0.000000000   0    0 3.100319534     0
##   532   0 0.000000000    0 1.598808368   0    0 0.000000000     0
##      Terms
## Docs         love
##   159 1.461720886
##   163 0.000000000
##   178 0.000000000
##   400 0.000000000
##   421 0.000000000
##   472 0.000000000
##   50  0.000000000
##   525 1.461720886
##   527 0.000000000
##   532 1.461720886
```

```
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
## 77.76636458 68.77637155 68.66996351 64.60649534 55.84940440 54.07372345
##        love       clash         app
## 45.36493131 42.48777519 31.29319954
```

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 predictors
##   2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 475, 474, 475, 474, 474, 474, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy      Kappa
##   1                   50      0.7983107819  0.005187831537
##   1                  100      0.8030309073  0.100972529170
##   1                  150      0.8185586099  0.251719781992
##   2                   50      0.8161942743  0.214069621088
##   2                  100      0.8162028201  0.281341413948
##   2                  150      0.8114798462  0.280302861433
##   3                   50      0.8168636946  0.274826252003
##   3                  100      0.8114883920  0.279902522189
##   3                  150      0.8087850734  0.282684783972
##
## 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 = 150,
##  interaction.depth = 1, 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 0.6128782934 0.38712170665
## 2 0.5754602498 0.42453975017
## 3 0.6422679898 0.35773201018
## 4 0.6866949197 0.31330508027
## 5 0.9802477705 0.01975222952
## 6 0.9307476992 0.06925230082
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N 152   5
##          Y  35   4
##
##                Accuracy : 0.7959184
##                  95% CI : (0.7326392, 0.8500172)
##     No Information Rate : 0.9540816
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0994716
##  Mcnemar's Test P-Value : 0.000004533294
##
##             Sensitivity : 0.8128342
##             Specificity : 0.4444444
##          Pos Pred Value : 0.9681529
##          Neg Pred Value : 0.1025641
##              Prevalence : 0.9540816
##          Detection Rate : 0.7755102
##    Detection Prevalence : 0.8010204
##       Balanced Accuracy : 0.6286393
##
##        '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] 35
```

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

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

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

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

```
#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
```

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

```
## [1] 0.8128342246
```

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

```
## [1] 0.4444444444
```

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

```
## [1] 0.9681528662
```

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

```
## [1] 0.1025641026
```

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

```
## [1] 0.9540816327
```

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

```
## [1] 0.7959183673
```

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

```
## [1] 0.03184713376
```

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

```
## [1] 0.8974358974
```

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

```
## [1] 0.5555555556
```

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

```
## [1] 0.1871657754
```

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)
```

20.3 Extracting data from Twitter

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

Chapter 21

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-evolutionary.js
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 201030 ...
```

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)
```

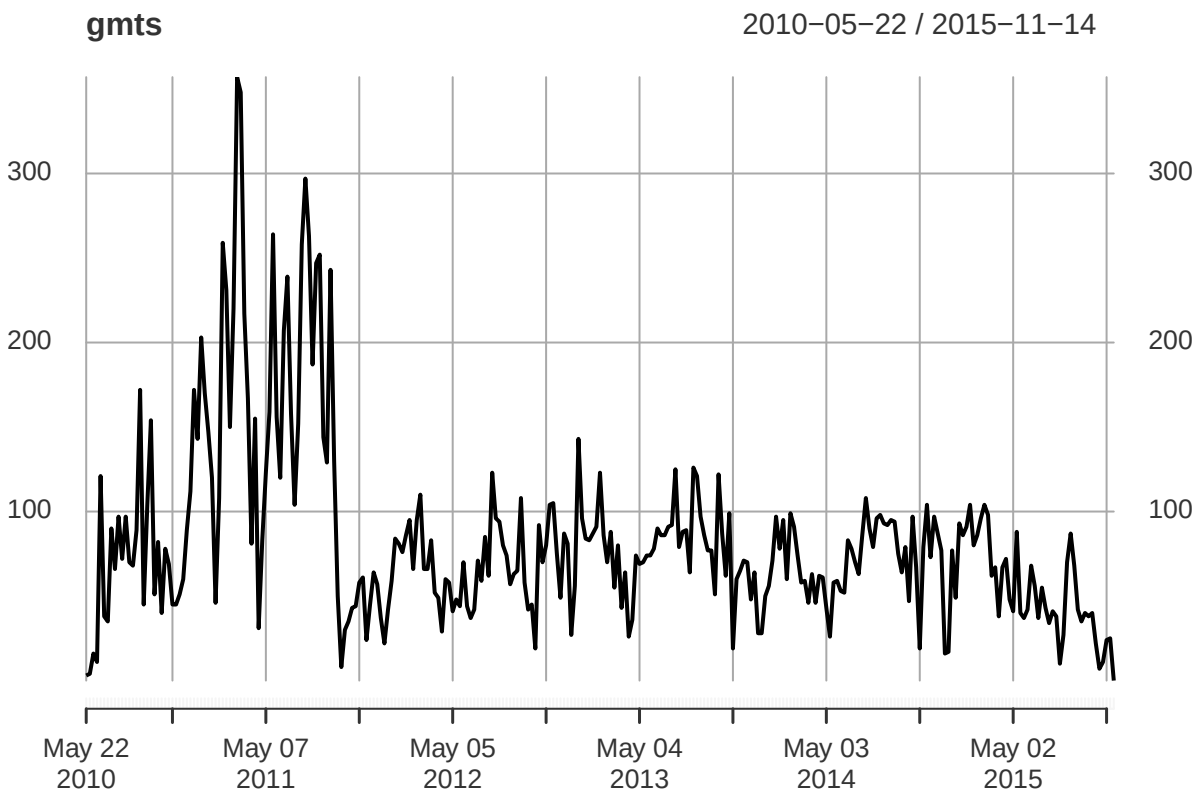
```
## An 'xts' object on 2010-05-22/2015-11-14 containing:
##   Data: int [1:287, 1] 3 4 16 11 121 38 35 90 66 97 ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
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:

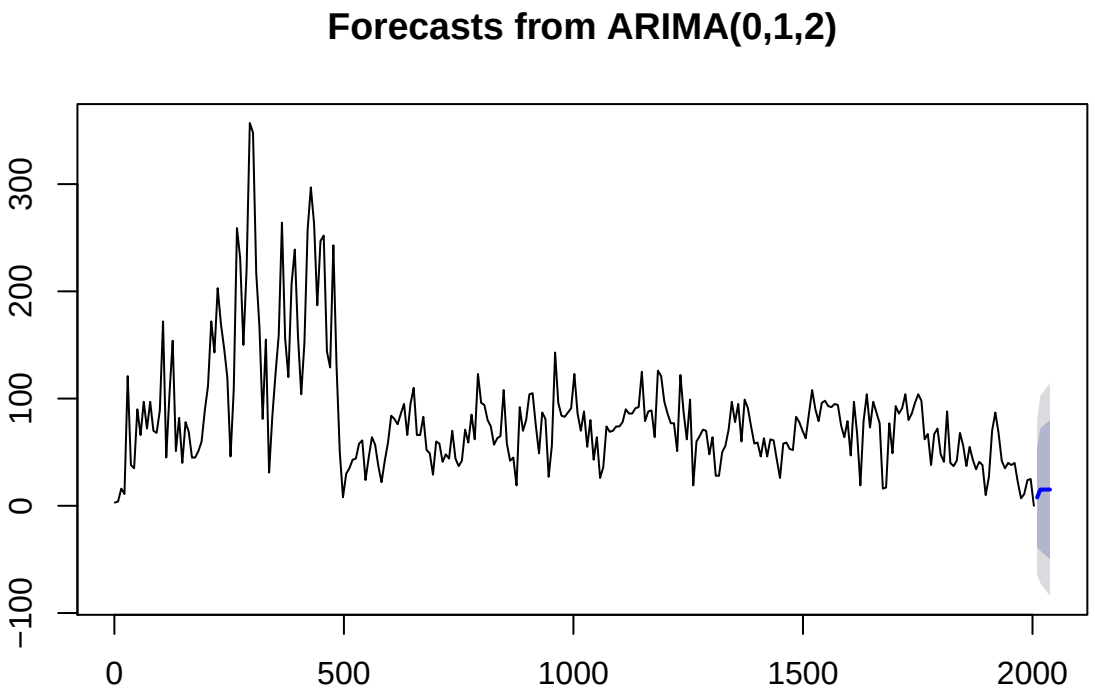
```
fit <- auto.arima(gmts)
fit
```

```
## Series: gmts
## ARIMA(0,1,2)
##
## Coefficients:
##               ma1         ma2
##       -0.3120662  -0.3068537
## s.e.   0.0580588   0.0642417
##
## sigma^2 estimated as 1341.396:  log likelihood=-1434.83
## AIC=2875.67   AICc=2875.75   BIC=2886.64
```

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

```
##      Point Forecast        Lo 80       Hi 80        Lo 95        Hi 95
## 2010    7.748275458 -39.18864059 54.68519151 -64.03554306  79.53209398
## 2017   15.157545088 -41.81337267 72.12846284 -71.97195479 102.28704496
## 2024   15.157545088 -44.55527171 74.87036189 -76.16532750 106.48041768
## 2031   15.157545088 -47.17667914 77.49176931 -80.17442420 110.48951438
## 2038   15.157545088 -49.69220842 80.00729859 -84.02159424 114.33668442
```

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



21.1 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 X

Bibliography

Bibliography

Abran, A. and Robillard, P. (1996). Function points analysis: an empirical study of its measurement processes. *Software Engineering, IEEE Transactions on*, 22(12):895 –910.

Albrecht, A. and Gaffney, J.E., J. (1983). Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering*, 9(6):639–648.

Bailey, J. W. and Basili, V. R. (1981). A meta-model for software development resource expenditures. In *Proceedings of the 5th international conference on Software engineering (ICSE'81)*, ICSE'81, pages 107–116, Piscataway, NJ, USA. IEEE Press.

Banker, R. D., Chang, H., and Kemerer, C. F. (1994). Evidence on economies of scale in software development. *Information and Software Technology*, 36(5):275 – 282.

Belady, L. and Lehman, M. (1979). *Research Directions in Software Technology*, chapter The characteristics of large systems. MIT Press, Cambridge, MA.

Blum, A. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Cano, J. R., Herrera, F., and Lozano, M. (2007). Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability. *Data & Knowledge Engineering*, 60(1):90–108. Intelligent Data Mining.

D'Ambros, M., Lanza, M., and Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR07)*, pages 31–41. IEEE CS Press.

D'Ambros, M., Lanza, M., and Robbes, R. (2011). Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, pages 1–47.

Dash, M., Liu, H., and Motoda, H. (2000). Consistency based feature selection. In *Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 98–109.

Dejaeger, K., Verbeke, W., Martens, D., and Baesens, B. (2012). Data mining techniques for software effort estimation: A comparative study. *Software Engineering, IEEE Transactions on*, 38(2):375 –397.

Desharnais, J. (1988). *Analyse statistique de la productivite des projects de development en informatique a partir de la technique des points de fonction*. Msc thesis, Univ. du Quebec a Montreal.

Dolado, J. (1997). A study of the relationships among albrecht and mark ii function points, lines of code 4gl and effort. *Journal of Systems and Software*, 37(2):161–173.

Dolado, J. (2001). On the problem of the software cost function. *Information and Software Technology*, 43(1):61–72.

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

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34.

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.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181.

Garcia, S., Derrac, J., Cano, J., and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435.

Gray, D., Bowes, D., Davey, N., Sun, Y., and Christianson, B. (2011). The misuse of the nasa metrics data program data sets for automated software defect prediction. In *15th Annual Conference on Evaluation Assessment in Software Engineering (EASE 2011)*, pages 96–103.

Hall, M. (1999). *Correlation-based Feature Selection for Machine Learning*. PhD thesis, University of Waikato, Department of Computer Science, Hamilton, New Zealand.

Hastings, T. and Sajeev, A. (2001). A vector-based approach to software size measurement and effort estimation. *IEEE Transactions on Software Engineering*, 27(4):337–350.

Heiat, A. and Heiat, N. (1997). A model for estimating efforts required for developing small-scale business applications. *Journal of Systems and Software*, 39(1):7–14.

Herraiz, I., Izquierdo-Cortazar, D., Rivas-Hernandez, F., Gonzalez-Barahona, J. M., Robles, G., nas Dominguez, S. D., Garcia-Campos, C., Gato, J. F., and Tovar, L. (2009). FLOSSMetrics: Free / libre / open source software metrics. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE Computer Society.

Hoekstra, R., Morey, R. D., Rouder, J. N., and Wagenmakers, E.-J. (2014). Robust misinterpretation of confidence intervals. *Psychonomic Bulletin & Review*, 21(5):1157–1164.

Howison, J., Conklin, M., and Crowston, K. (2006). FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*, 1(3).

Jeffery, R. and Stathis, J. (1996). Function point sizing: Structure, validity and applicability. *Empirical Software Engineering*, 1:11–30. 10.1007/BF00125809.

Jorgensen, M. (2004). Realism in assessment of effort estimation uncertainty: it matters how you ask. *IEEE Transactions on Software Engineering*, 30(4):209–217.

Jørgensen, M., Indahl, U., and Sjøberg, D. (2003). Software effort estimation by analogy and 'regression toward themean'. *Journal of Systems and Software*, 68(3):253–262.

Jørgensen, M. and Shepperd, M. (2007). A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33(1):33–53.

Kemerer, C. F. (1987). An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429.

Kitchenham, B., Pfleeger, S. L., McColl, B., and Eagan, S. (2002). An empirical study of maintenance and development estimation accuracy. *Journal of Systems and Software*, 64(1):57–77.

Kitchenham, B. A. (2002). The question of scale economies in software — why cannot researchers agree? *Information and Software Technology*, 44(1):13–24.

Kitchenham, B. A. and Taylor, N. (1985). Software project development cost estimation. *Journal of Systems and Software*, 5(4):267–278.

Kohavi, R. and John, G. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 1-2:273–324.

Langley, P. (1994). Selection of relevant features in machine learning. In *Procs. Of the AAAI Fall Symposium on Relevance*, pages 140–144.

Li, J., Ruhe, G., Al-Emran, A., and Richter, M. M. (2007). A flexible method for software effort estimation by analogy. *Empirical Software Engineering*, 12(1):65–106.

Lincke, R., Lundberg, J., and Löwe, W. (2008). Comparing software metrics tools. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis (ISSTA'08)*, ISSTA'08, pages 131–142, New York, NY, USA. ACM.

Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C., and Baldi, P. (2009). Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 18:300–336.

Liu, H. and Motoda, H. (1998). *Feature Selection for Knowlegde Discovery and Data Mining*. Kluwer Academic Publishers, London, UK.

Liu, H. and Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Trans. on Knowledge and Data Eng.*, 17(3):1–12.

Mair, C., Shepperd, M., and Jørgensen, M. (2005). An analysis of data sets used to train and validate cost prediction systems. *SIGSOFT Software Engineering Notes*, 30(4):1–6.

Maxwell, K. (2002). *Applied statistics for software managers*. Prentice Hall.

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

Mišić, V. B. and Tevsić, D. N. (1998). Estimation of effort and complexity: An object-oriented case study. *Journal of Systems and Software*, 41(2):133–143.

Miyazaki, Y., Terakado, M., Ozaki, K., and Nozaki, H. (1994). Robust regression for developing software estimation models. *Journal of Systems and Software*, 27(1):3–16.

Moser, S., Henderson-Sellers, B., and Mišić, V. B. (1999). Cost estimation based on business models. *Journal of Systems and Software*, 49(1):33–42.

Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K., and Murphy, B. (2012). Change bursts as defect predictors. In *21st IEEE International Symposium on Software Reliability Engineering (ISSRE 2012)*.

Nussbaum, L. and Zacchiroli, S. (2010). The ultimate debian database: Consolidating bazaar metadata for quality assurance and data mining. In *7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 52–61.

Schofield, C. (1998). *An Empirical investigation into software effort estimation by analogy*. PhD thesis, Bournemouth University.

Shearer, C. (2000). The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4).

Shepperd, M. and Cartwright, M. (2001). Predicting with sparse data. *Software Engineering, IEEE Transactions on*, 27(11):987 –998.

Shepperd, M. and Schofield, C. (1997). Estimating software project effort using analogies. *Software Engineering, IEEE Transactions on*, 23(11):736 –743.

Shepperd, M., Song, Q., Sun, Z., and Mair, C. (2013). Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215.

Shippey, T., Hall, T., Counsell, S., and Bowes, D. (2016). So you need more method level datasets for your software defect prediction? voila! In *10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'16)*, ESEM'16, pages 12:1–12:6, New York, NY, USA. ACM.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. (2010). Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*.

Ugarte, M., Militino, A., and Arnholt, A. (2015). *Probability and Statistics with R, Second Edition*. Taylor & Francis.

Van Antwerp, M. and Madey, G. (2008). Advances in the sourceforge research data archive (srda). In *Fourth International Conference on Open Source Systems, IFIP 2.13 (WoPDaSD 2008)*, Milan, Italy.

Vasa, R. (2010). *Growth and Change Dynamics in Open Source Software Systems*. PhD thesis, Faculty of Information and Communication Technologies Swinburne University of Technology Melbourne, Australia.

Williams, G. J. (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer.

Woodfield, S., Shen, V., and Dunsmore, H. (1981). A study of several metrics for programming effort. *Journal of Systems and Software*, 2(2):97–103.

Zimmermann, T., Premraj, R., and Zeller, A. (2007). Predicting defects for eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering (PROMISE'07)*, PROMISE '07, pages 9–, Washington, DC, USA. IEEE Computer Society.