Data Analysis in Software Engineering with R

Javier Dolado and Daniel Rodriguez

2017-02-13

Contents

Welcome

This is the website for **Data Analysis in Software Engineering (DASE)**. This book will teach you how to do data science with R in SE.

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 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
- RStudio, RCommander and RKWard are the best known GUIs for R.
- R has multiple online resources and books.
- R coding style
- Getting help in R
  - RStudio cheat sheet
  - Base R cheat sheet
  - Advanced R cheat sheet
  - Data Visualization cheat sheet
  - 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
```

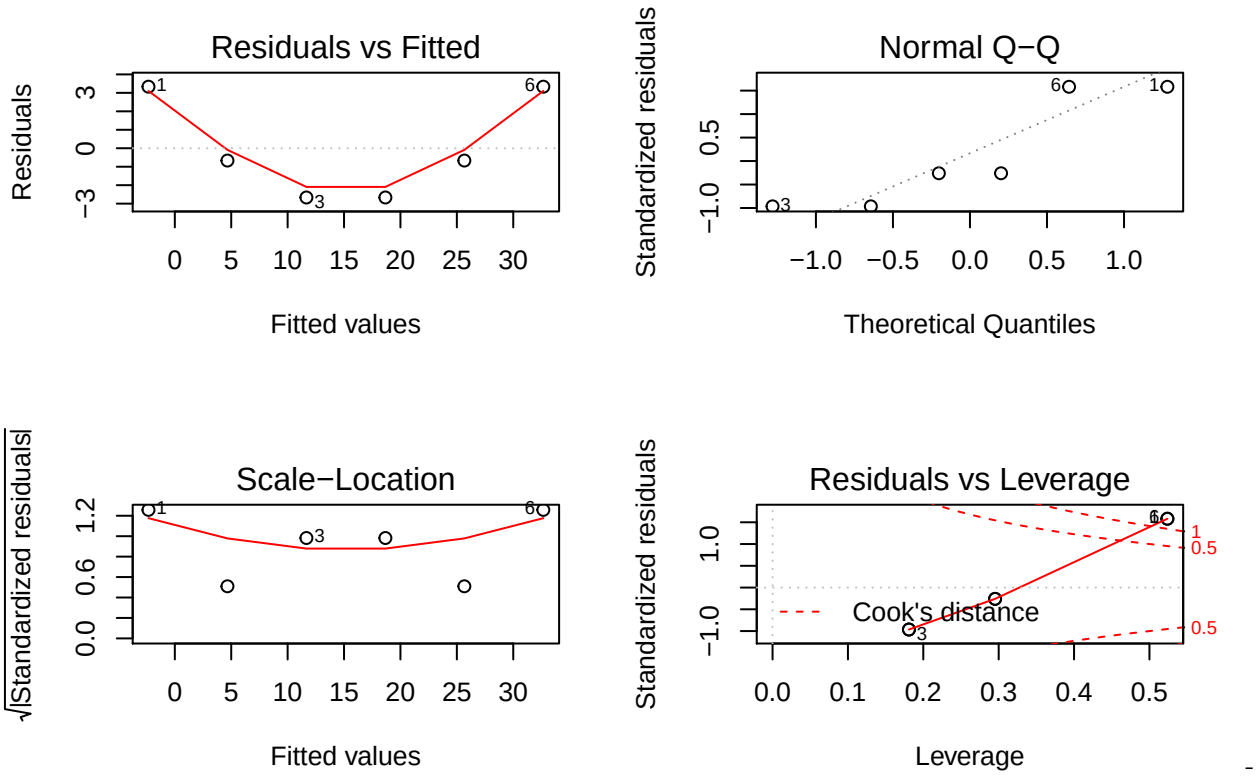
```
##
## Call:
```

```
## 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
```

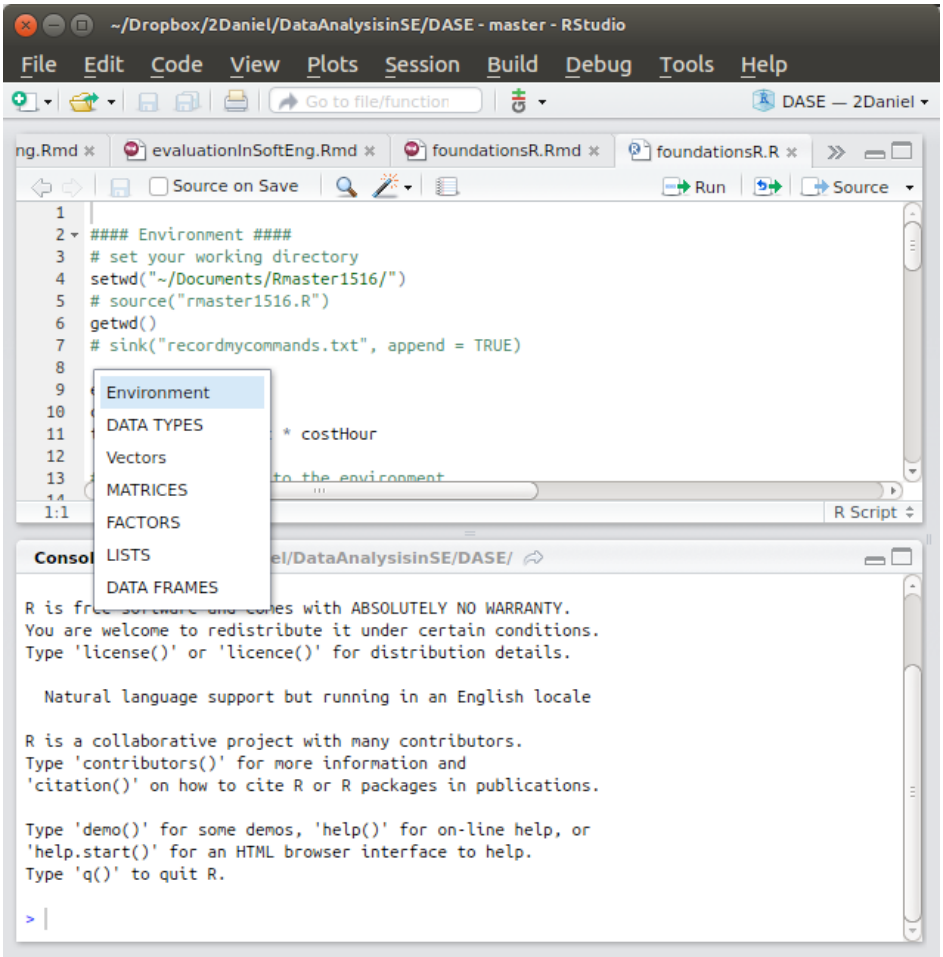


R script. # A file with R commands # comments source("filewithcommands.R") sink("recordmycommands.lis") savehistory() - From command line: + Rscript
Rscript file | -e (e.g. Rscript -e 2+2)
quit()

- Variables
- Operators
  - assign operator <-
  - sequence operator : Example: mynums <- 0:20
  - arithmetic operators: + - = / ^ %/% (integer division) %% (modulus operator)
- The workspace. Objects.
  - ls() objects() ls.str() lists and describes
  - rm(x) delete a variable
  - s.str()
  - objects()
  - rm(totalCost)
  - str( ) #(Structure) provides information about the variable

---

- 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/DASEdown"
```

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

1.2 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")
```

1.2.1 Mising values

- NA 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
```

1.3 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
```

1.3.1 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))
```

1.3.2 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
```

1.4 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,]   15   10   -3   -1
## [2,]   -8    7    3   -6
## [3,]    4    2    1    0
```

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

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

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

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

```
mymat3[3,]
```

```
## [1] 4 2 1 0
```

```
mymat3[,4]
```

```
## [1] -1 -6  0
```

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

```
## [1] 7
```

```
mymat3[9]
```

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

```
## Subsetting multiple elements

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

```
## [1] -8  3
```

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

```
##      [,1] [,2] [,3]
## [1,]   -8    3   -6
## [2,]    4    1    0
```

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

```
## c1 c3
## -8  3
```

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

```
## c1 c3
## -8  3
```

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

```
##    c1 c3 c4
## r2 -8  3 -6
## r3  4  1  0
```

```
### 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
```

1.5 Factors

```
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 ...
```

```
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
```

1.6 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", "Donostia")
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 "Donostia"
```

1.7 Data frames

```
thenames <- c("Ane", "Mike", "Xabi", "Florentino", "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 Florentino   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 4 5 3 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" "Florentino" ...
##  $ 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"       "Florentino" "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
str(mydf0asvec)
```

```
## 'data.frame':    5 obs. of  1 variable:
##  $ Employee: logi  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 Florentino  15     TRUE    176     88
## 5     Edurne  65    FALSE    199    109
```

```
# add a column

birthplace <- c("Tolosa", "London", "Zarautz", "Irun", "New York")

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

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

```
# add a row

xabier <- data.frame(FirstName = "Xabier", Age = 77, Employee=TRUE, height= 170, weight = 65, birthplac
mydataframe <- rbind (mydataframe, xabier)
mydataframe
```

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

```
## 3       Xabi  33     TRUE    158     99    Zarautz
## 4 Florentino  15     TRUE    176     88       Irun
## 5     Edurne  65    FALSE    199    109   New York
## 6     Xabier  77     TRUE    170     65   Donostia
```

```
# sorting

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

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

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

```
##    FirstName Age birthplace
## 6     Xabier  77   Donostia
## 5     Edurne  65   New York
## 1        Ane  44     Tolosa
## 3       Xabi  33    Zarautz
## 2       Mike  20     London
## 4 Florentino  15       Irun
```

1.8 Reading Data

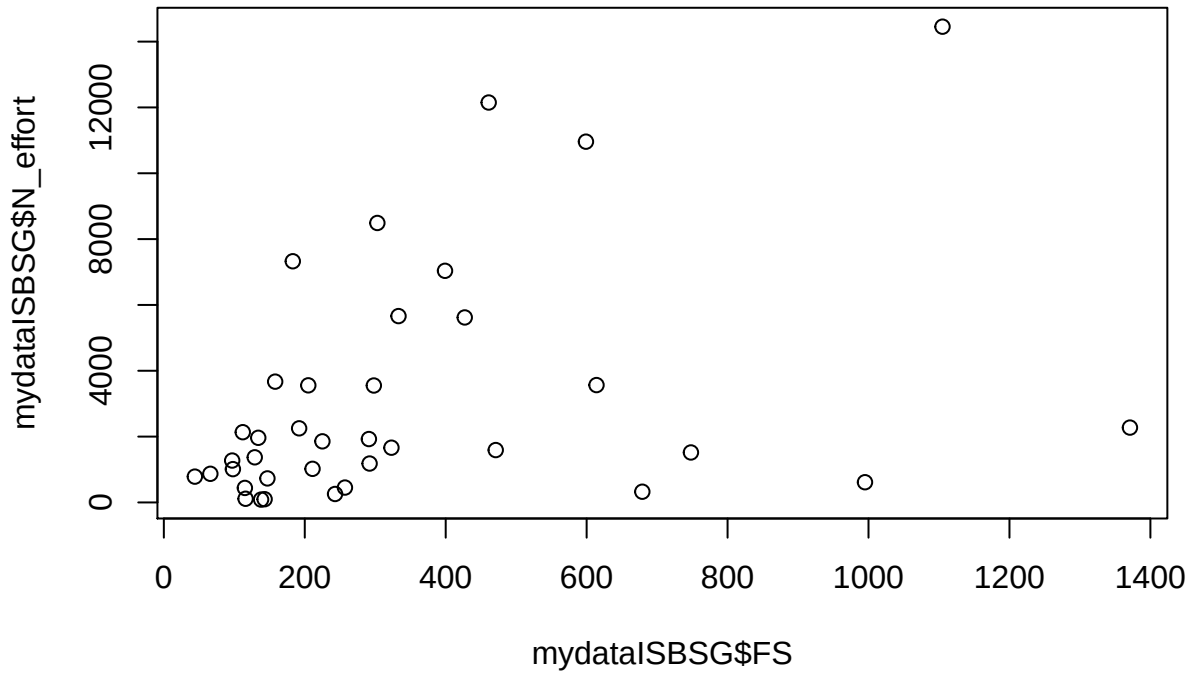
```
library(foreign)
isbsg <- read.arff("datasets/effortEstimation/isbsg10teaser.arff")

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


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 ...
```

1.9 Plots

There are several graphic packages that are recommended, in particular ggplot. However, there is some basic support in the R base for graphics:

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



1.10 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
```

(Torgo, 2010) (García et al., 2015)

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)

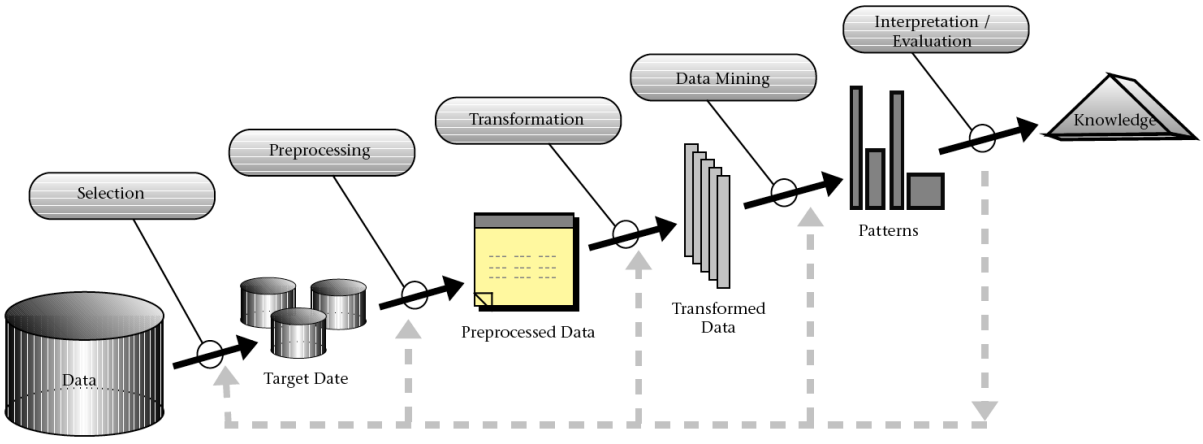


Figure 2.1: KDD Process

The Cross Industry Process for Data Mining (CRISP-DM, 1996) also provides a common and well-developed framework for delivering data mining projects identifying six steps:

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

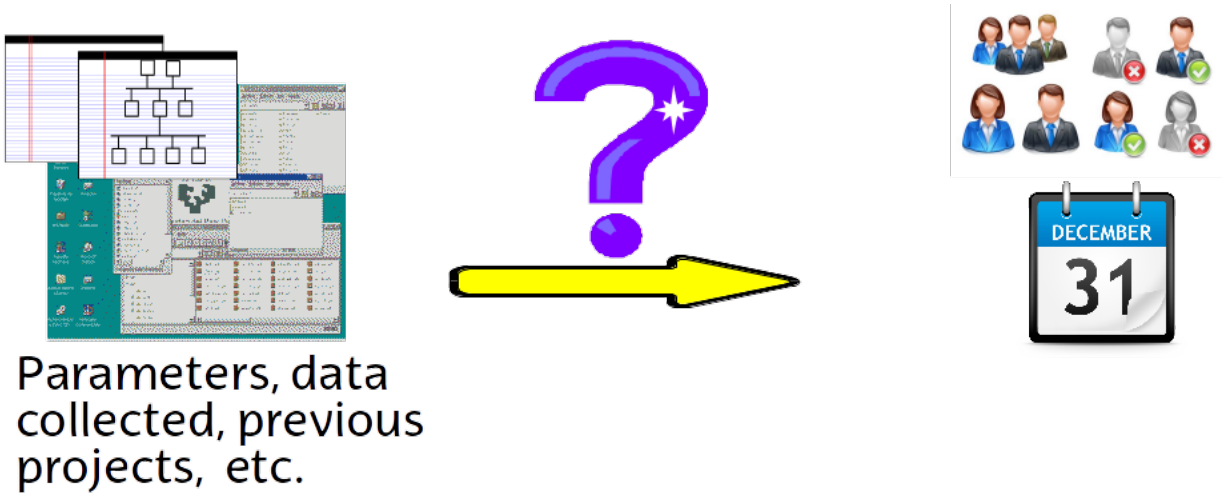


Figure 2.2:

- 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, Data Mining: Practical Machine Learning Tools and Techniques, MK 2011, ISBN: 978-0-12-374856-0

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.

3.1 Types of information stored in the repositories

- Meta-information about the project itself and the people that participated.
  - Low-level information
    - Mailing Lists (ML)
    - Bugs 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.
- 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

3.2 Repositories

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

- FLOSSMole (Howison et al., 2006) http://flossmole.org/
- FLOSSMetrics (Herraiz et al., 2009): http://flossmetrics.org/
- PROMISE (PRedictOr Models In Software Engineering) [8]: http://openscience.us/repo/
- Qualitas Corpus (QC) [9]: http://qualitascorpus.com/
- Sourcerer Project [10]: http://sourcerer.ics.uci.edu/
- Ultimate Debian Database (UDD) [11] http://udd.debian.org/
- Software-artifact Infrastructure Repository (SIR) [16] http://sir.unl.edu
- OpenHub: https://www.openhub.net/ (before Ohloh [17])
- SourceForge Research Data Archive (SRDA) [18] http://zerlot.cse.nd.edu/
- SECOLD (Source code ECOsystem Linked Data): http://www.secold.org/

Not openly available:

- 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 [19]: http://www.ict.swin.edu.au/research/projects/helix/
- Bug Prediction Dataset (BPD) [12,13]: http://bug.inf.usi.ch/
- Eclipse Bug Data (EBD) [11,15]: http://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/

3.3 Some Tools to extract data and dashboards

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

SonarQube http://www.sonarqube.org/

3.4 References (To be fixed)

6 J. Howison, M. Conklin, and K. Crowston, "FLOSSmole: A collaborative repository for FLOSS research data and analyses," International Journal of Information Technology and Web Engineering, vol. 1, no. 3, 2006.

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

8 G. Boetticher, T. Menzies, and T. Ostrand. (2007) Promise repository of empirical software engineering data. West Virginia University, Department of Computer Science. [Online]. Available: http://promisedata.org/repository

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

10 E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: mining and searching internet-scale software repositories," Data Mining and Knowledge Discovery, vol. 18, pp. 300-336, 2009, 10.1007/s10618-008-0118-x.

11 T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in Proceedings of the Third International Workshop on Predictor Models in Software Engineering (PROMISE'07), ser. PROMISE '07.Washington, DC, USA: IEEE Computer Society, 2007, pp. 9-.

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

13 M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," Empirical Software Engineering, pp. 1-47, 10.1007/s10664-011-9173-9.

14 C. Lokan, T. Wright, P. Hill, and M. Stringer, "Organizational benchmarking using the isbsg data repository," IEEE Software, vol. 18, no. 5, pp. 26 -32, sep/oct 2001.

15 N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change bursts as defect predictors," in Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE 2012), November 2010.

16 H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," Empirical Software Engineering, vol. 10, pp. 405-435, October 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1089922.1089928

17 Ohloh, "Ohloh." [Online]. Available: http://www.ohloh.net/

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

19 R. Vasa, "Growth and change dynamics in open source software systems," Ph.D. dissertation, Faculty of Information and Communication Technologies Swinburne University of Technology Melbourne, Australia, 2010.

Part III

Exploratory and Descriptive Data analysis

Chapter 4

Exploratory Data Analysis

4.1 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)

4.2 Basic Plots

```
+ **Histogram**: The histogram defines a sequence of breaks and then counts the number of observations in the

+ **Boxplot**: The boxplot is used to summarize data succinctly, quickly displaying if the data is symmetric


  ![](figures/boxplotexp.png)


+ **Q-Q plot**: This plot is used to determine if the data is close to being normally distributed. The quantil

+ **Scatterplot**: A scatter plot provides a graphical view of the relationship between two sets of numbers:

+ **Kernel Density Plot**: This plot visualizes the underlying distribution of a variable. Kernel density es
+ **Violin Plot**: A violin plot is a combination of a boxplot and a kernel density plot.
```

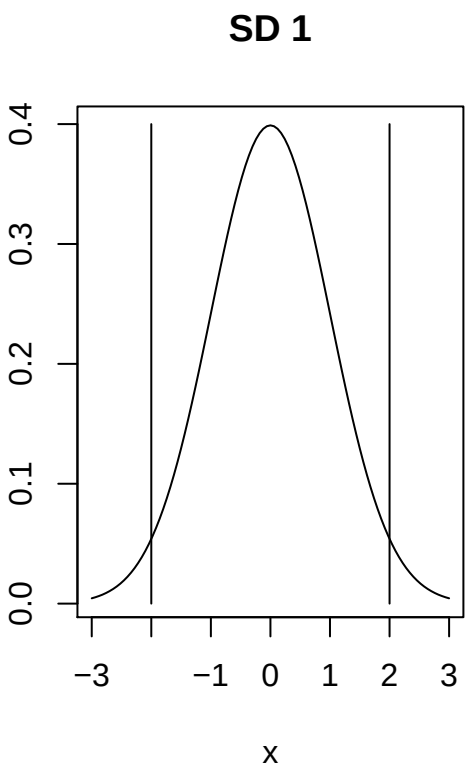
Chapter 5

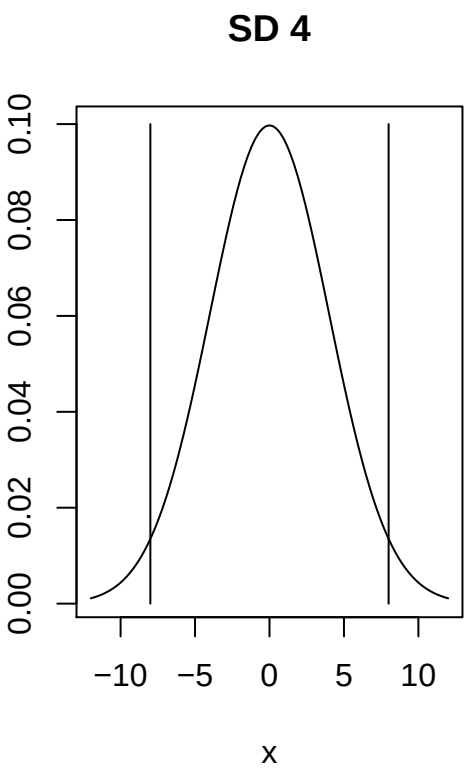
Descriptive Statistics

5.1 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.
- 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)

```
main.title <- "Area within 2 SD 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)
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
```

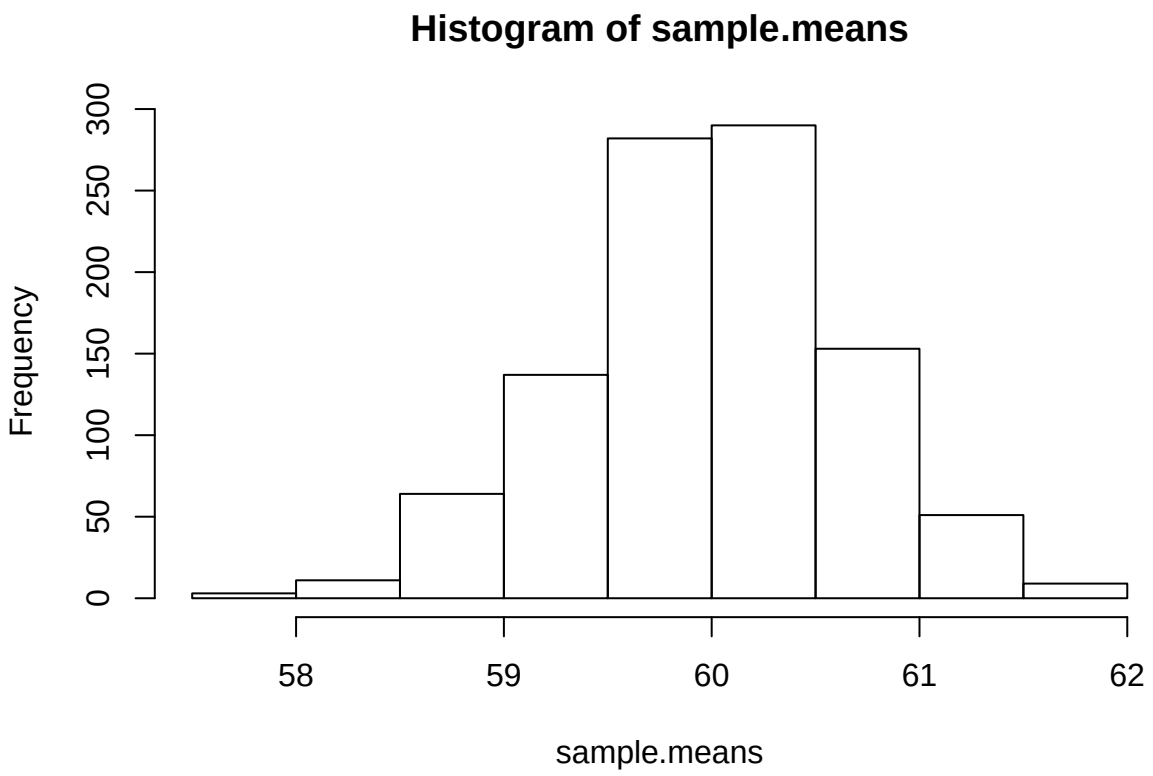
```
## [1] 59.99138
```

```
sd40
```

```
## [1] 0.656247
```

- 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".

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



5.2 Getting the Data. Descriptive statistics.

- Set the path to your files
- Read the Telecom1 dataset and print out the summary statistics with the command *summary*

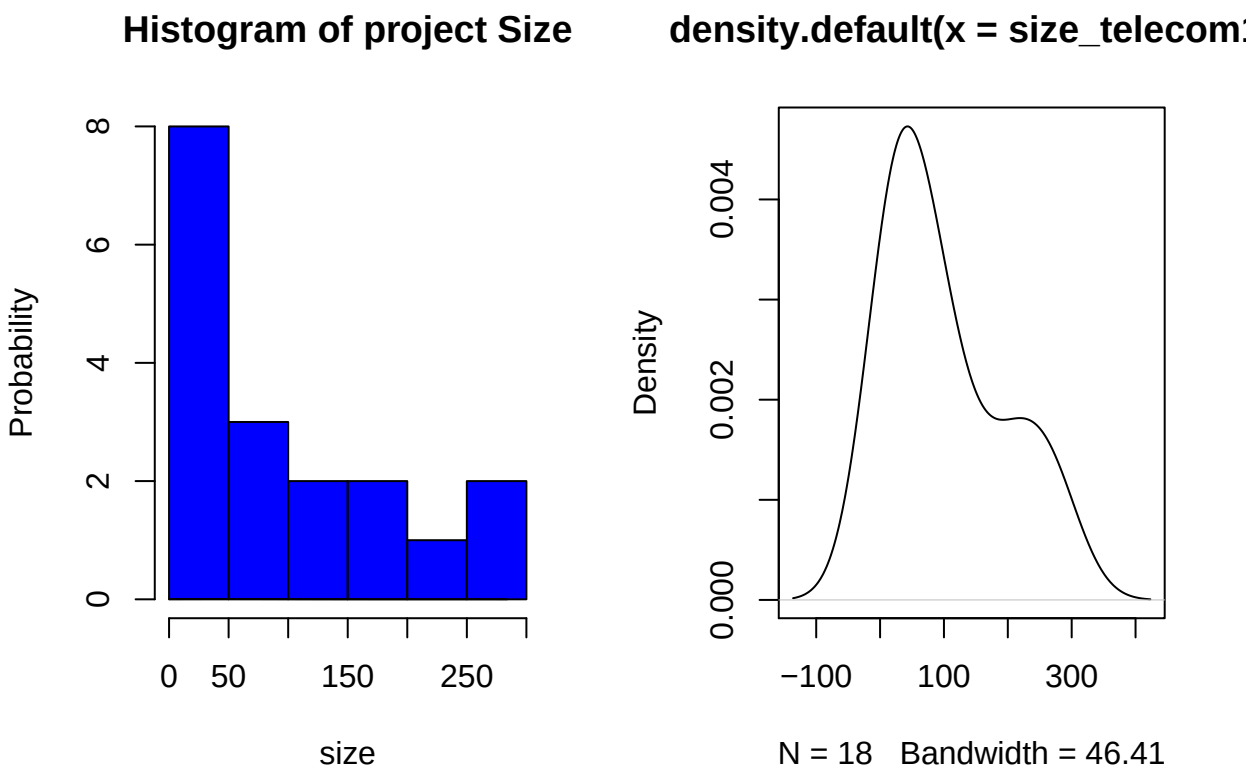
```
options(digits=3)
telecom1 <- read.table("datasets/effortEstimation/Telecom1.csv", sep=",",header=TRUE, stringsAsFactors=
summary(telecom1)
```

```
##       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
```

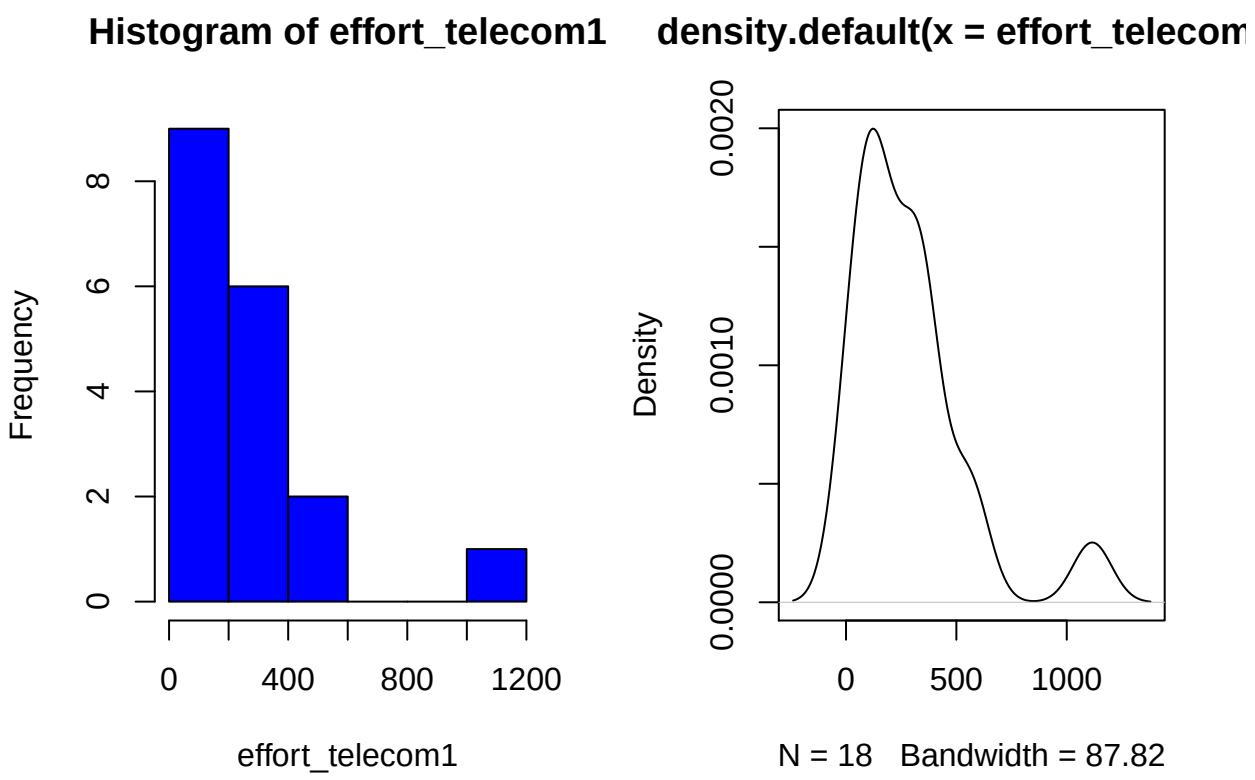
- We see that this dataset has three variables (or parameters) and few data points (18)
  - size: the independent variable
  - effort: the dependent variable
  - EstTotal: the estimates coming from an estimation method
- Basic Plots

```
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 Size')
lines(density(size_telecom1, na.rm = T, from = 0, to = max(size_telecom1)))
```

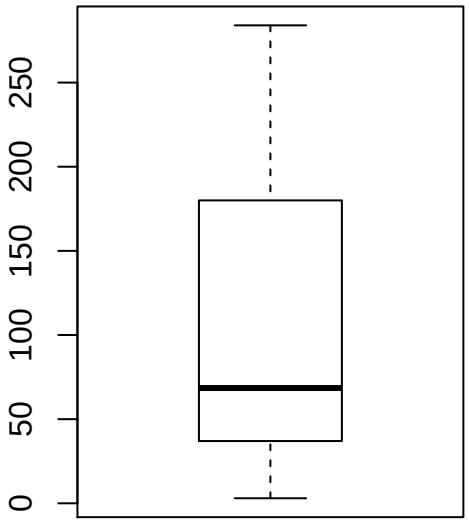
```
plot(density(size_telecom1))
```

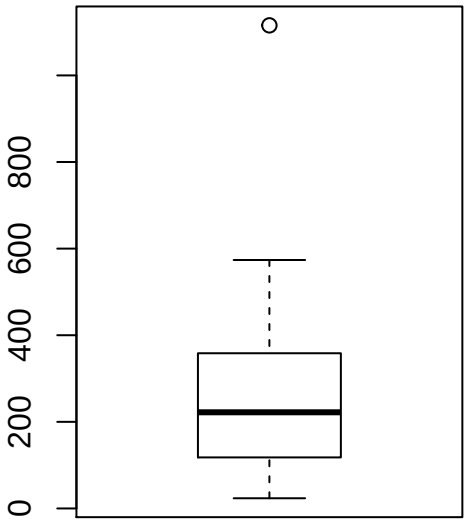



```
hist(effort_telecom1, col="blue")
plot(density(effort_telecom1))
```

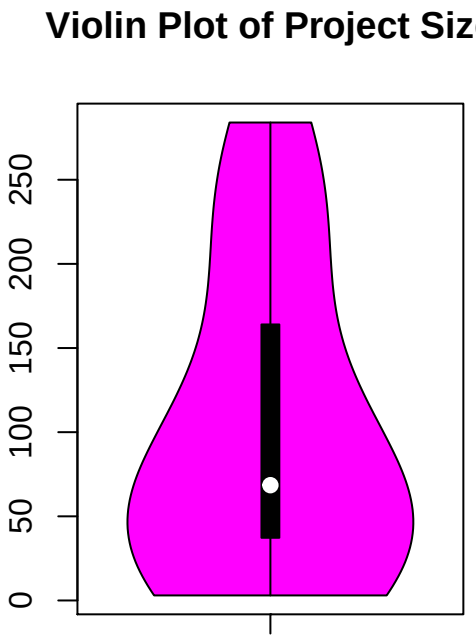


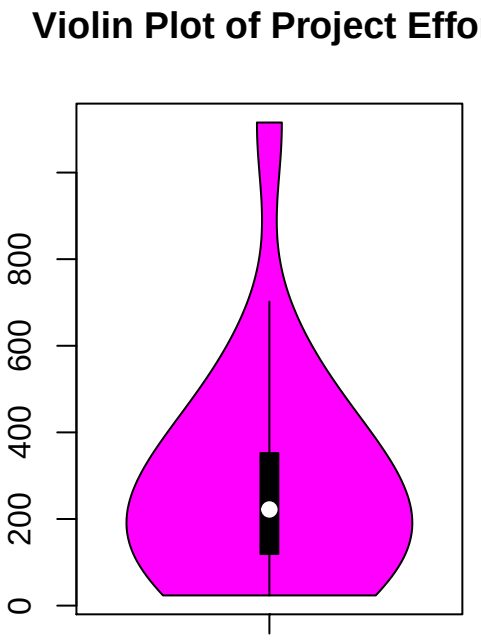
```
boxplot(size_telecom1)
boxplot(effort_telecom1)
```



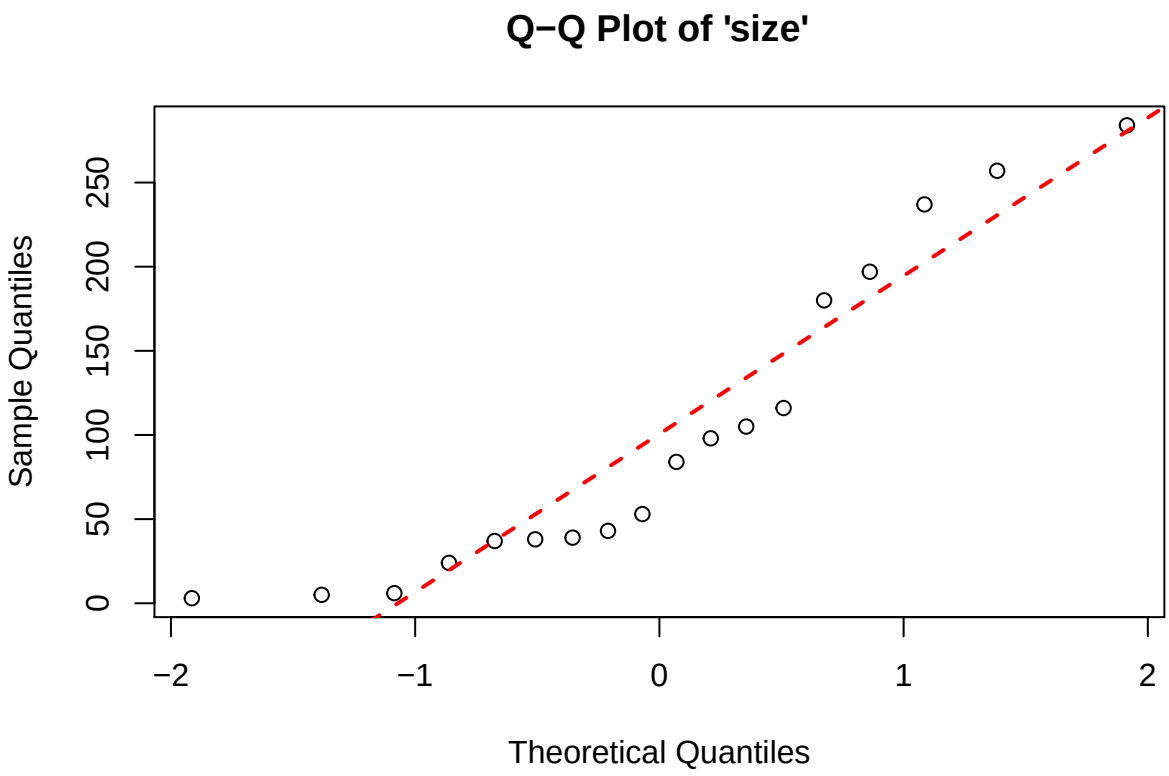



```
# violin plots for those two variables
library(vioplot)
vioplot(size_telecom1, names = '')
title("Violin Plot of Project Size")
vioplot(effort_telecom1, names = '')
title("Violin Plot of Project Effort")
```

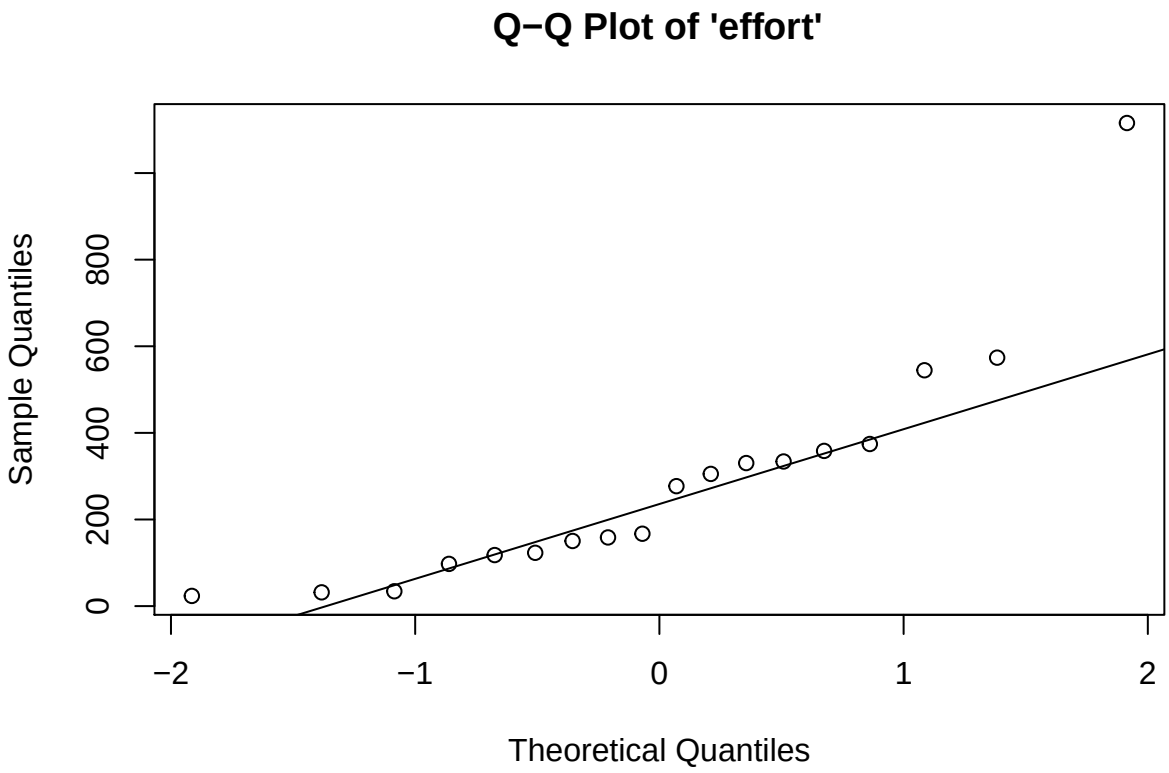





```
par(mfrow=c(1,1))
qqnorm(size_telecom1, main="Q-Q Plot of 'size'")
qqline(size_telecom1, col=2, lwd=2, lty=2) #draws a line through the first and third quartiles
```



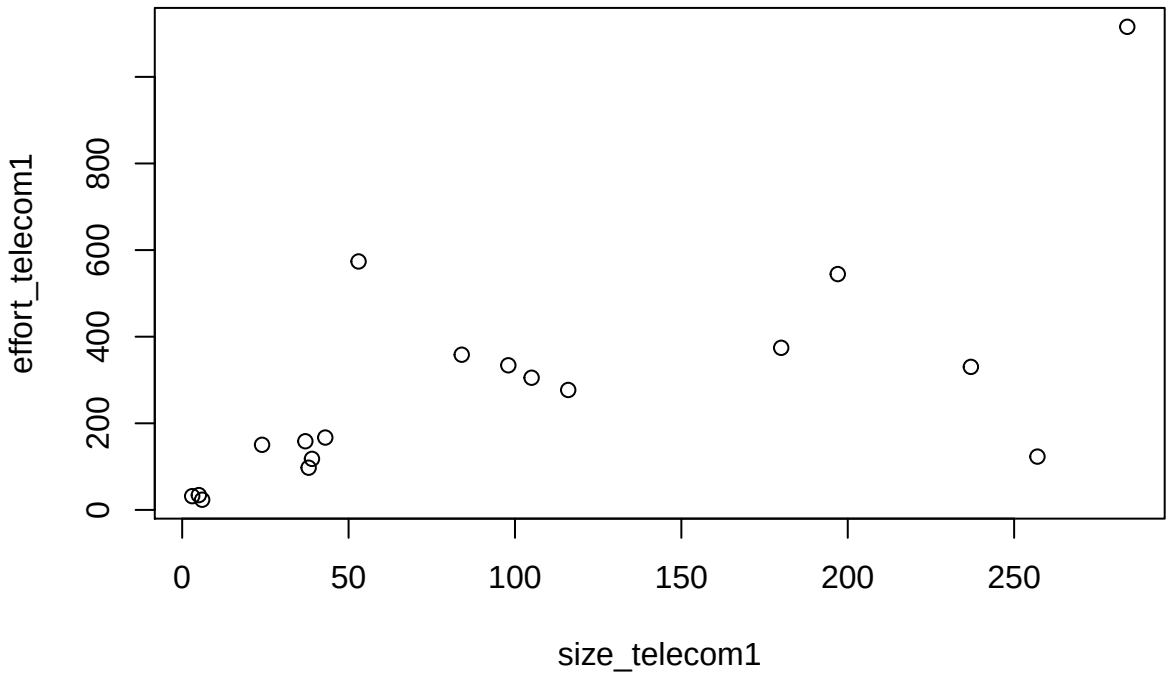

```
qqnorm(effort_telecom1,  main="Q-Q Plot of 'effort'")
qqline(effort_telecom1)
```



- We 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)
```



5.3 China dataset

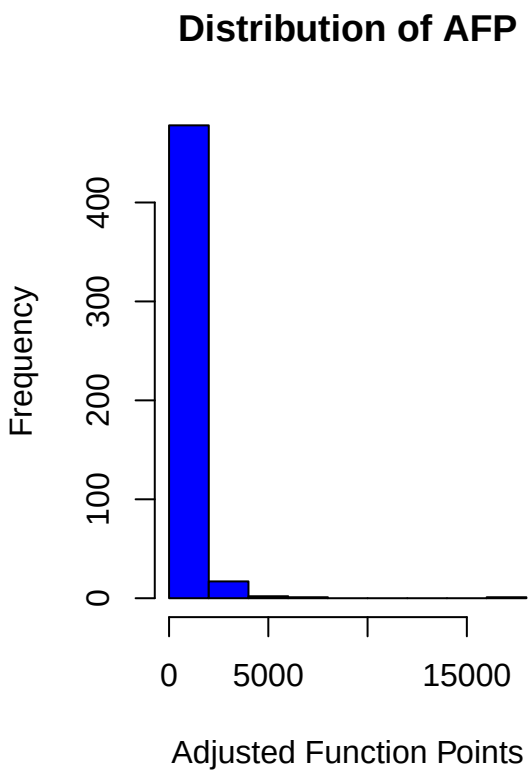
```
library(foreign)
china <- read.arff("datasets/effortEstimation/china.arff")
china_size <- china$AFP
summary(china_size)
```

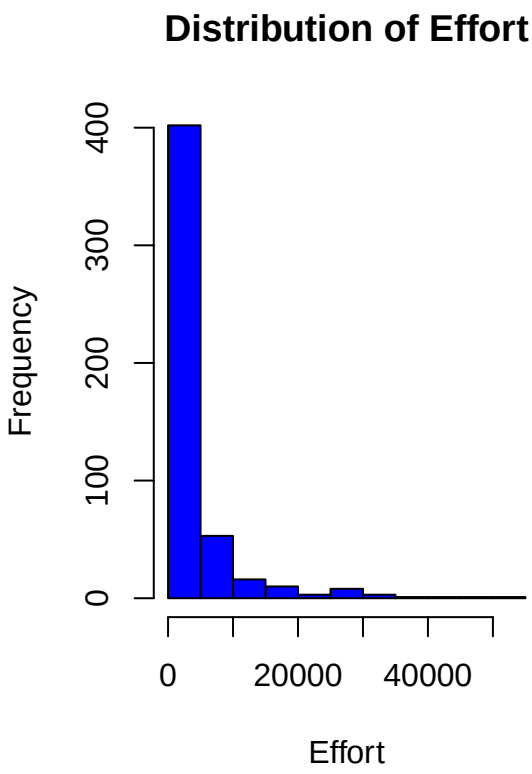
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       9     100     215     487     438   17500
```

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

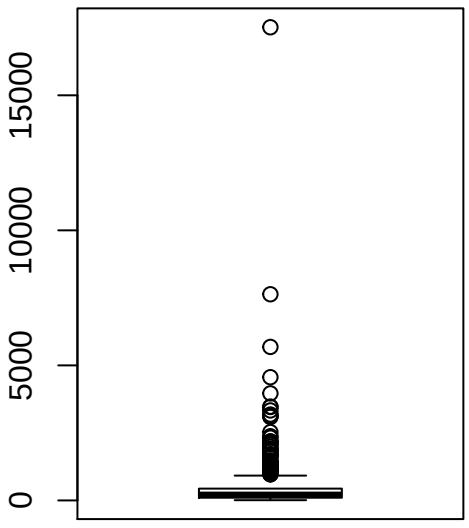
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      26     704    1830    3920    3830   54600
```

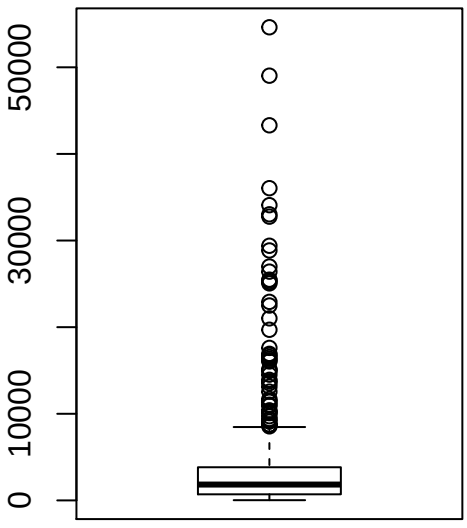
```
par(mfrow=c(1,2))
hist(china_size, col="blue", xlab="Adjusted Function Points", main="Distribution of AFP")
hist(china_effort, col="blue",xlab="Effort", main="Distribution of Effort")
```



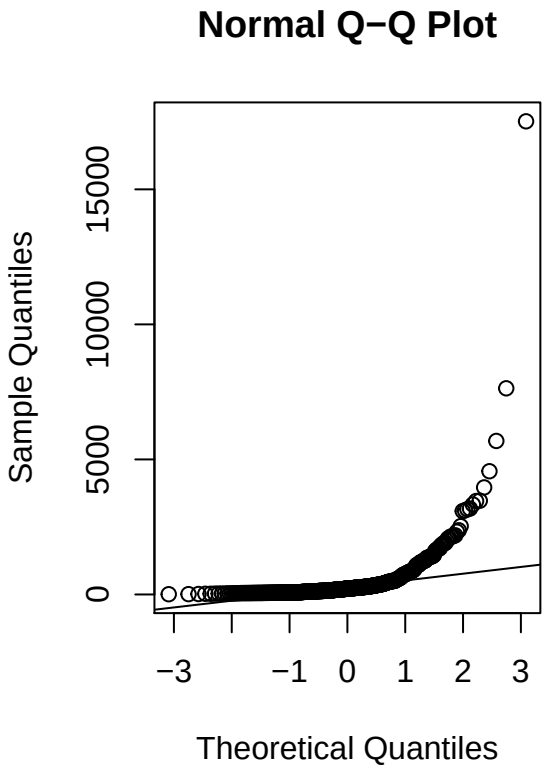


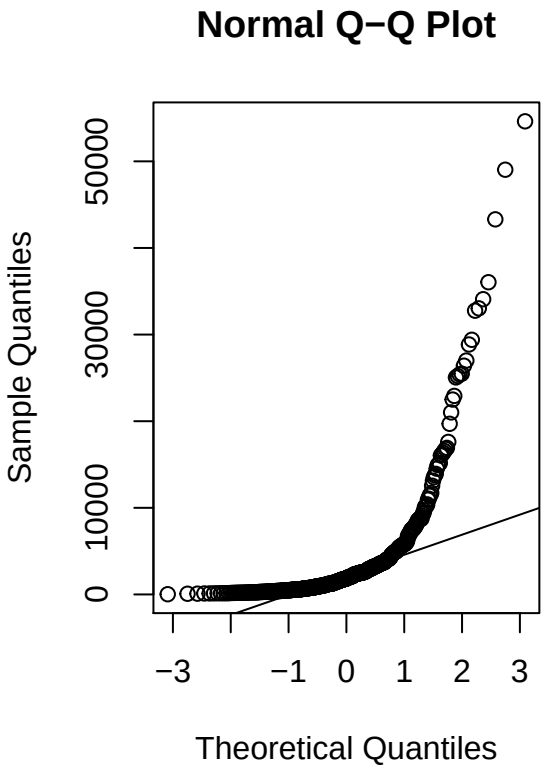

```
boxplot(china_size)
boxplot(china_effort)
```






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

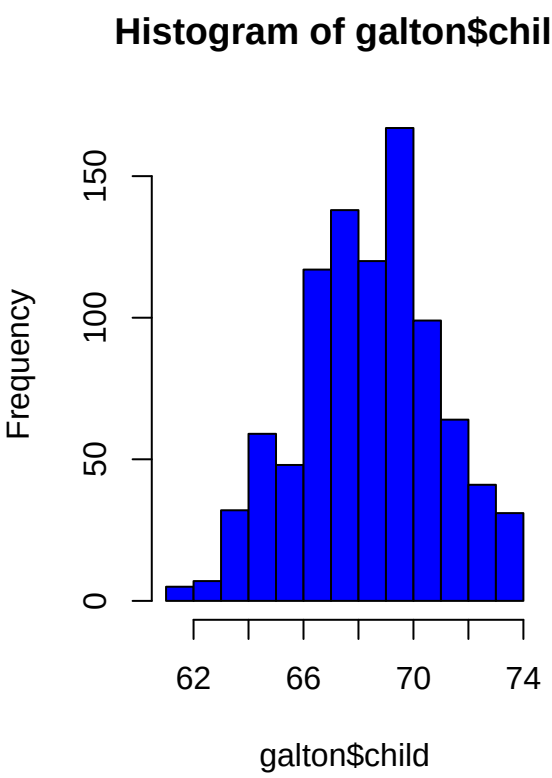


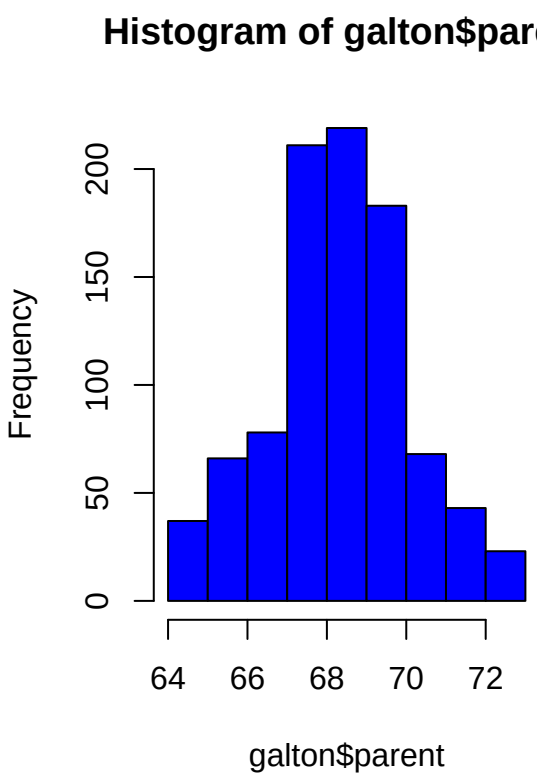


- We observe the non-normality of the data.

5.4 Normality. Galton data

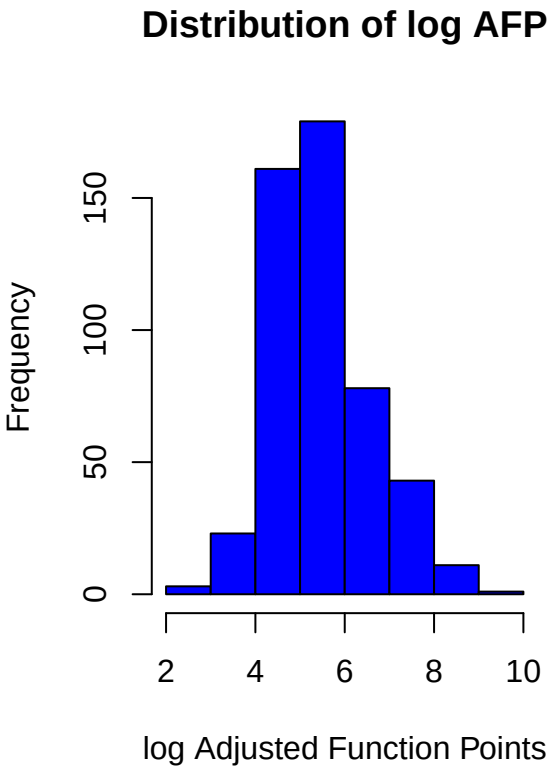
- It is the data based on the famous 1885 Francis Galton's study about the relationship between the heights of adult children and the heights of their parents.

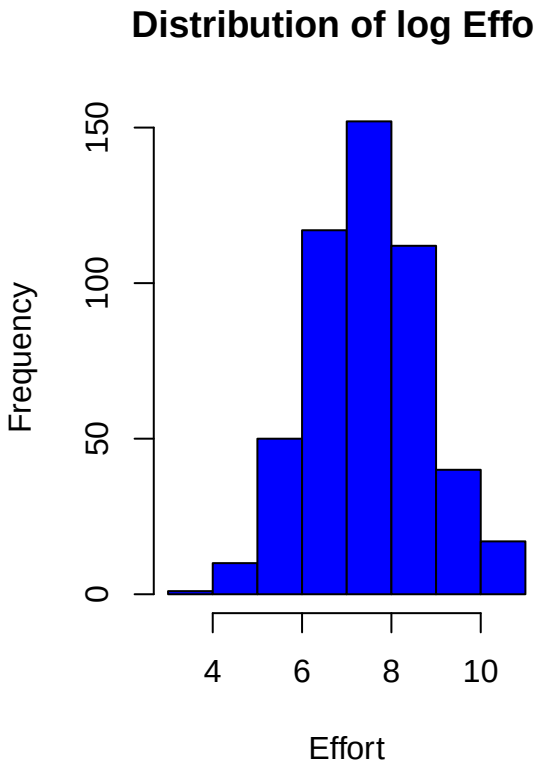


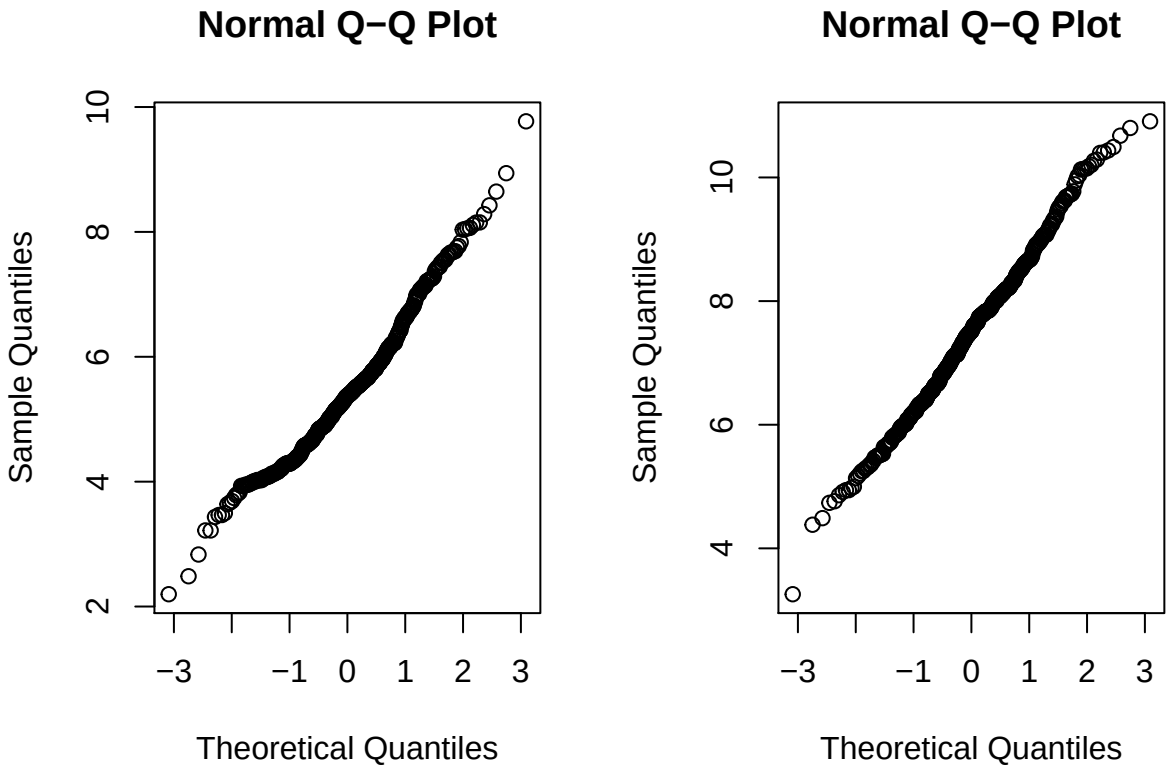


5.5 Normalization

- China dataset. Take logs in both independent variables.







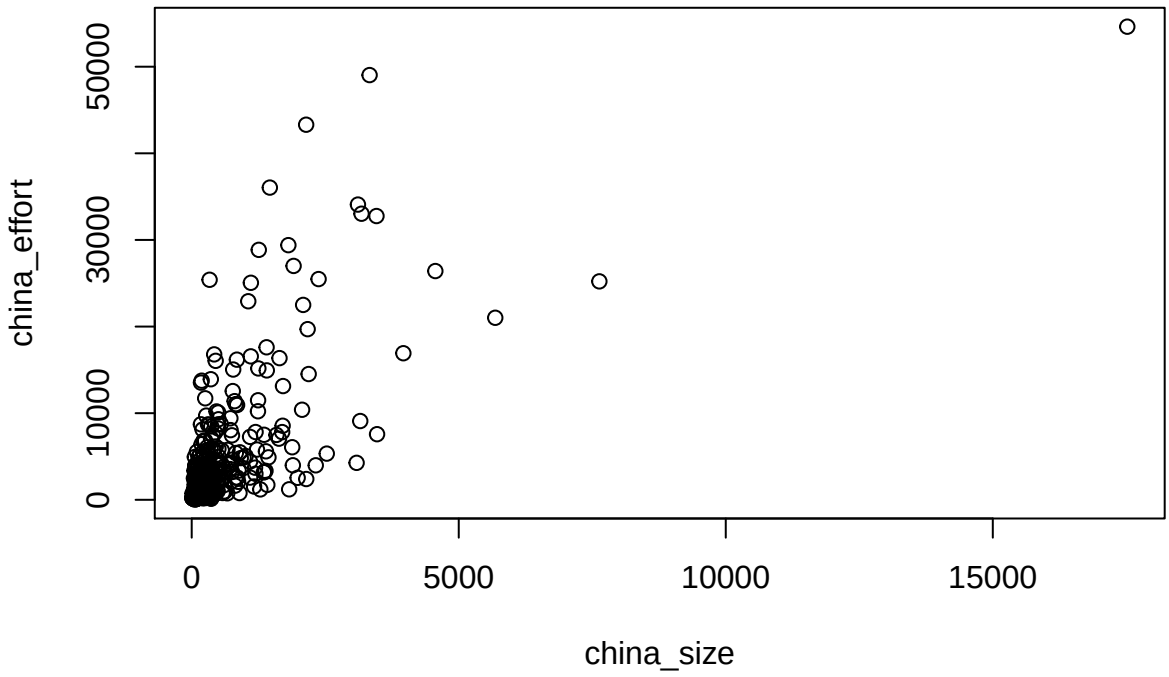
- If the log-log transformation is used the estimation equation is:

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

5.6 China dataset. 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.

```
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
```

Chapter 6

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
- 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
```

		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

Figure 6.1:

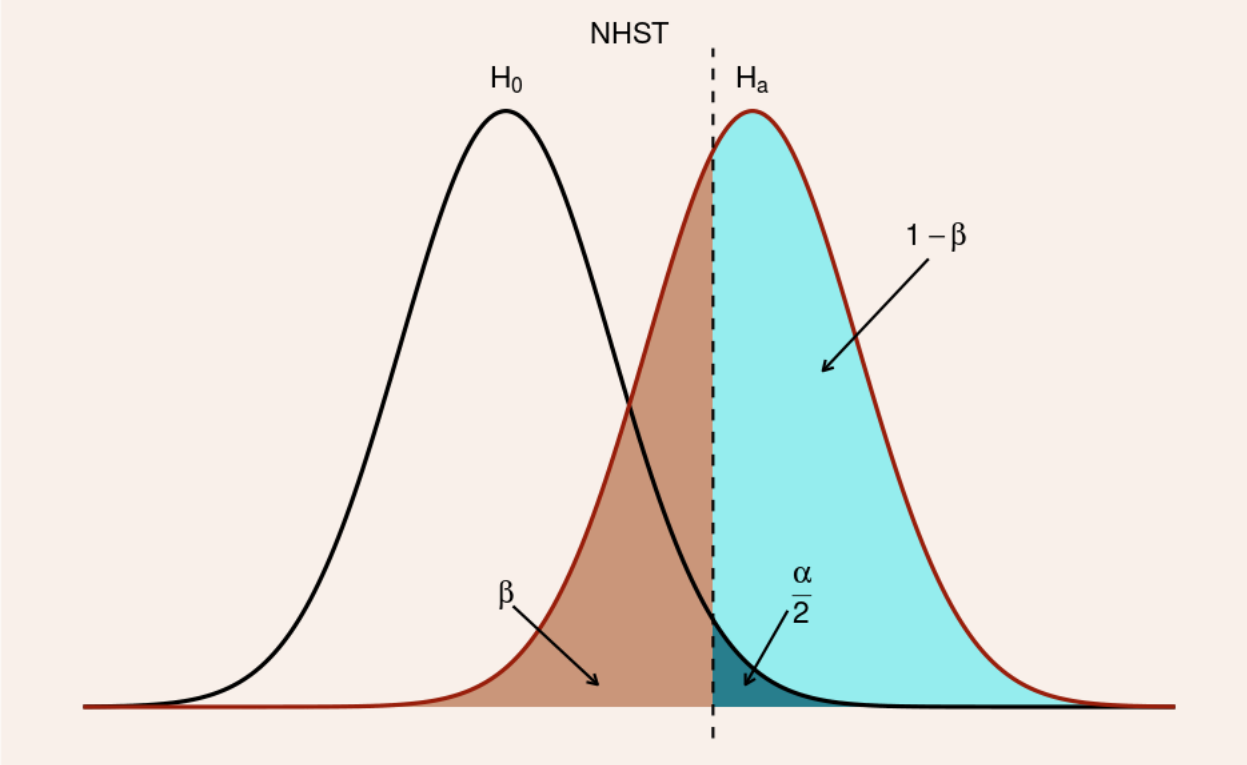


Figure 6.2:

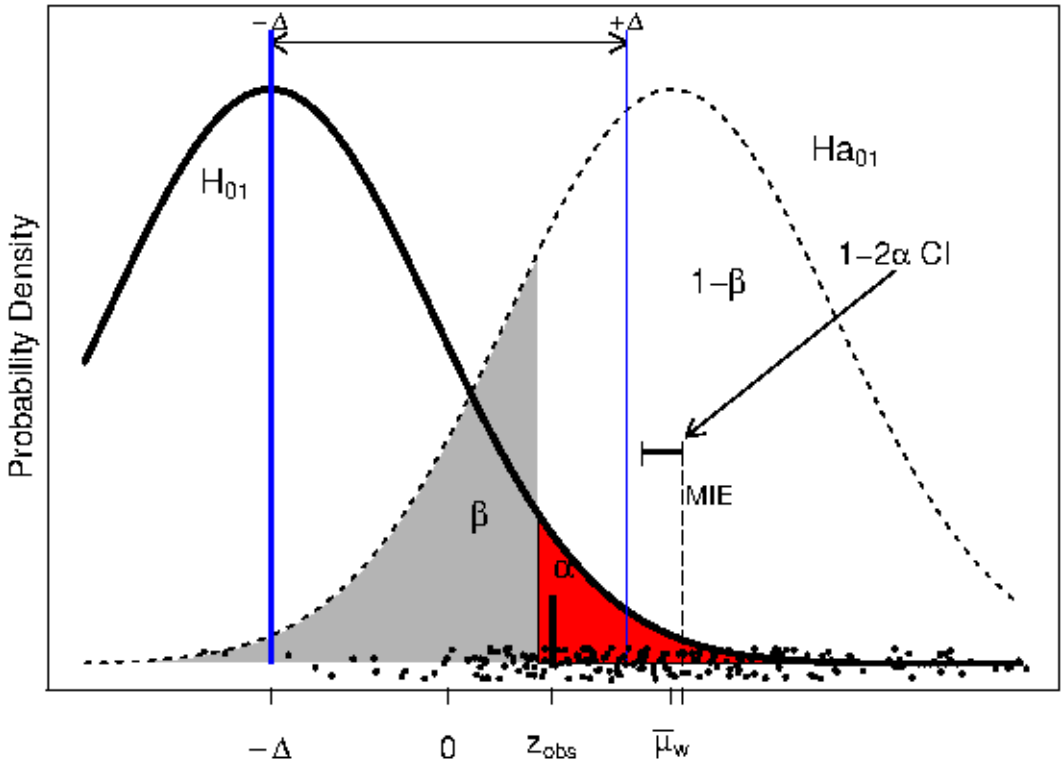


Figure 6.3:

```
## 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/

6.1 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

Figure 6.4:

Part IV

Basic Models

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

Chapter 7

Basics of Preprocessing

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)

7.1 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)

7.2 Missing values

Three types of problems are usually associated with MVs in DM [5]:

1. loss of efficiency
2. complications in handling and analyzing the data
3. bias resulting from differences between missing and complete data.

Imputation consists in replacing missing values for estimates of those missing values. Many algorithms do cannot handle missing values and imputation methods are needed.

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.

7.2.1 Imputation methods

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

7.3 Noise

Imperfections of the real-world data that influences negatively in the induced machine learning models.

Approaches to deal with noisy data: + 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.

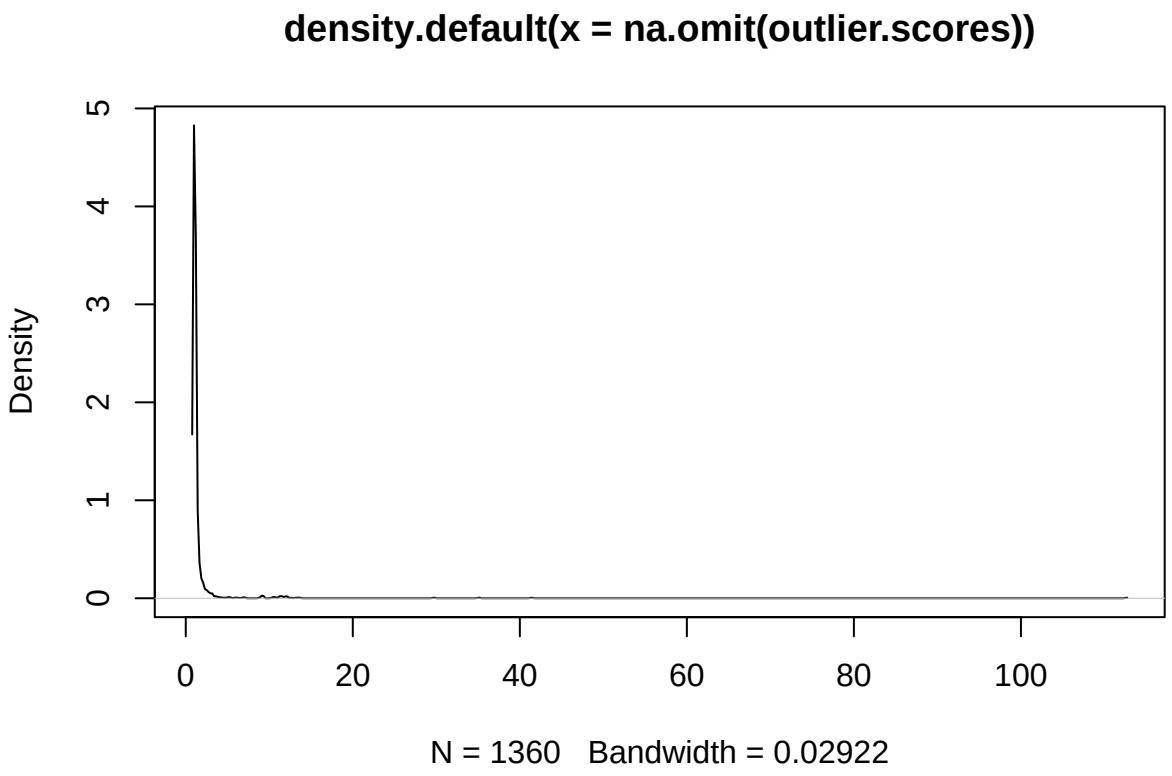
7.4 Outliers

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.

7.5 Feature selection (FS)

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 feature selection received a thorough treatment in pattern recognition and machine learning. Most of the feature selection algorithms tackle the task as a search problem, where each state in the search specifies a distinct subset of the possible attributes~**?**. 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~**?**.

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 a 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~**??**.

Feature subset algorithms search through candidate feature subsets guide by a certain evaluation measure~**?** 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 **?**, the correlation measure **?** and the estimated accuracy of a learning algorithm **?**.

- *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~**?** notes that feature selection algorithms that search through the space of feature subsets must address four main issues: the starting point of the search, the organization of the search, the evaluation of features subsets and the criterion used to terminate the search. Different algorithms address theses issues differently.

It is impractical to look at all possible feature subsets, even if the size is small. Feature selection algorithms usually proceed greedily. They can 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~**?** for a review).

7.6 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) [68] when the subset is used with a distance based method (kNN) and
- Training Set Selection (TSS) [21, 130] in which an actual model is learned.

It is also a search problem.

7.7 Discretization

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

-square

7.8 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
```

7.9 Normalization

7.9.1 Min-Max Normalization

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

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

7.9.2 Z-score normalization

TBD

7.10 Transformations

7.10.1 Linear Transformations and Quadratic Trans formations

TBD

7.10.2 Box-cox transformation

TBD

7.10.3 Nominal to Binary tranformations

TBD

Chapter 8

Preprocessing in R

8.1 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 × 22
##    LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT LOC_COMMENTS
##        <dbl>        <dbl>                <dbl>        <dbl>
## 1          6           21                    0           10
## 2          5           15                    0            2
## 3          2            5                    0            0
## 4          4            5                    0            2
## 5          2           11                    0            2
## 6          2           23                    0            3
## 7          1           11                    0            2
## 8          1           13                    0            2
## 9          2           17                    0            2
## 10         3            1                    0            0
## # ... with 241 more rows, and 18 more variables:
## #   CYCLOMATIC_COMPLEXITY <dbl>, 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 <fctr>
```

Another operator is `%in%`.

Select:

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

```
## # A tibble: 2,096 × 6
##    LOC_BLANK LOC_CODE_AND_COMMENT 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 variables: Defective <fctr>
```

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 × 6
##    LOC_BLANK LOC_CODE_AND_COMMENT 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 variables: Defective <fctr>
```

It is easier usign the chaining method:

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

```
## # A tibble: 2,096 × 6
##    LOC_BLANK LOC_CODE_AND_COMMENT 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 variables: Defective <fctr>
```

Arrange ascending

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

```
## # A tibble: 2,096 × 2
##    LOC_TOTAL Defective
##        <dbl>    <fctr>
## 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 × 2
##    LOC_TOTAL Defective
##        <dbl>    <fctr>
## 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 × 4
##    NUM_OPERANDS NUM_OPERATORS Defective HalsteadLength
##           <dbl>         <dbl>    <fctr>          <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 × 2
##   Defective avg_loc
##      <fctr>   <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)
```

```
## # A tibble: 2 × 7
##   Defective BRANCH_COUNT_mean LOC_TOTAL_mean BRANCH_COUNT_min
##      <fctr>             <dbl>          <dbl>            <dbl>
## 1         N              3.68           15.9                1
## 2         Y             10.12           44.7                1
## # ... with 3 more variables: LOC_TOTAL_min <dbl>, BRANCH_COUNT_max <dbl>,
## #   LOC_TOTAL_max <dbl>
```

It seems than Defective modules are larger than the non-Defective ones.

Count with:

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

```
## # A tibble: 2 × 2
##   Defective     n
##      <fctr> <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 × 22
##   LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT LOC_COMMENTS
##       <dbl>        <dbl>                <dbl>        <dbl>
## 1         0            5                    0            1
## 2         0            1                    0            0
## # ... with 18 more variables: CYCLOMATIC_COMPLEXITY <dbl>,
## #   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 <fctr>
```

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

```
## # A tibble: 105 × 22
##    LOC_BLANK BRANCH_COUNT LOC_CODE_AND_COMMENT LOC_COMMENTS
##        <dbl>        <dbl>                <dbl>        <dbl>
```

```
## 1          1            1                    0            0
## 2         14            5                    0            5
## 3          2            1                    0            0
## 4          2           15                    0            5
## 5          1            1                    0            0
## 6          0            1                    0            0
## 7          5            1                    0            0
## 8          5            1                    0            0
## 9          0            1                    0            0
## 10         2            7                    0            0
## # ... with 95 more rows, and 18 more variables:
## #   CYCLOMATIC_COMPLEXITY <dbl>, 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 <fctr>
```

```
# 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             <fctr> N, N, N, N, N, N, N, N, N, N, N, N, N, ...
```

8.2 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"
```

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	...	Attn	C
a_{11}	...	a1n	c1
a_{21}	...	a2n	c2
...	...	...	...
a_{m1}	...	amn	cm

Columns are usually called attributes and there is class attribute, which can be numeric or discrete. When the class is numeric, it is a regression problem. With discrete values, we talk about binary (two values) classification or multi-label classification

9.1 Regression

9.1.1 Linear Regression

- This procedure fits a straight line to the data. 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.
- 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.

9.2 Regression: Galton Data

Not related to Software Engineering but …

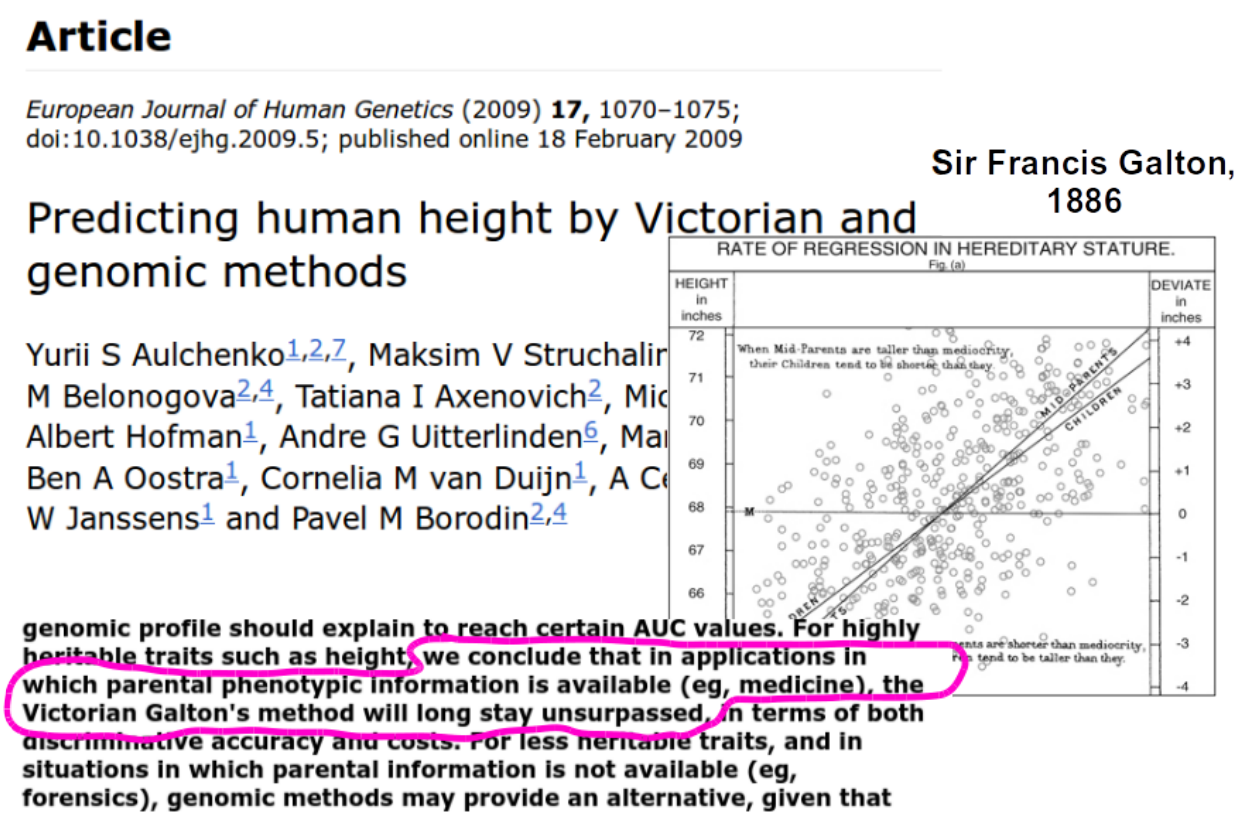


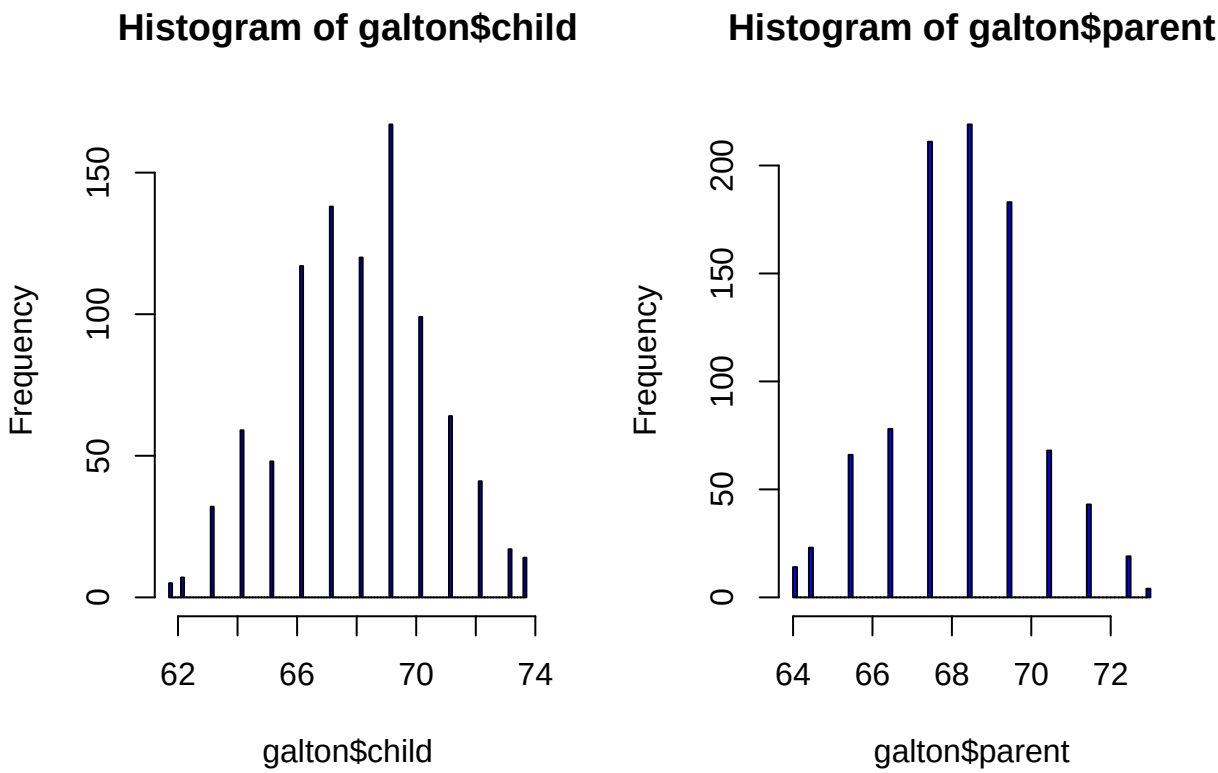
Figure 9.1: Galton Data

```
library(UsingR); data(galton)
```

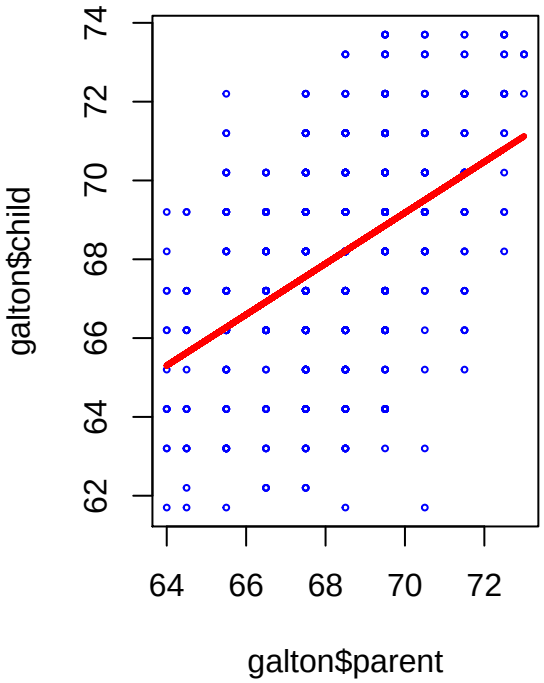
```
## Loading required package: MASS
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
## The following object is masked from 'package:sm':
##
##     muscle
## Loading required package: HistData
## Loading required package: Hmisc
## Loading required package: survival
##
## Attaching package: 'survival'
## The following object is masked from 'package:caret':
##
##     cluster
## Loading required package: Formula
```

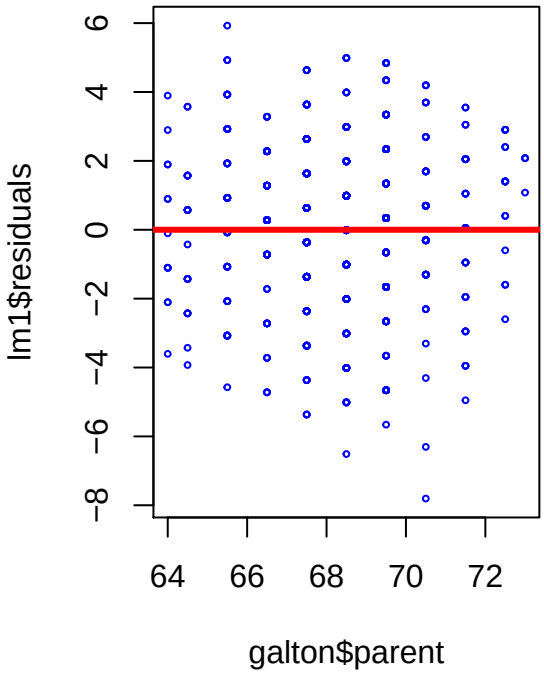
```
##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:dplyr':
##
##     combine, src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, round.POSIXt, trunc.POSIXt, units

##
## Attaching package: 'UsingR'

## The following object is masked _by_ '.GlobalEnv':
##
##     galton

## The following object is masked from 'package:survival':
##
##     cancer
```

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

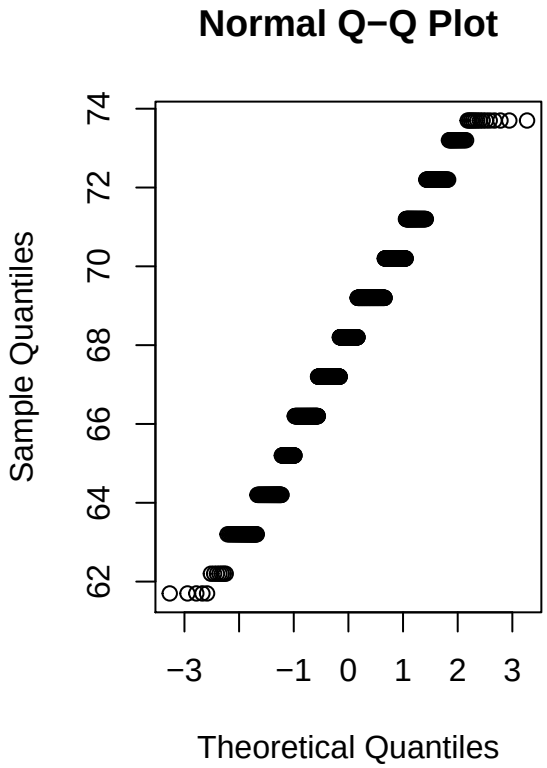



```
plot(galton$parent,galton$child,pch=1,col="blue", cex=0.4)
lm1 <- lm(galton$child ~ galton$parent)
lines(galton$parent,lm1$fitted,col="red",lwd=3)
plot(galton$parent,lm1$residuals,col="blue",pch=1, cex=0.4)
abline(c(0,0),col="red",lwd=3)
```






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



9.3 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 value*s: there should be no obvious trend.
- *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.

9.4 Linear regression

9.4.1 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.

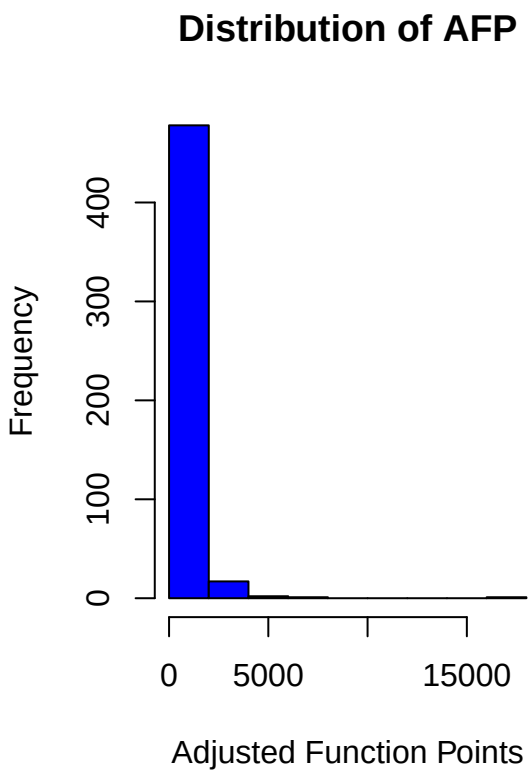
```
library(foreign)
china <- read.arff("./datasets/effortEstimation/china.arff")
china_size <- china$AFP
summary(china_size)
```

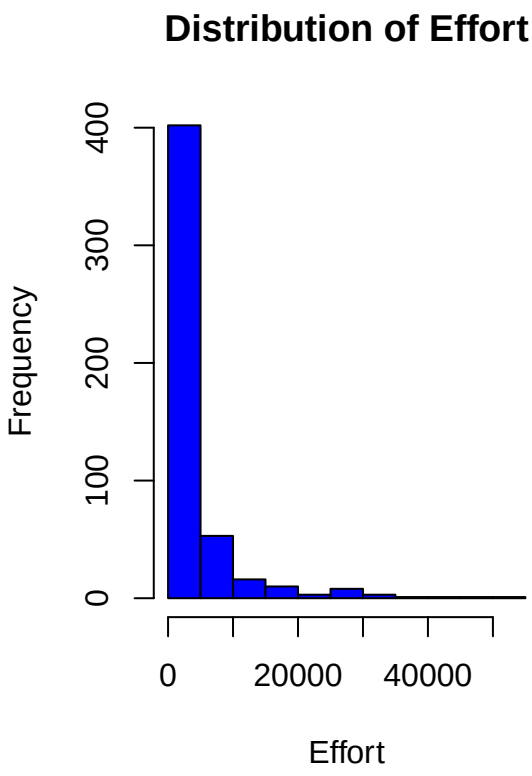
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       9     100     215     487     438   17500
```

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

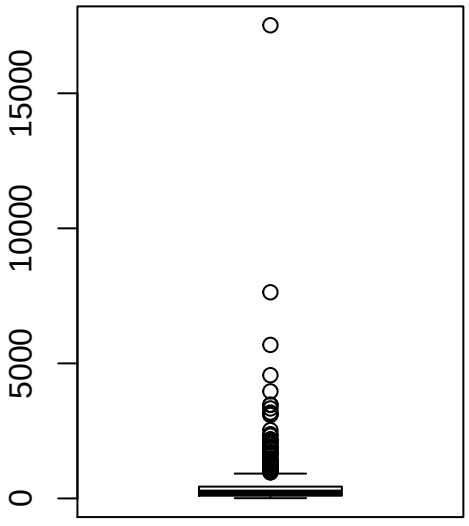
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      26     704    1830    3920    3830   54600
```

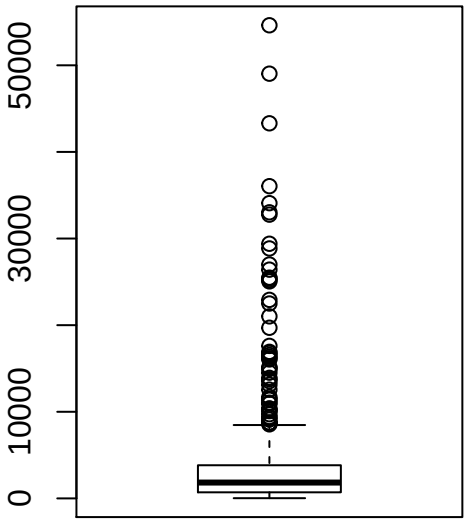
```
par(mfrow=c(1,2))
hist(china_size, col="blue", xlab="Adjusted Function Points", main="Distribution of AFP")
hist(china_effort, col="blue",xlab="Effort", main="Distribution of Effort")
```



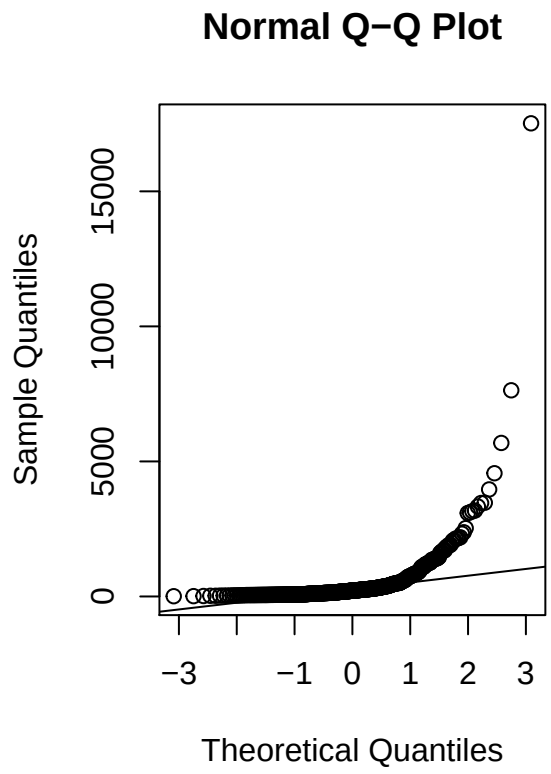


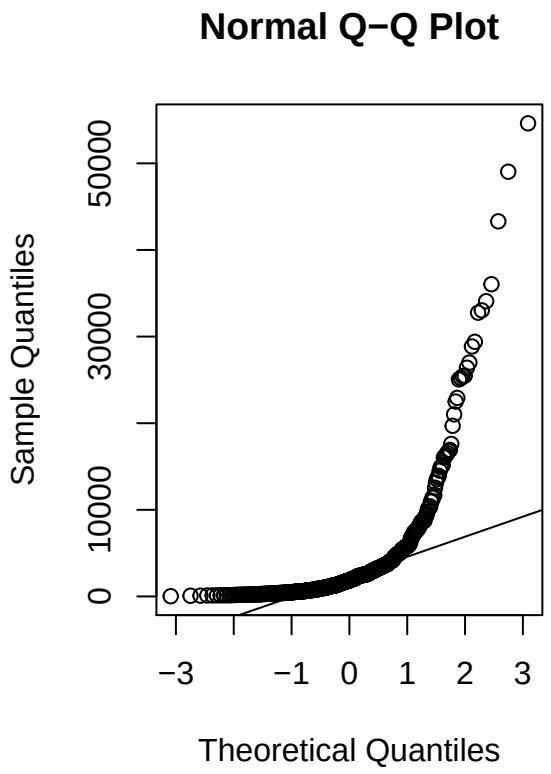

```
boxplot(china_size)
boxplot(china_effort)
```

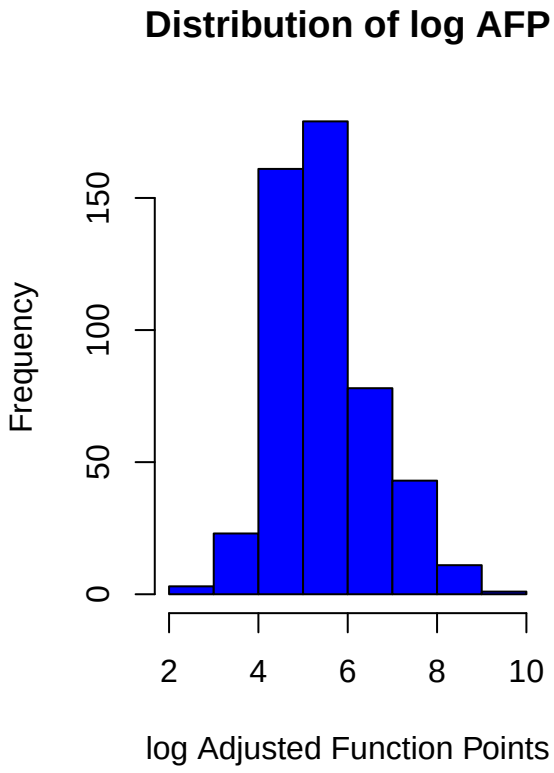


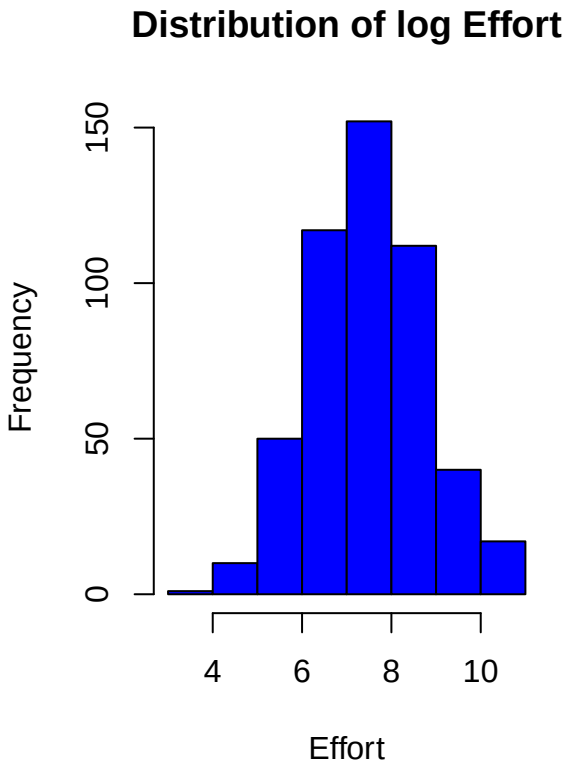


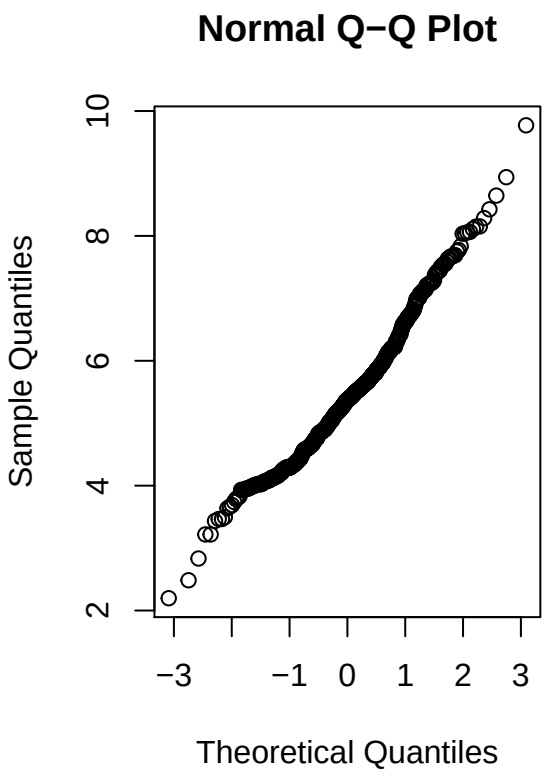

```
qqnorm(china_size)
qqline(china_size)
qqnorm(china_effort)
qqline(china_effort)
```

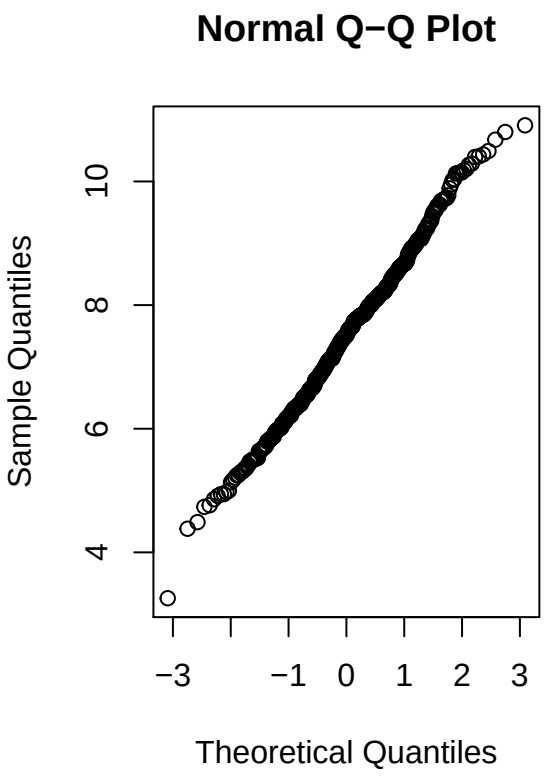

Normal Q−Q Plot
Sample Quantiles
0
5000
10000
15000
−3
−1
0
1
2
3
Theoretical Quantiles


Normal Q−Q Plot
Sample Quantiles
0
10000
30000
50000
−3
−1
0
1
2
3
Theoretical Quantiles

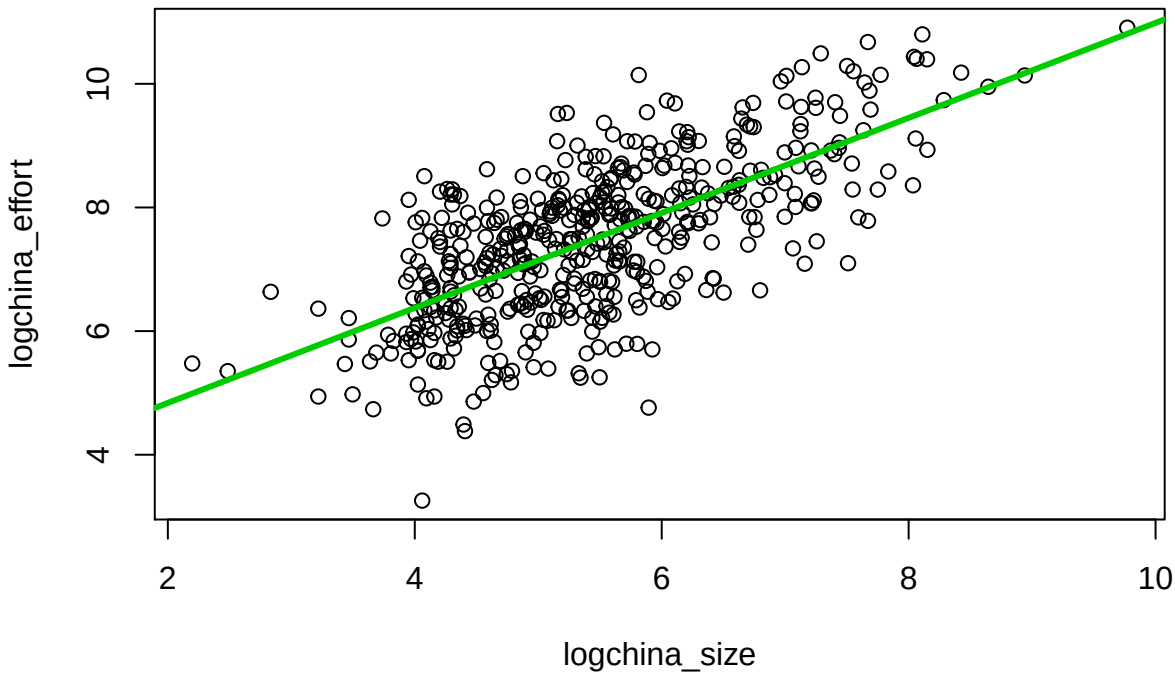

Distribution of log AFP
Frequency
0
50
100
150
2
4
6
8
10
log Adjusted Function Points


Distribution of log Effort
Frequency
0
50
100
150
4
6
8
10
Effort

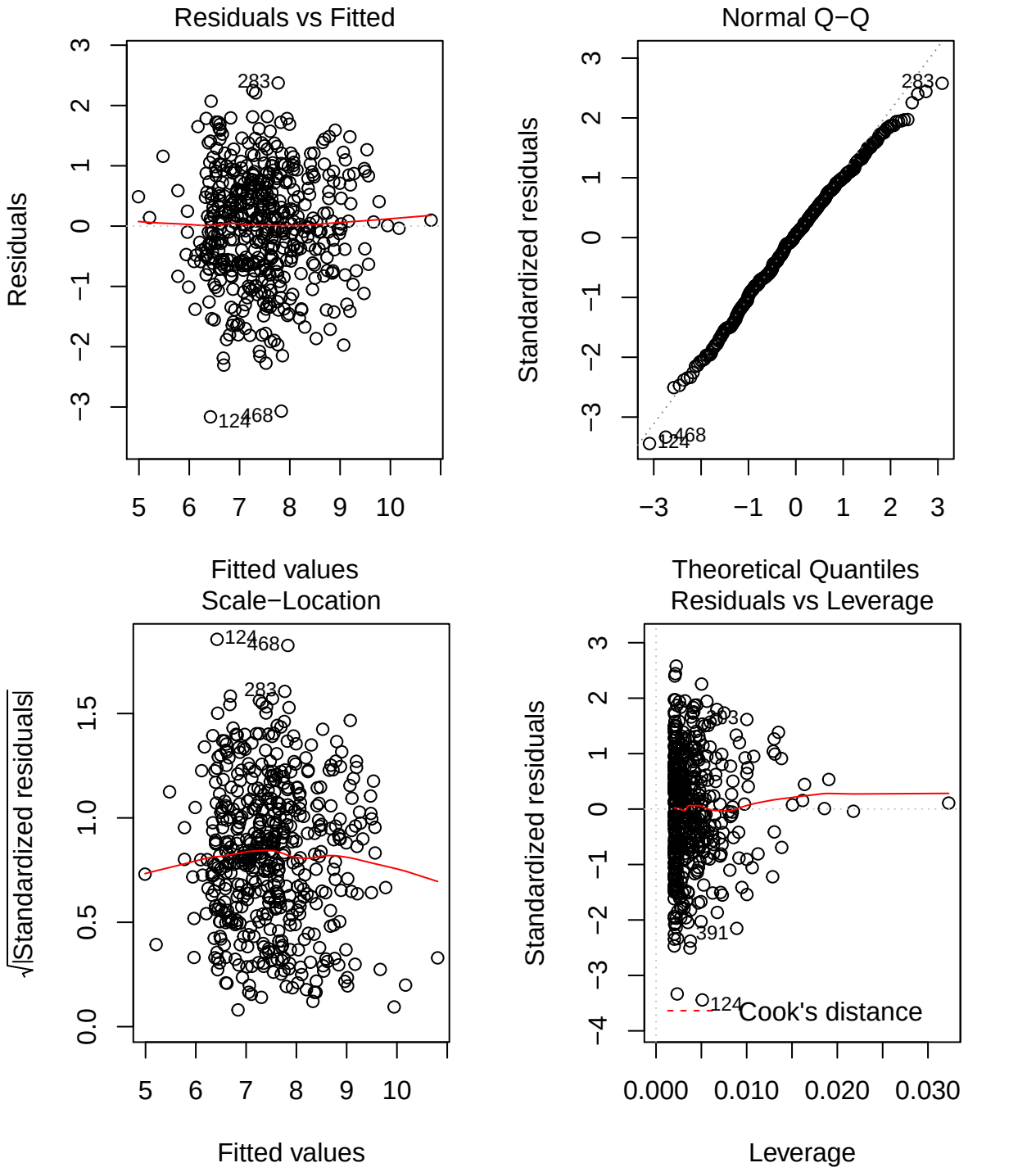





```
linmodel_logchina <- lm(logchina_effort ~ logchina_size)
par(mfrow=c(1,1))
plot(logchina_size, logchina_effort)
abline(linmodel_logchina, lwd=3, col=3)
```




```
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
```

9.5 Supervised Classification

Here we will use defect prediction as ensample of several machine learning techniques.

> **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

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

9.5.1 The caret package

The caret (Classification And REgression Training) package 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

9.5.2 Defect Prediction as a running example

We will show the use of different classification techniques in the problem of defect prediction.

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.

For example, using one of the NASA datasets used extensively in defect prediction:

```
library(caret)
library(foreign)

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

Then we need to divide the data into training and testing.

```
# Split data into training and test datasets
set.seed(1)
inTrain <- createDataPartition(y=kc1$Defective,p=.75,list=FALSE)
kc1.train <- kc1[inTrain,]
kc1.test <- kc1[-inTrain,]
```

Another approach to dividing the data:

```
# Split data into training and test datasets

set.seed(1)
ind <- sample(2, nrow(kc1), replace = TRUE, prob = c(0.75, 0.25))
kc1.train <- kc1[ind==1, ]
kc1.test <- kc1[ind==2, ]
```

9.6 Linear Discriminant Analysis (LDA)

One classical approach to classification is Linear Discriminant Analysis (LDA). And the basic all would be as follows.

```
ldaModel <- train (Defective ~ ., data=kc1.train, method="lda", preProc=c("center","scale"))

ldaModel
```

```
## Linear Discriminant Analysis
##
## 1573 samples
##   21 predictors
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1573, 1573, 1573, 1573, 1573, 1573, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.855     0.286
##
##
```

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=kc1.train, method="lda", trControl=ctrl, preProc=c("center","sca

ldaModel
```

```
## Linear Discriminant Analysis
##
## 1573 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: 1416, 1416, 1415, 1416, 1415, 1416, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.854     0.288
##
##
```

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=kc1.train, method="lda", trControl=ctrl, preProc=c("center

ldaModel3xcv10
```

```
## Linear Discriminant Analysis
##
## 1573 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: 1416, 1416, 1416, 1416, 1416, 1415, ...
## Resampling results:
##
##   ROC    Sens   Spec
##   0.789  0.962  0.26
##
##
```

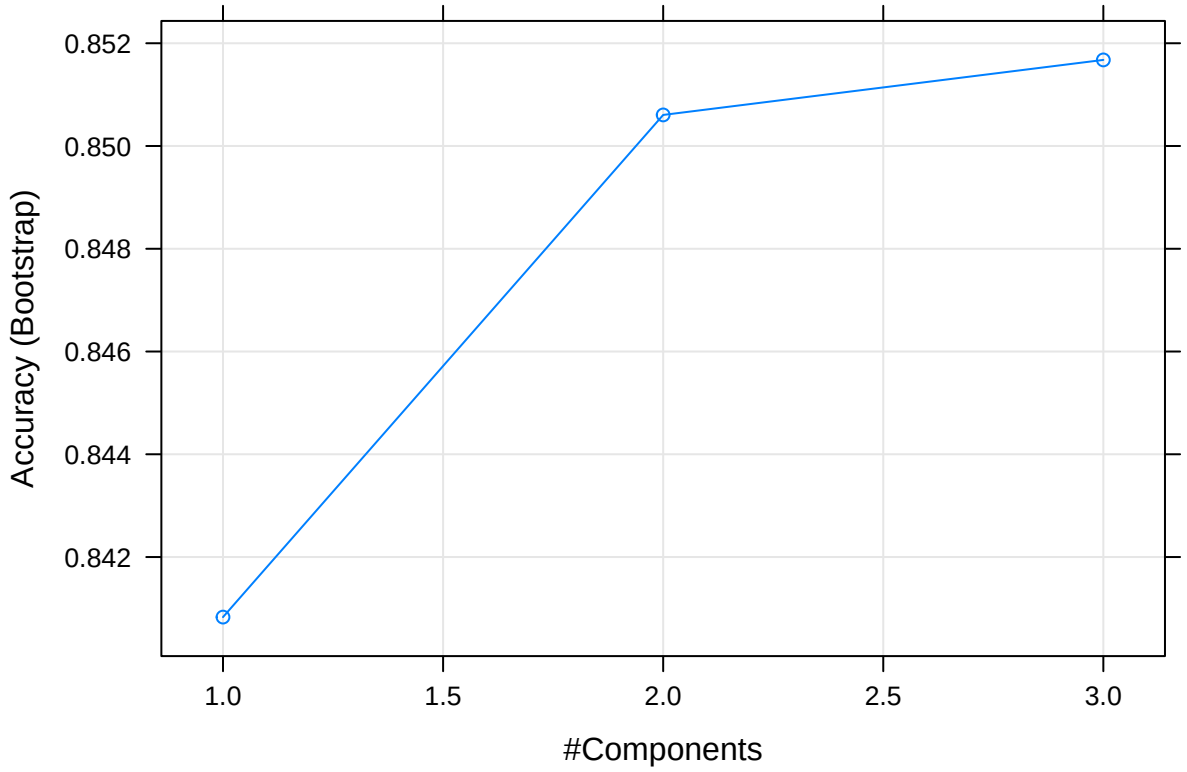
Most methods have parameters that need to be optimised and that is one of the

```
plsFit3x10cv <- train (Defective ~ ., data=kc1.train, method="pls", trControl=trainControl(classProbs=T

plsFit3x10cv
```

```
## Partial Least Squares
##
```

```
## 1573 samples
##   21 predictors
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1573, 1573, 1573, 1573, 1573, 1573, ...
## Resampling results across tuning parameters:
##
##   ncomp  Accuracy  Kappa
##   1      0.841     0.112
##   2      0.851     0.166
##   3      0.852     0.191
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was ncomp = 3.
```

```
plot(plsFit3x10cv)
```



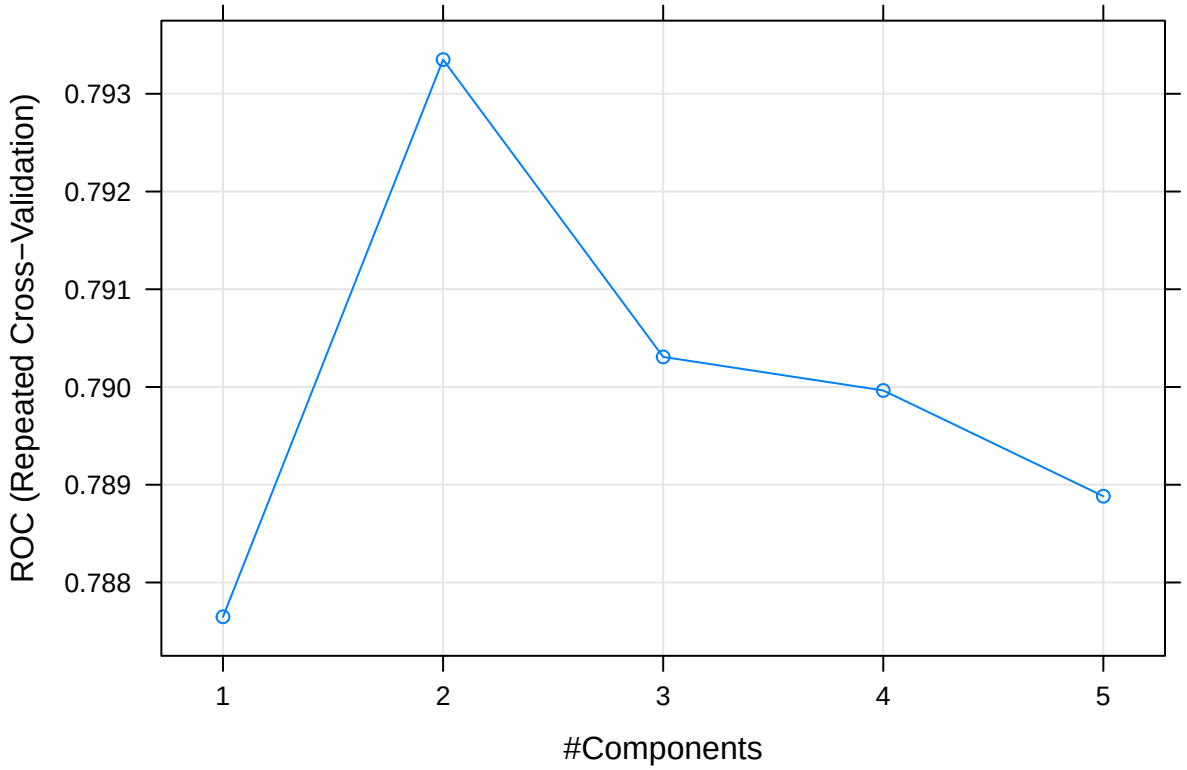
The parameter `tuneLength` allow us to specify the number values per parameter to consider.

```
plsFit3x10cv <- train (Defective ~ ., data=kc1.train, method="pls", trControl=ctrl, metric="ROC", tuneL

plsFit3x10cv
```

```
## Partial Least Squares
##
## 1573 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: 1415, 1416, 1417, 1415, 1416, 1416, ...
## Resampling results across tuning parameters:
##
##   ncomp  ROC    Sens   Spec
##   1      0.788  0.981  0.0929
##   2      0.793  0.984  0.1311
##   3      0.790  0.982  0.1517
##   4      0.790  0.986  0.1626
##   5      0.789  0.985  0.1596
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was ncomp = 2.
```

```
plot(plsFit3x10cv)
```



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

```
plsProbs <- predict(plsFit3x10cv, newdata = kc1.test, type = "prob")
```

```
plsClasses <- predict(plsFit3x10cv, newdata = kc1.test, type = "raw")
confusionMatrix(data=plsClasses,kc1.test$Defective)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N 439  69
##          Y   3  12
##
```

```
##                Accuracy : 0.862
##                  95% CI : (0.83, 0.891)
##     No Information Rate : 0.845
##     P-Value [Acc > NIR] : 0.152
##
##                   Kappa : 0.212
##  Mcnemar's Test P-Value : 1.85e-14
##
##             Sensitivity : 0.993
##             Specificity : 0.148
##          Pos Pred Value : 0.864
##          Neg Pred Value : 0.800
##              Prevalence : 0.845
##          Detection Rate : 0.839
##    Detection Prevalence : 0.971
##       Balanced Accuracy : 0.571
##
##        'Positive' Class : N
##
```

9.6.1 Predicting the number of defects (numerical class)

From the Bug Predictiono Repository 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: 718, 718, 718, 718, 719, 718, ...
## Resampling results:
##
##   RMSE   Rsquared
##   0.841  0.386
##
##
```

Others such as Elasticnet:

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

```
## Loading required package: glmnet
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-5
```

```
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, 718, 718, 718, 718, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda  RMSE   Rsquared
##   0.10   0.0012  0.813  0.341
##   0.10   0.0120  0.818  0.334
##   0.10   0.1202  0.808  0.340
##   0.55   0.0012  0.812  0.341
##   0.55   0.0120  0.823  0.327
##   0.55   0.1202  0.812  0.347
##   1.00   0.0012  0.812  0.341
##   1.00   0.0120  0.819  0.331
##   1.00   0.1202  0.817  0.345
##
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.12.
```

9.6.2 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
## -3.573  -0.613  -0.538  -0.497   2.099
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.08638    0.13462  -15.50  < 2e-16 ***
## cbo          0.05646    0.00705    8.01  1.1e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 831.84  on 797  degrees of freedom
## Residual deviance: 725.93  on 796  degrees of freedom
## AIC: 729.9
##
## Number of Fisher Scoring iterations: 5
```

Predict a single point:

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

```
##     1
## 0.128
```

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

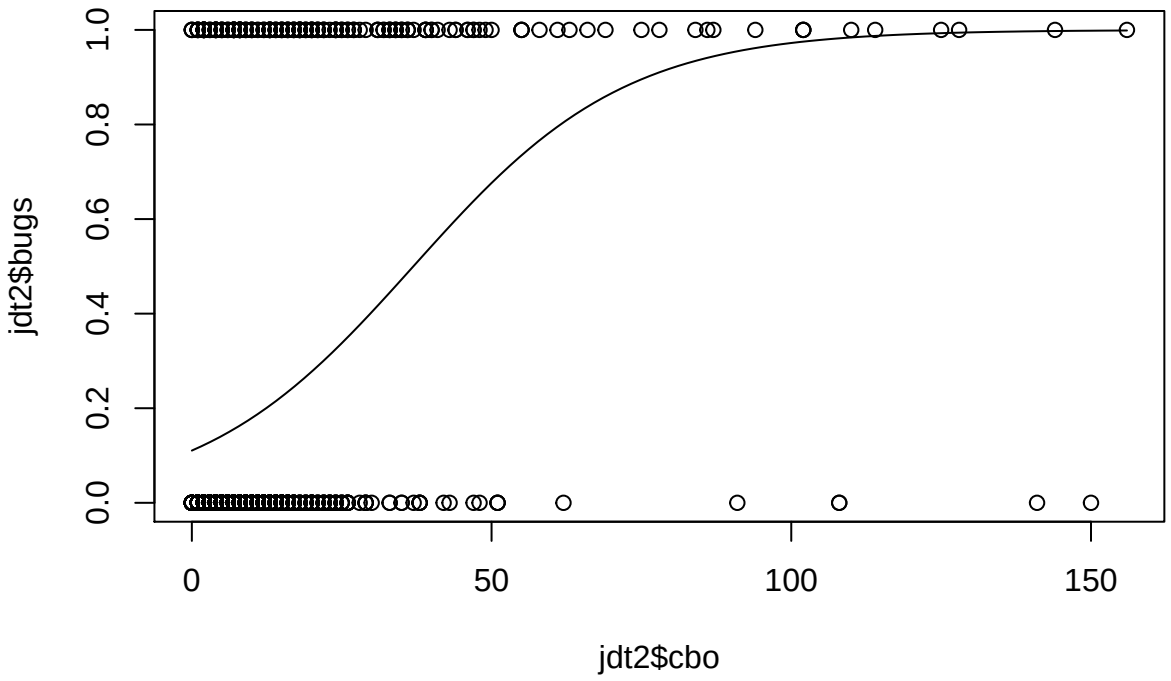
```
results <- predict(glmLogit, jdtTest, type = "response")

range(jdtTrain$cbo)
```

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

```
range(results)
```

```
## [1] 0.110 0.984
```

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




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

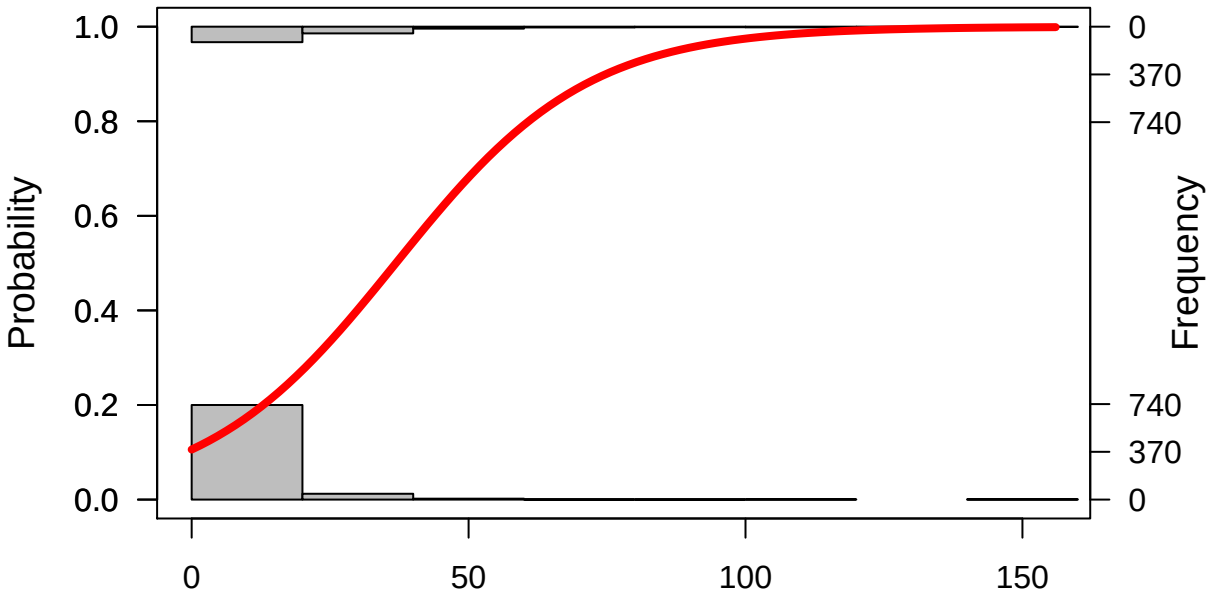
Another type of graph:

```
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")
```



9.7 Classification Trees

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

```
# Build a decision tree
library(party)

kc2 <- read.arff("./datasets/defectPred/D1/MC1.arff")
str(kc2)
```

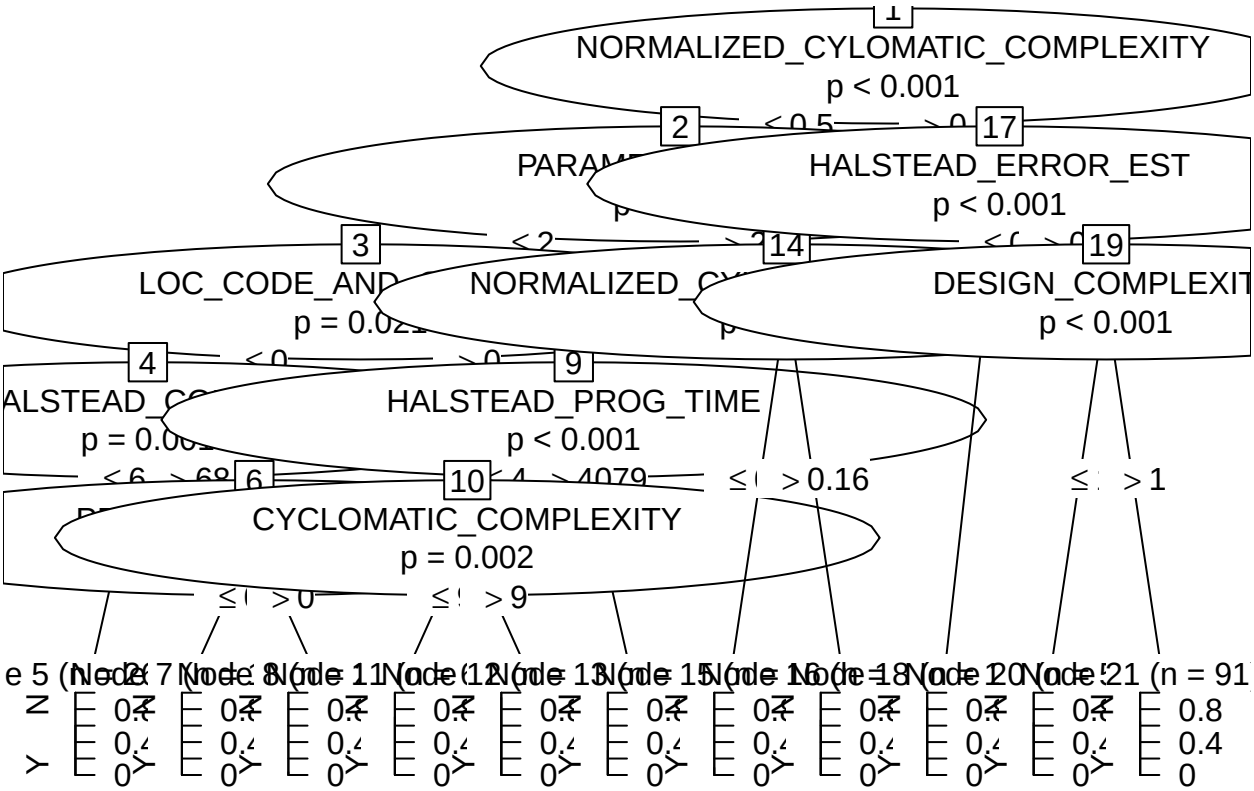
```
## 'data.frame':    9277 obs. of  39 variables:
##  $ LOC_BLANK                     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ BRANCH_COUNT                  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ CALL_PAIRS                    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ LOC_CODE_AND_COMMENT          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ LOC_COMMENTS                  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CONDITION_COUNT               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CYCLOMATIC_COMPLEXITY         : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ CYCLOMATIC_DENSITY            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ DECISION_COUNT                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ DESIGN_COMPLEXITY             : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ DESIGN_DENSITY                : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ EDGE_COUNT                    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ ESSENTIAL_COMPLEXITY          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ ESSENTIAL_DENSITY             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ LOC_EXECUTABLE                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ PARAMETER_COUNT               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ GLOBAL_DATA_COMPLEXITY        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ GLOBAL_DATA_DENSITY           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HALSTEAD_CONTENT              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HALSTEAD_DIFFICULTY           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HALSTEAD_EFFORT               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HALSTEAD_ERROR_EST            : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ HALSTEAD_LENGTH                : num  1 1 0 0 1 1 0 0 1 1 ...
##  $ HALSTEAD_LEVEL                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HALSTEAD_PROG_TIME             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HALSTEAD_VOLUME                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ MAINTENANCE_SEVERITY           : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ MODIFIED_CONDITION_COUNT       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ MULTIPLE_CONDITION_COUNT       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ NODE_COUNT                     : num  2 2 2 2 2 2 2 2 2 2 ...
##  $ NORMALIZED_CYLOMATIC_COMPLEXITY: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ NUM_OPERANDS                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ NUM_OPERATORS                  : num  1 1 0 0 1 1 0 0 1 1 ...
##  $ NUM_UNIQUE_OPERANDS            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ NUM_UNIQUE_OPERATORS           : num  1 1 0 0 1 1 0 0 1 1 ...
##  $ NUMBER_OF_LINES                : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ PERCENT_COMMENTS               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ LOC_TOTAL                      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Defective                      : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
```

```
set.seed(1)
inTrain <- createDataPartition(y=kc2$Defective,p=.60,list=FALSE)
kc2.train <- kc2[inTrain,]
kc2.test <- kc2[-inTrain,]

kc2.formula <- kc2$Defective ~ .
kc2.ctree <- ctree(kc2.formula, data = kc2.train)

# predict on test data
pred <- predict(kc2.ctree, newdata = kc2.test)
# check prediction result
table(pred, kc2.test$Defective)
```

```
##
## pred    N    Y
##    N 3683   27
##    Y    0    0
```

```
plot(kc2.ctree)
```



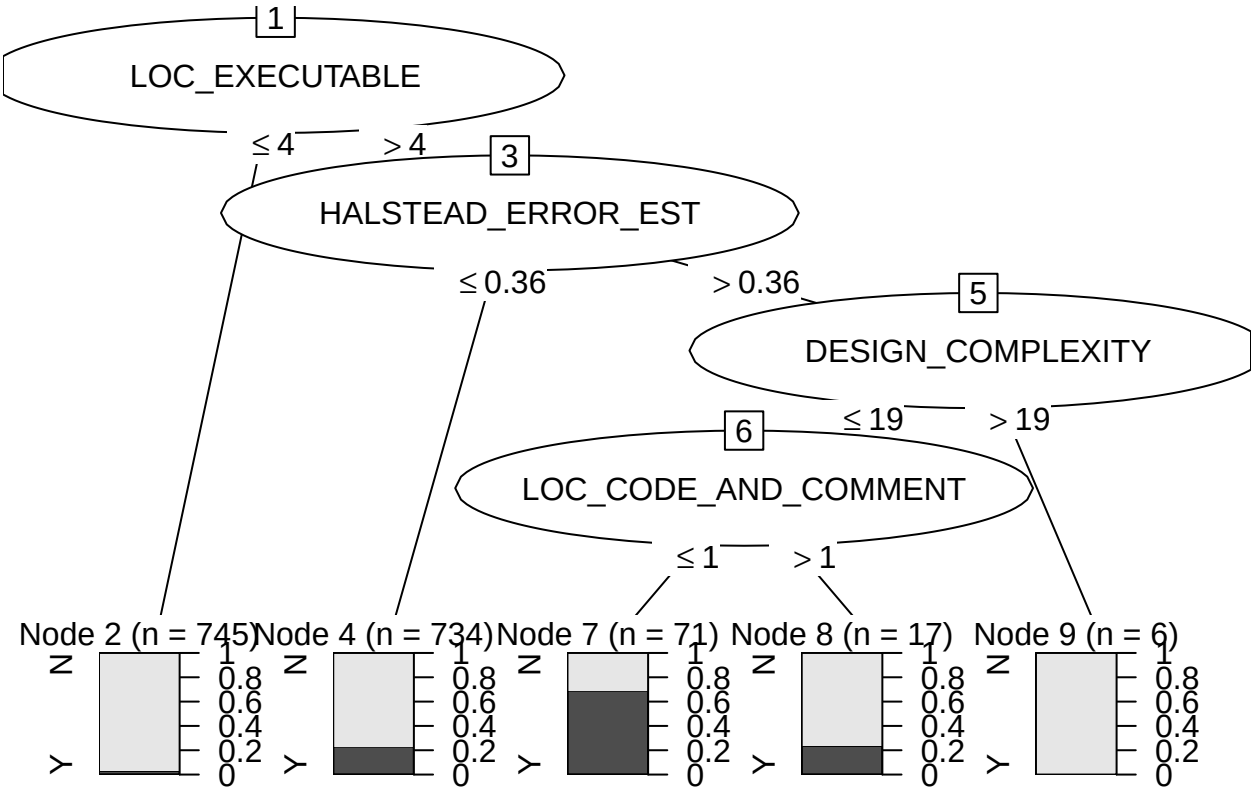
Using the C50, there are two ways, specifying train and testing

```
library(C50)
c50t <- C5.0(kc1.train[,-ncol(kc1.train)], kc1.train[,ncol(kc1.train)])
summary(c50t)
```

```
##
## Call:
## C5.0.default(x = kc1.train[, -ncol(kc1.train)], y =
##  kc1.train[, ncol(kc1.train)])
##
##
## C5.0 [Release 2.07 GPL Edition]      Mon Feb 13 19:42:15 2017
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 1573 cases (22 attributes) from undefined.data
##
## Decision tree:
##
## LOC_EXECUTABLE <= 4: N (745/22)
## LOC_EXECUTABLE > 4:
## :...HALSTEAD_ERROR_EST <= 0.36: N (734/169)
##     HALSTEAD_ERROR_EST > 0.36:
##     :...DESIGN_COMPLEXITY > 19: N (6)
##         DESIGN_COMPLEXITY <= 19:
##         :...LOC_CODE_AND_COMMENT <= 1: Y (71/22)
##             LOC_CODE_AND_COMMENT > 1: N (17/4)
```

```
##
##
## Evaluation on training data (1573 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##       5  217(13.8%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##    1307    22    (a): class N
##     195    49    (b): class Y
##
##
##  Attribute usage:
##
##  100.00% LOC_EXECUTABLE
##   52.64% HALSTEAD_ERROR_EST
##    5.98% DESIGN_COMPLEXITY
##    5.59% LOC_CODE_AND_COMMENT
##
##
## Time: 0.0 secs
```

```
plot(c50t)
```



```
c50tPred <- predict(c50t, kc1.train)
table(c50tPred, kc1.train$Defective)
```

```
##
## c50tPred    N    Y
##        N 1307  195
##        Y   22   49
```

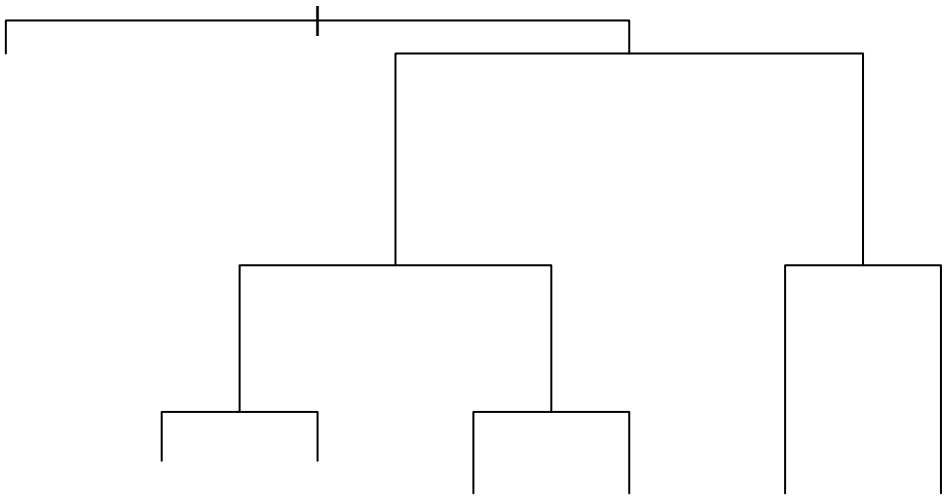
or using the formula approach:

```
# Using the formula notation
c50t2 <- C5.0(Defective ~ ., kc1.train)
c50tPred2 <- predict(c50t2, kc1.train)
table(c50tPred2, kc1.train$Defective)
```

```
##
## c50tPred2    N    Y
##         N 1307  195
##         Y   22   49
```

Using the 'rpart' package

```
# Using the 'rpart' package
library(rpart)
kc1.rpart <- rpart(Defective ~ ., data=kc1.train)
plot(kc1.rpart)
```



```
library(rpart.plot)
#asRules(kc1.rpart)
#fancyRpartPlot(kc1.rpart)
```

9.8 Rules

C5 Rules

```
library(C50)
c50r <- C5.0(kc1.train[,-ncol(kc1.train)], kc1.train[,ncol(kc1.train)], rules = TRUE)
summary(c50r)
```

```
##
## Call:
## C5.0.default(x = kc1.train[, -ncol(kc1.train)], y =
##  kc1.train[, ncol(kc1.train)], rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Mon Feb 13 19:42:17 2017
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 1573 cases (22 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (1479/191, lift 1.0)
##  HALSTEAD_ERROR_EST <= 0.36
##  ->  class N  [0.870]
##
## Rule 2: (94/41, lift 3.6)
##  HALSTEAD_ERROR_EST > 0.36
##  ->  class Y  [0.563]
##
## Default class: N
##
##
## Evaluation on training data (1573 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##       2  232(14.7%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##    1288    41    (a): class N
##     191    53    (b): class Y
##
##
##  Attribute usage:
##
##  100.00% HALSTEAD_ERROR_EST
##
##
## Time: 0.0 secs
```

```
c50rPred <- predict(c50r, kc1.train)
table(c50rPred, kc1.train$Defective)
```

```
##
## c50rPred    N    Y
##        N 1288  191
##        Y   41   53
```

9.9 Distanced-based Methods

IB1 and IB-k

```
library(class)

ind <- sample(2, nrow(iris), replace=T, prob=c(0.7, 0.3))
kc1.train <- kc1[ind==1, ]
kc1.test <- kc1[ind==2, ]

m1 <- knn(train=kc1.train[,-22], test=kc1.test[,-22], cl=kc1.train[,22], k=3)

table(kc1.test[,22],m1)
```

```
##    m1
##       N   Y
##   N 455  24
##   Y  64  15
```

9.10 Probabilistic Methods

9.10.1 Naive Bayes

Using the `klaR` package with `caret`:

```
library(caret)
library(klaR)
model <- NaiveBayes(Defective ~ ., data = kc1.train)
predictions <- predict(model, kc1.test[,-22])
confusionMatrix(predictions$class, kc1.test$Defective)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   N   Y
##          N 442  53
##          Y  37  26
##
##                Accuracy : 0.839
##                  95% CI : (0.806, 0.868)
##     No Information Rate : 0.858
##     P-Value [Acc > NIR] : 0.917
##
##                   Kappa : 0.275
##  Mcnemar's Test P-Value : 0.114
##
##             Sensitivity : 0.923
##             Specificity : 0.329
##          Pos Pred Value : 0.893
##          Neg Pred Value : 0.413
##              Prevalence : 0.858
##          Detection Rate : 0.792
##    Detection Prevalence : 0.887
```

```
##       Balanced Accuracy : 0.626
##
##        'Positive' Class : N
##
```

Using the `e1071` package:

```
library (e1071)
n1 <-naiveBayes(kc1.train$Defective ~ ., data=kc1.train)

# Show first 3 results using 'class'
head(predict(n1,kc1.test, type = c("class")),3) # class by default
```

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

```
# Show first 3 results using 'raw'
head(predict(n1,kc1.test, type = c("raw")),3)
```

```
##      N        Y
## [1,] 1 2.40e-09
## [2,] 1 5.76e-09
## [3,] 1 5.76e-09
```

9.10.2 Bayesian Networks

To Do

Chapter 10

Unsupervised Classification

From the predictive (unsupervised) point of view, patterns are found to predict future behaviour or estimate. This include association rules, clustering, or tree clustering which purpose is to join 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	...	Attn
a_{11}	...	a1n
a_{21}	...	a2n
...	...	...
a_{m1}	...	amn

10.1 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)
```

kMeans

```
#library(reshape, quietly=TRUE)
#kc1.kmeans <- kmeans(sapply(na.omit(kc1.train), rescaler, "range"), 10)
```

10.2 Association rules

```
library(arules)
```

```
##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:modeltools':
##
##     info
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

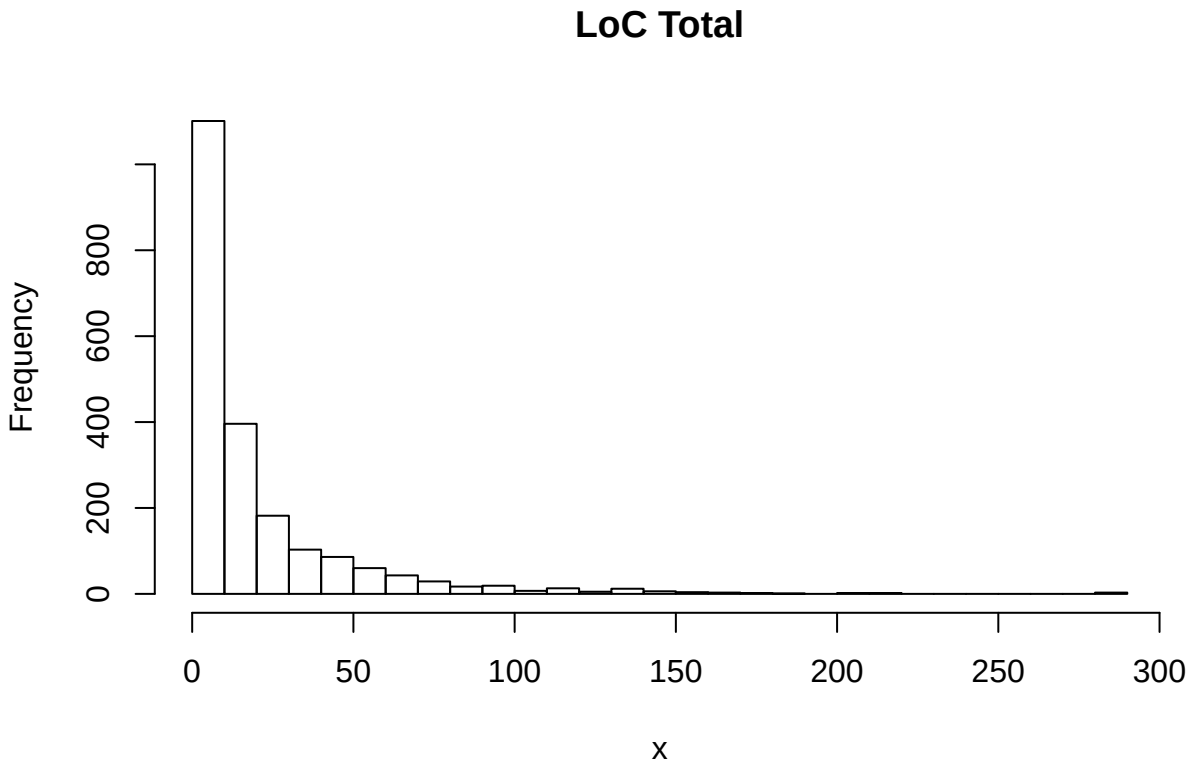
```
x <- as.numeric(kc1$LOC_TOTAL)
str(x)
```

```
##  num [1:2096] 5 3 3 3 12 5 3 3 3 13 ...
```

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0     3.0     9.0    20.4    24.0   288.0
```

```
hist(x, breaks=30, main="LoC Total")
```




```
 xDisc <- discretize(x, categories=5)
# table(xDisc)
```

```
 for(i in 1:21) kc1[,i] <- discretize(kc1[,i],  "frequency", categories=5)

str(kc1)
```

```
## 'data.frame':    2096 obs. of  22 variables:
##  $ LOC_BLANK           : Factor w/ 4 levels "0","1","[2, 4)",..: 1 1 1 1 3 1 1 1 1 3 ...
##  $ BRANCH_COUNT        : Factor w/ 4 levels "1","3","[4, 8)",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ LOC_CODE_AND_COMMENT : Factor w/ 2 levels "0","[1,12]": 1 1 1 1 1 1 1 1 1 1 ...
##  $ LOC_COMMENTS         : Factor w/ 3 levels "0","1","[2,44]": 1 1 1 1 1 1 1 1 1 1 ...
##  $ CYCLOMATIC_COMPLEXITY: Factor w/ 4 levels "1","2","[3, 5)",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ DESIGN_COMPLEXITY    : Factor w/ 3 levels "1","[2, 4)","[4,45]": 1 1 1 1 1 1 1 1 1 1 ...
##  $ ESSENTIAL_COMPLEXITY : Factor w/ 2 levels "1","[3,26]": 1 1 1 1 1 1 1 1 1 1 ...
##  $ LOC_EXECUTABLE       : Factor w/ 5 levels " 0","[ 1,  3)",..: 3 2 2 2 3 3 2 2 2 3 ...
##  $ HALSTEAD_CONTENT     : Factor w/ 5 levels "[ 0.00,  5.80)",..: 3 1 1 1 3 3 1 1 1 4 ...
##  $ HALSTEAD_DIFFICULTY  : Factor w/ 5 levels "[ 0.00, 1.57)",..: 3 1 1 1 3 3 1 1 1 3 ...
##  $ HALSTEAD_EFFORT      : Factor w/ 5 levels "[  0.0,   12.2)",..: 3 1 1 1 3 3 1 1 1 3 ...
##  $ HALSTEAD_ERROR_EST   : Factor w/ 5 levels "0.00","0.01",..: 2 1 1 1 3 2 1 1 1 3 ...
##  $ HALSTEAD_LENGTH      : Factor w/ 5 levels "[ 0,  5)","[ 5,  10)",..: 3 1 1 1 3 3 1 1 1 4 ...
##  $ HALSTEAD_LEVEL       : Factor w/ 5 levels "[0.00,0.08)",..: 4 1 1 1 3 4 1 1 1 3 ...
##  $ HALSTEAD_PROG_TIME   : Factor w/ 5 levels "[ 0.00,   0.68)",..: 3 1 1 1 3 3 1 1 1 3 ...
##  $ HALSTEAD_VOLUME      : Factor w/ 5 levels "[  0.0,  10.0)",..: 3 1 1 1 3 3 1 1 1 4 ...
##  $ NUM_OPERANDS         : Factor w/ 5 levels "[ 0,  2)","[ 2,  4)",..: 3 1 1 1 3 3 1 1 1 3 ...
##  $ NUM_OPERATORS        : Factor w/ 5 levels "[ 0,  4)","[ 4,  7)",..: 3 1 1 1 3 3 1 1 1 4 ...
##  $ NUM_UNIQUE_OPERANDS  : Factor w/ 5 levels "[ 0,  2)","[ 2,  4)",..: 2 1 1 1 3 2 1 1 1 4 ...
##  $ NUM_UNIQUE_OPERATORS : Factor w/ 5 levels "[ 0, 4)","[ 4, 6)",..: 2 1 1 1 2 2 1 1 1 3 ...
##  $ LOC_TOTAL            : Factor w/ 5 levels "[ 1,  3)","[ 3,  6)",..: 2 2 2 2 3 2 2 2 2 3 ...
##  $ Defective            : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
```

```
rules <- apriori(kc1, parameter = list(support=0.60, confidence=0.800, minlen=3))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.6      3
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 1257
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[94 item(s), 2096 transaction(s)] done [0.00s].
## sorting and recoding items ... [5 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [18 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
rules
```

```
## set of 18 rules
```

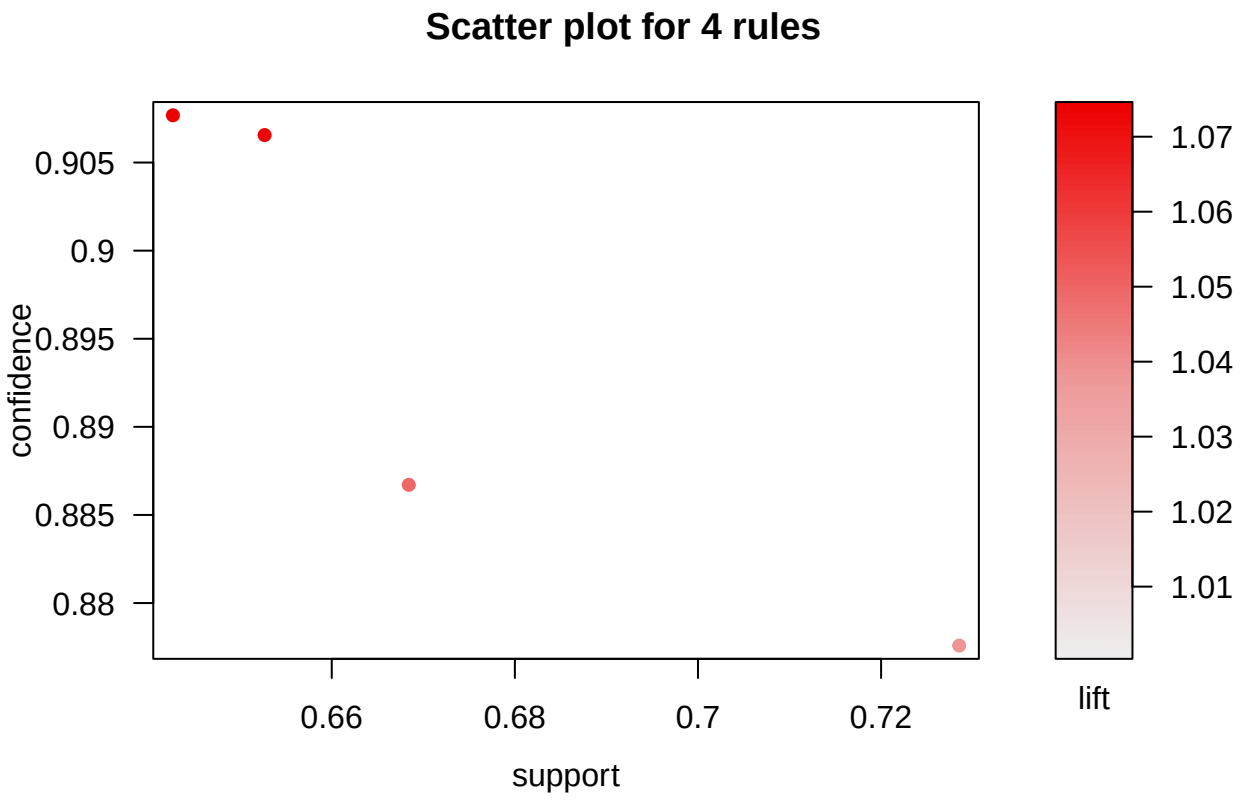
```
rules <- apriori(kc1,
   parameter = list(minlen=3, supp=0.6, conf=0.8),
   appearance = list(rhs=c("Defective=Y", "Defective=N"),
   default="lhs"),
   control = list(verbose=F))

 #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
## [1] {LOC_COMMENTS=0,
##      ESSENTIAL_COMPLEXITY=1} => {Defective=N}   0.653      0.907 1.07
## [2] {LOC_CODE_AND_COMMENT=0,
##      LOC_COMMENTS=0}         => {Defective=N}   0.668      0.887 1.05
## [3] {LOC_CODE_AND_COMMENT=0,
##      ESSENTIAL_COMPLEXITY=1} => {Defective=N}   0.729      0.878 1.04
## [4] {LOC_CODE_AND_COMMENT=0,
##      LOC_COMMENTS=0,
##      ESSENTIAL_COMPLEXITY=1} => {Defective=N}   0.643      0.908 1.07
```

```
 rules
```

```
## set of 4 rules
```

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



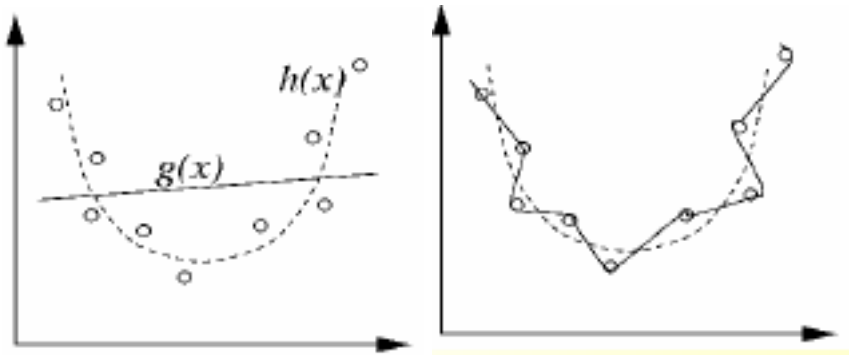
Chapter 11

Evaluation of Models

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

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

11.1 Underfitting vs. Overfitting



For example, increasing the tree size, decreases the training and testing errors. However, at some point after (tree complexity), training error keeps decreasing but testing error increases. Many algorithms have parameters to determine the model complexity (e.g., in decision trees is the prunning parameter)

11.2 Building and Validating a Model

11.2.1 Holdout approach

Holdout approach consists of dividing the dataset into *training* (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).

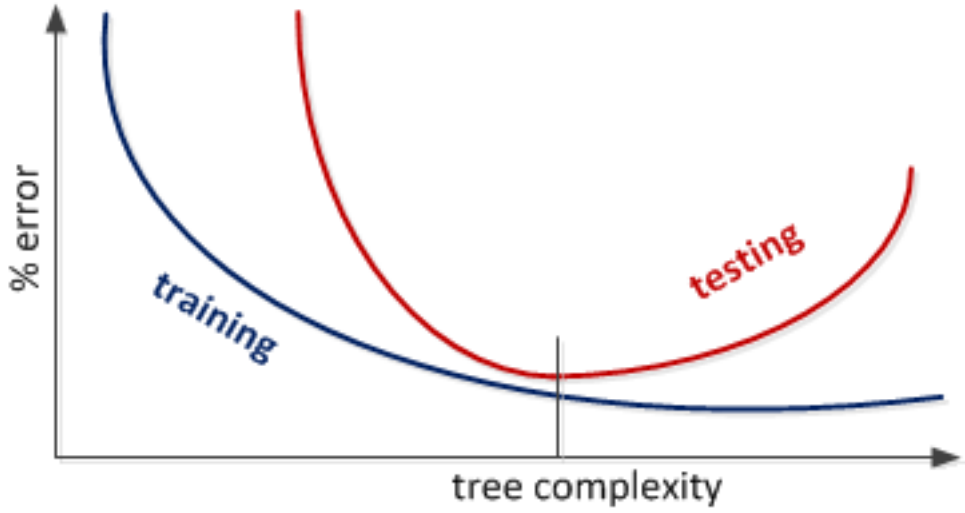


Figure 11.1: Overfitting in trees

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.

Figure 11.2:

11.3 Cross Validation (CV)

- *k-Fold Cross-Validation*: it involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, the the methods is fit on the remaining k-1 folds. This procedure is repeated k times. If k is equal to n we are in the previous method.
- 1st step: split dataset ($\mathcal{D}$) 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

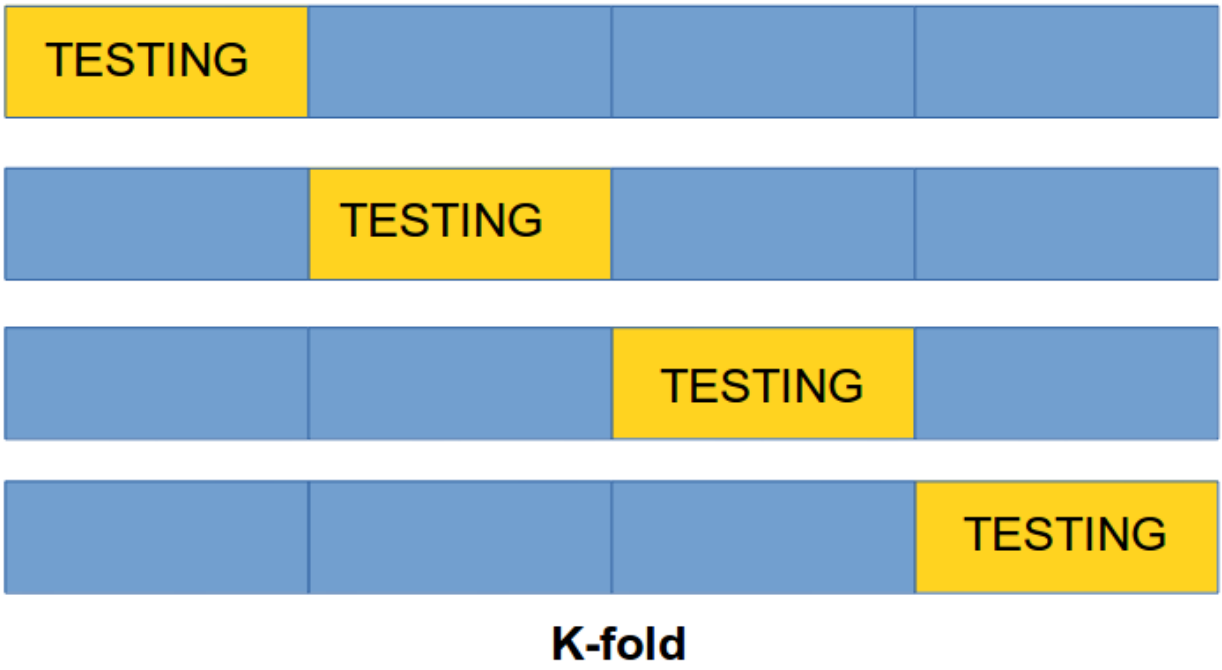


Figure 11.3: k-fold

- *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.

11.3.1 China dataset. Split data into Training and Testing

- The data is already divided into two different files

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

```
## [1] 332
```

```
logchina_size <- log(chinaTrain$AFP)
logchina_effort <- log(chinaTrain$Effort)
linmodel_logchina_train <- lm(logchina_effort ~ logchina_size)
par(mfrow=c(1,1))
plot(logchina_size, logchina_effort)
abline(linmodel_logchina_train, lwd=3, col=4)
```

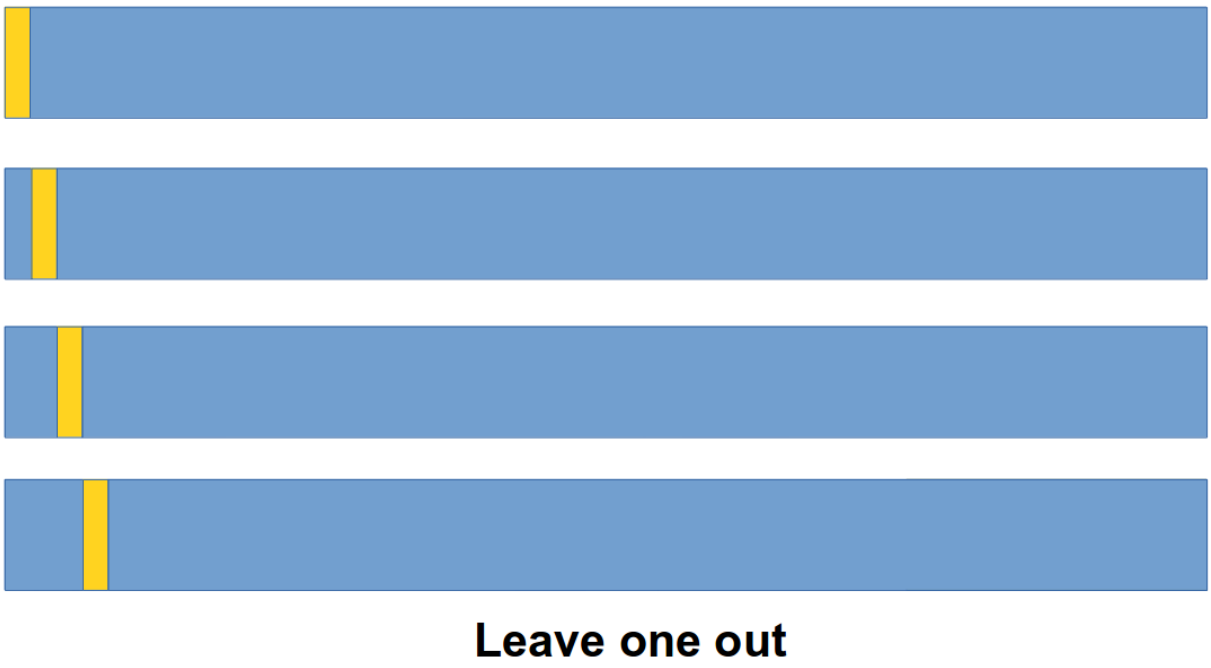
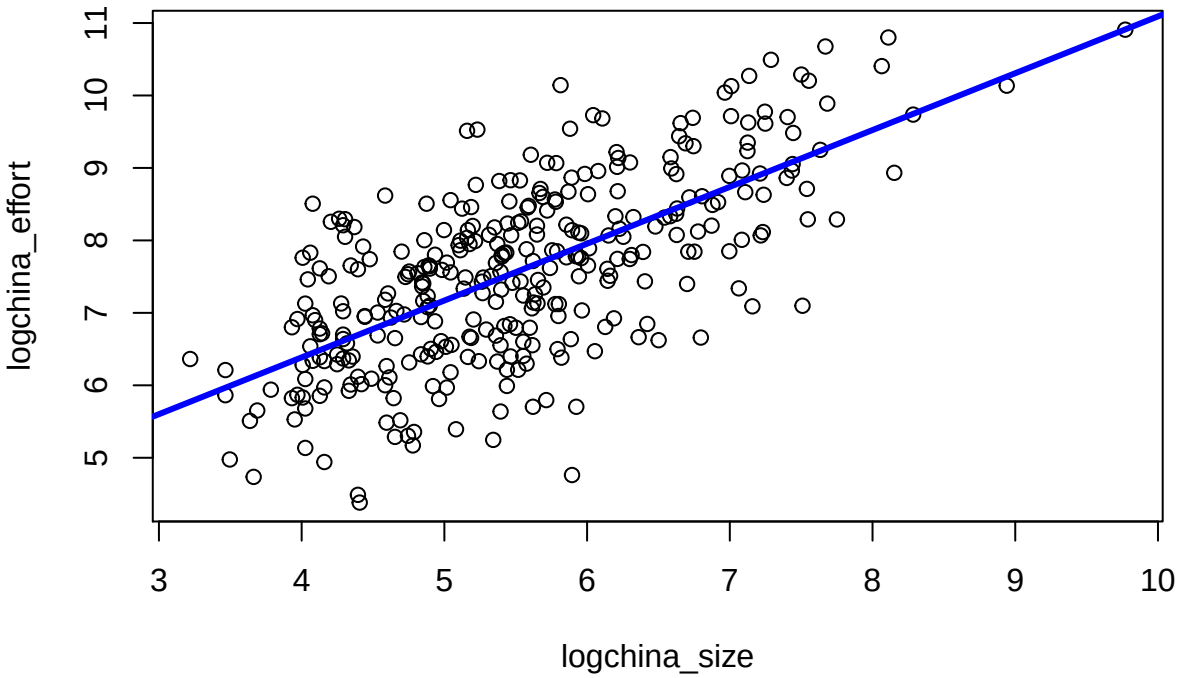
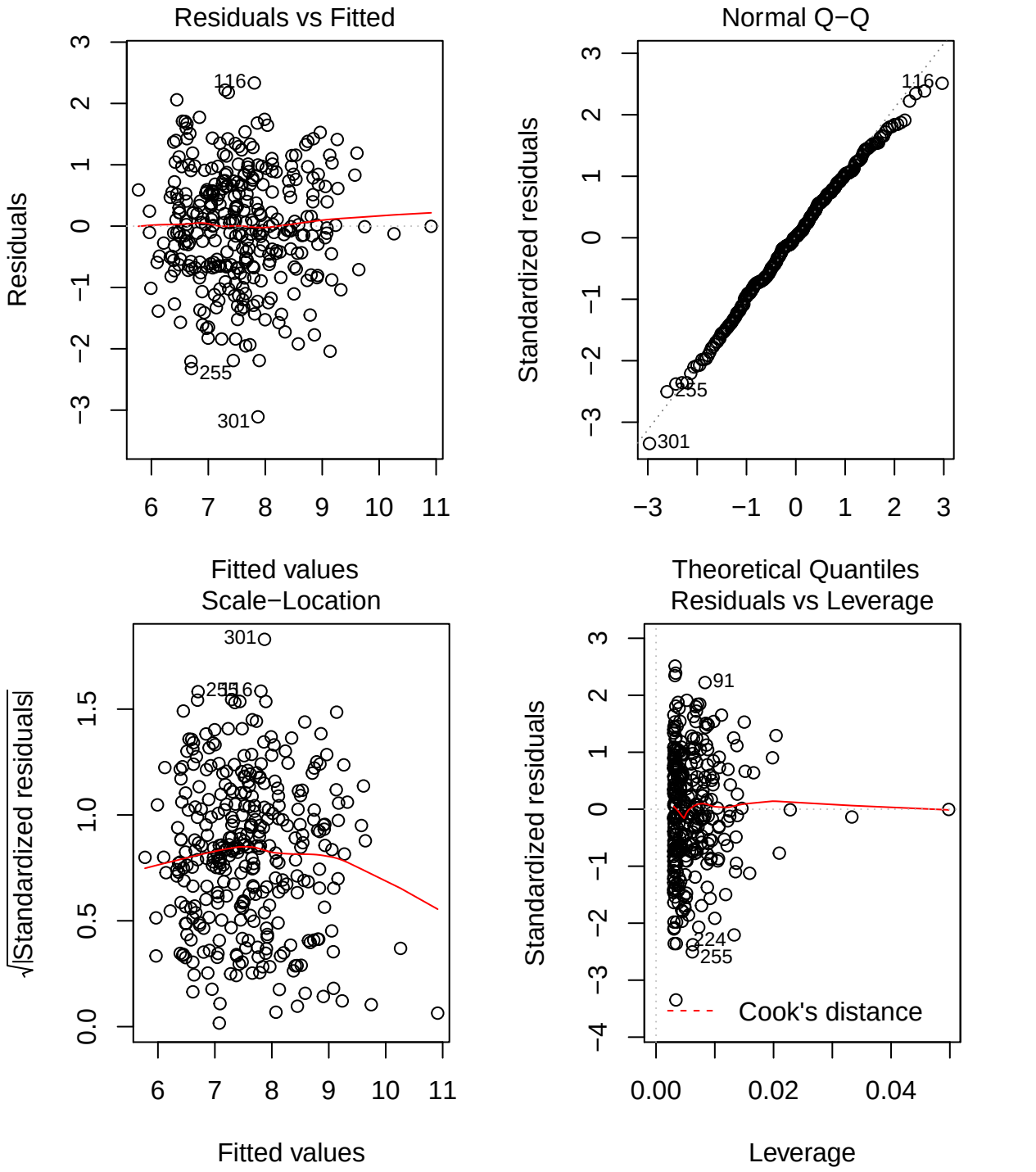


Figure 11.4: LOO




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




```
linmodel_logchina_train
```

```
##
## Call:
## lm(formula = logchina_effort ~ logchina_size)
##
## Coefficients:
##   (Intercept)  logchina_size
##         3.249          0.784
```

11.4 Evaluation of Classifiers

11.4.1 Discrete Evaluation

The confusion matrix (which can be extended to multiclass problems). The following table shows the possible outcomes for binary classification problems:

	$PredPos$	$PredNeg$
$ActPos$	TP	FN
$ActNeg$	FP	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).

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*:

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

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

f-measure

G-mean

$\sqrt{PD \times Precision}$

G-mean2

$\sqrt{PD \times Specificity}$

j-coeff = sensitivity + specificity - 1 = PD -PF

(Jiang, Cubic and Ma, 2008 ESE)

11.4.2 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):

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

11.4.3 Graphical Evaluation

ROC

Precision Recall

Another evaluation technique to consider when data is imbalanced is the *Receiver Operating Characteristic* (*ROC*)~**?** curve which provides a graphical visualisation of the results.

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

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

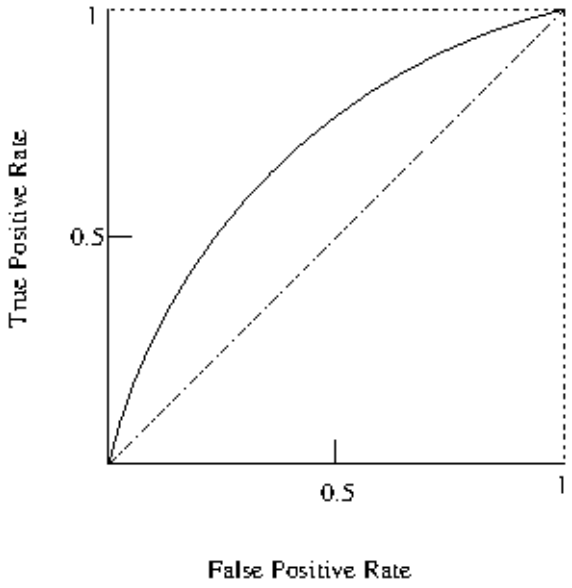


Figure 11.5: Receiver Operating Characteristic

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. %NB: I gave an example with two classes, but in fact the weighted average make sense only when you have more then two classes. When you have only two classes weighting does not make sense, and the measures should be computed relative to the minority class. In other words, you are interested to know if you are able to detect the minority

11.4.4 Metrics used in Software Engineering and Defect 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~**?**.

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.

11.4.5 Numeric Prediction Evaluation

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

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

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

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.

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 $\frac{(p_1-a_1)^2+...+(p_n-a_n)^2}{(a_1-\hat{a})^2+...+(a_n-\hat{a})^2}$ ($\hat{a}$ 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 defined 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 12

Measures of Evaluation used in Software Engineering

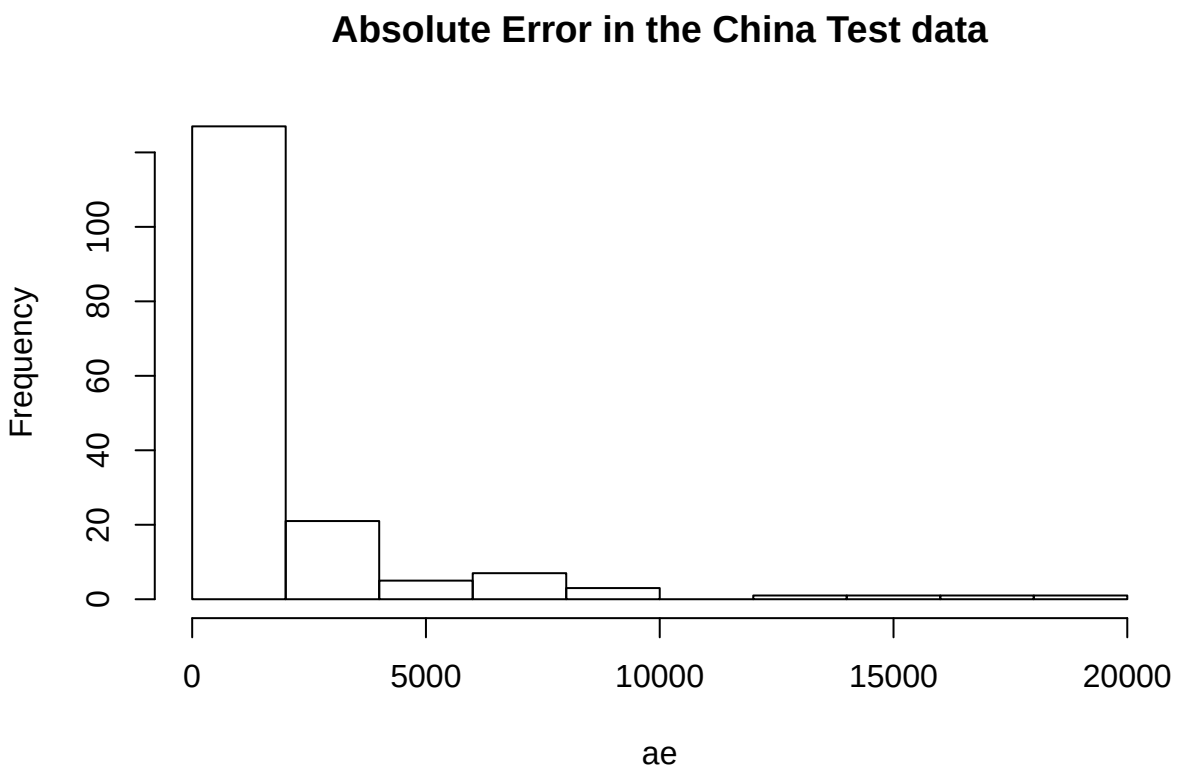
There are several measures usually used:

- 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* $r \neq t$*.*

12.1 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)
```

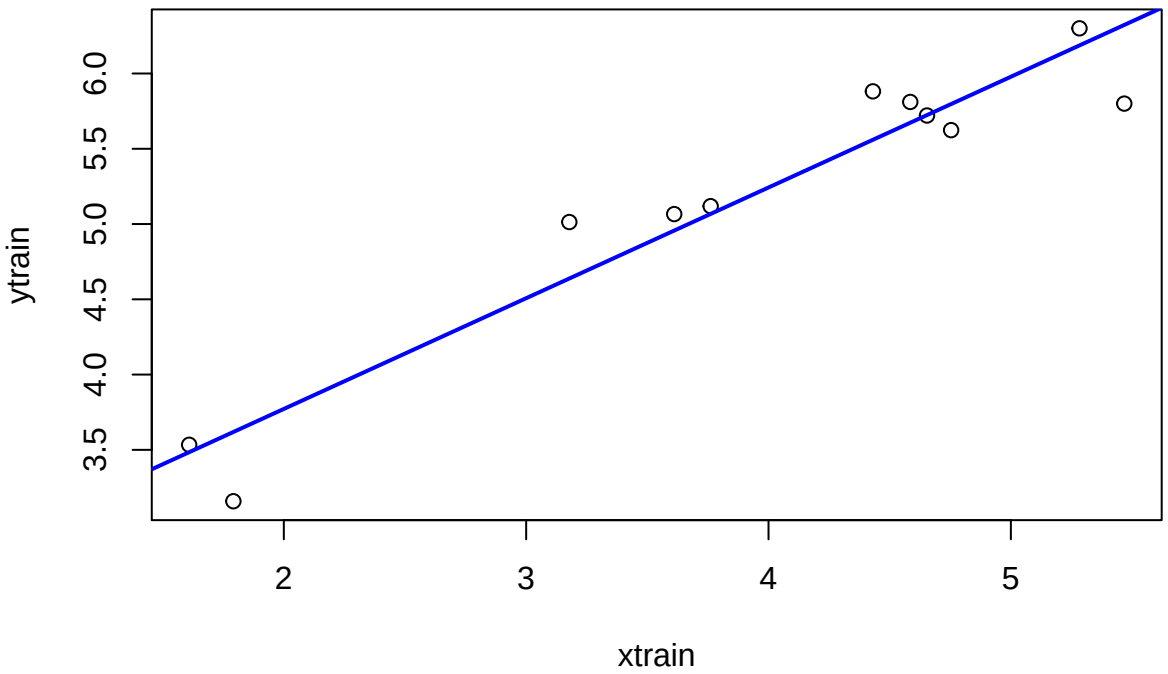
```
pred  <-  predEffort <= uppred & predEffort >= lowpred  #pred is a vector with logical values
Lpred <- sum(pred)/length(pred)
Lpred
```

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

12.2 Building a Linear Model on the Telecom1 dataset

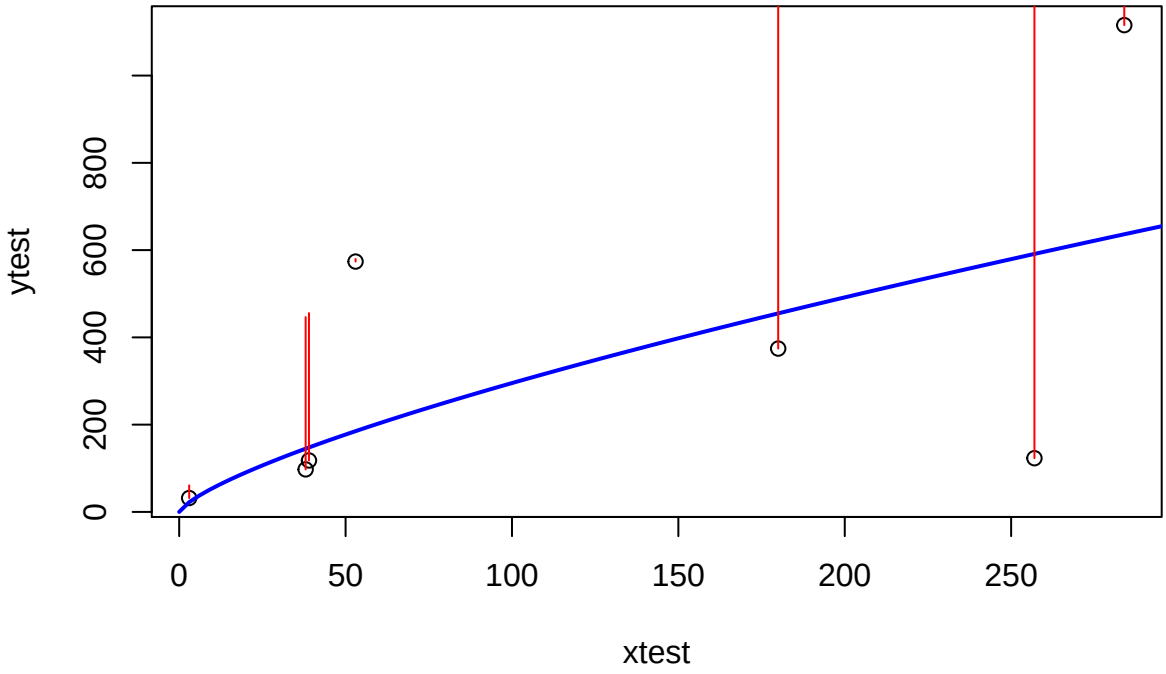
- *Although there are few datapoints we split the file into Train (2/3) and Test (1/3)*

```
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+b1*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")
```



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

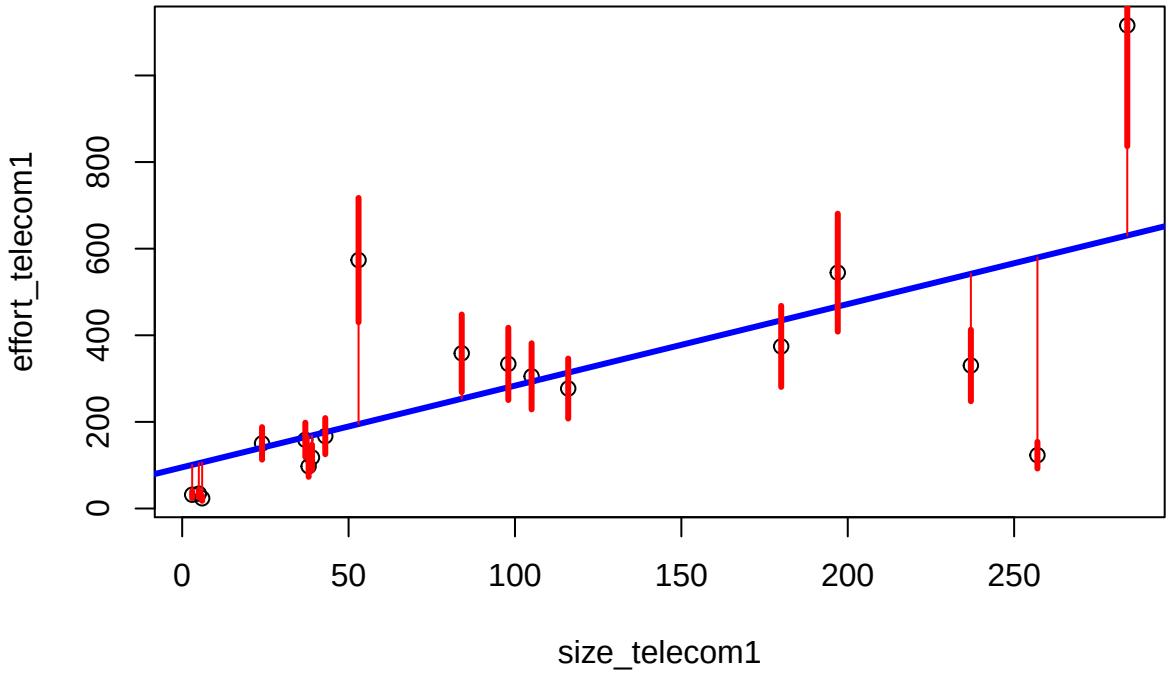
- *Just to visualize results*

```
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
```

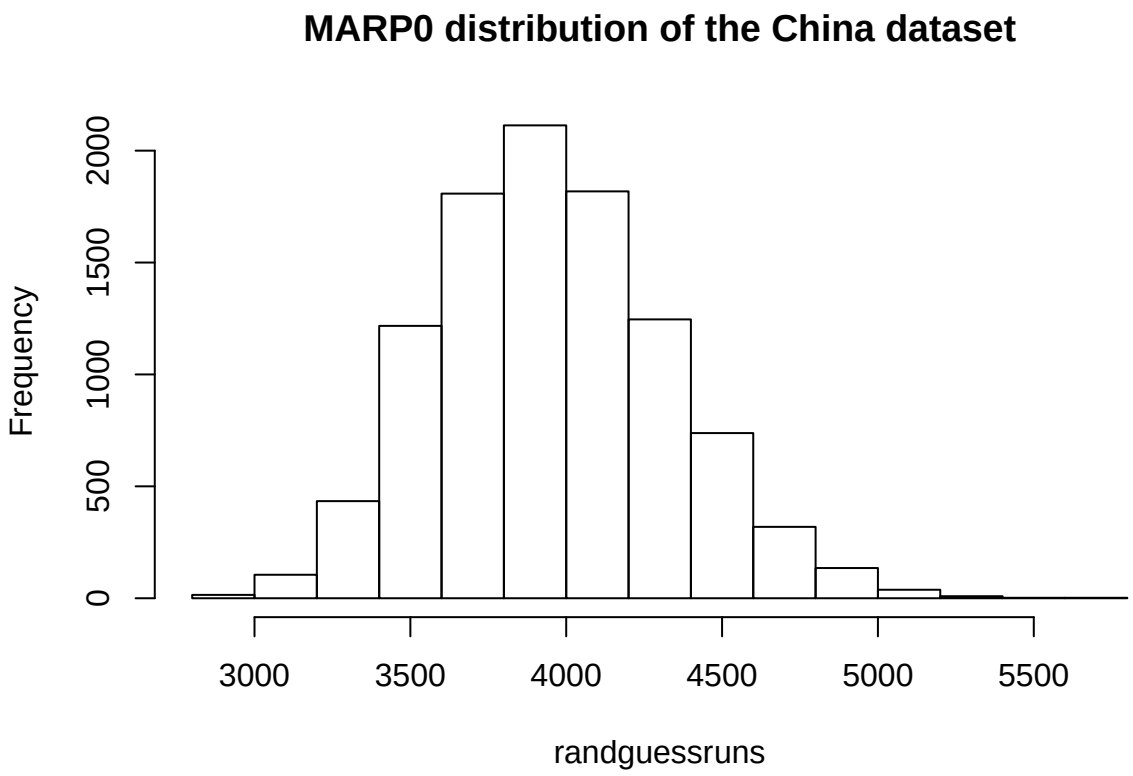
12.4 Standardised Accuracy. MARP0. ChinaTest

- *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
```

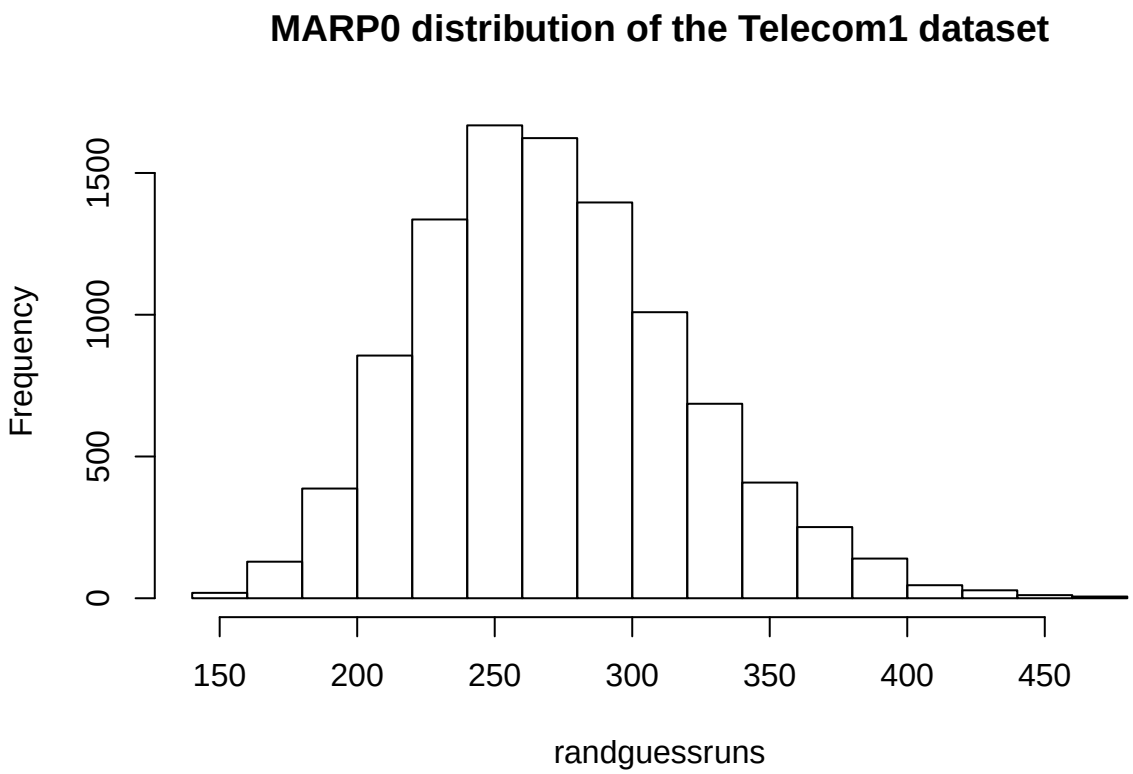
12.5 Standardised Accuracy. MARP0. Telecom1

- *Computing* $MARP_0$

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



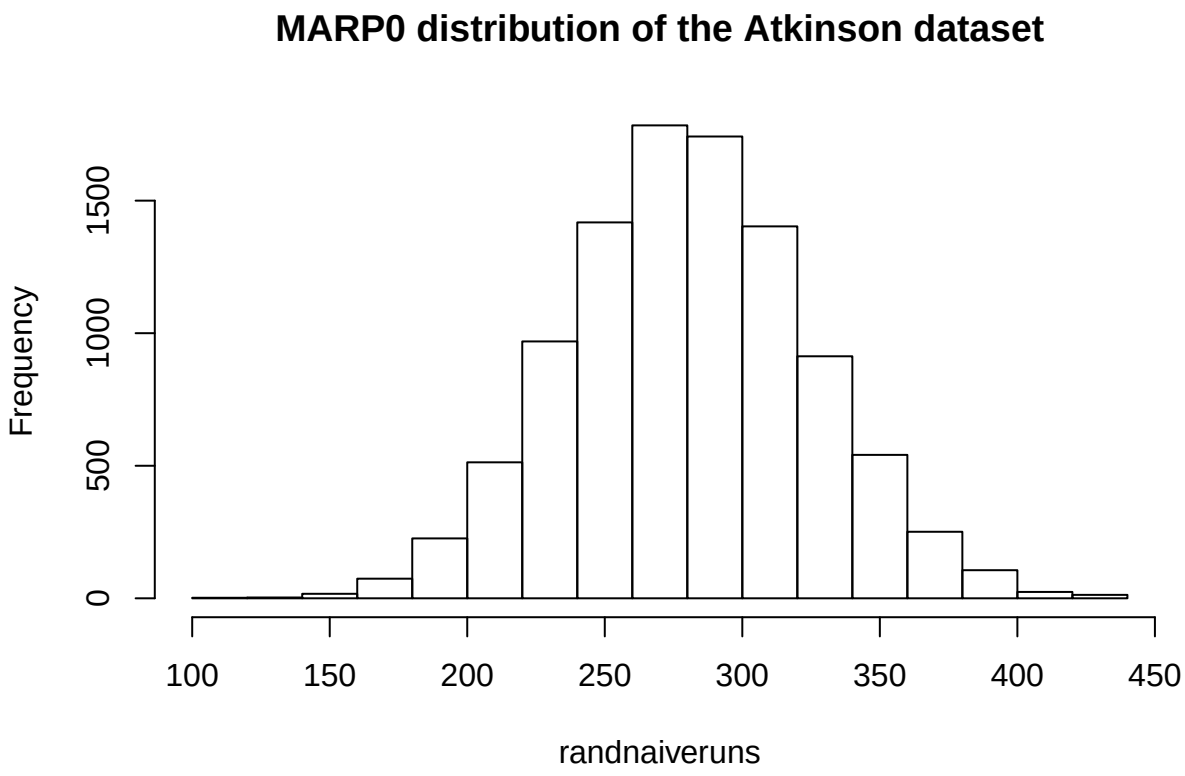

```
saTelecom1 <- (1- mar_tel1/marp0telecom1)*100
saTelecom1
```

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

12.5.1 MARP0 in the Atkinson dataset

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

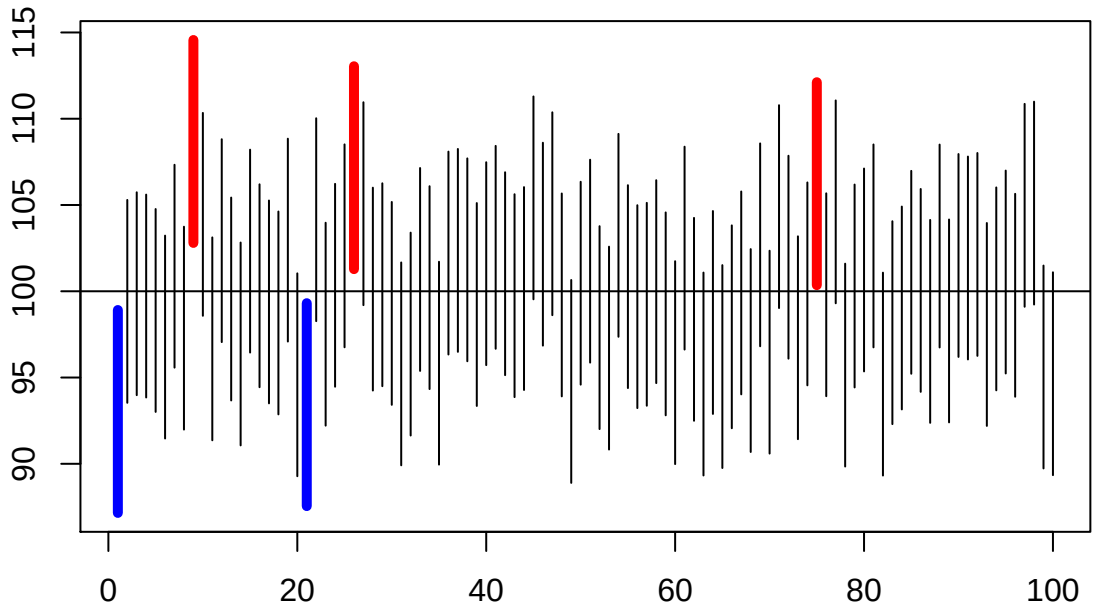
```
## [1] 281
```



12.6 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*

```
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:* Robust misinterpretation of confidence intervals, *Rink Hoekstra, Richard D. Morey, Jeffrey N. Rouder, Eric-Jan Wagenmakers. Psychonomic Bulletin & Review, October 2014*

12.7 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.*
- *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
```

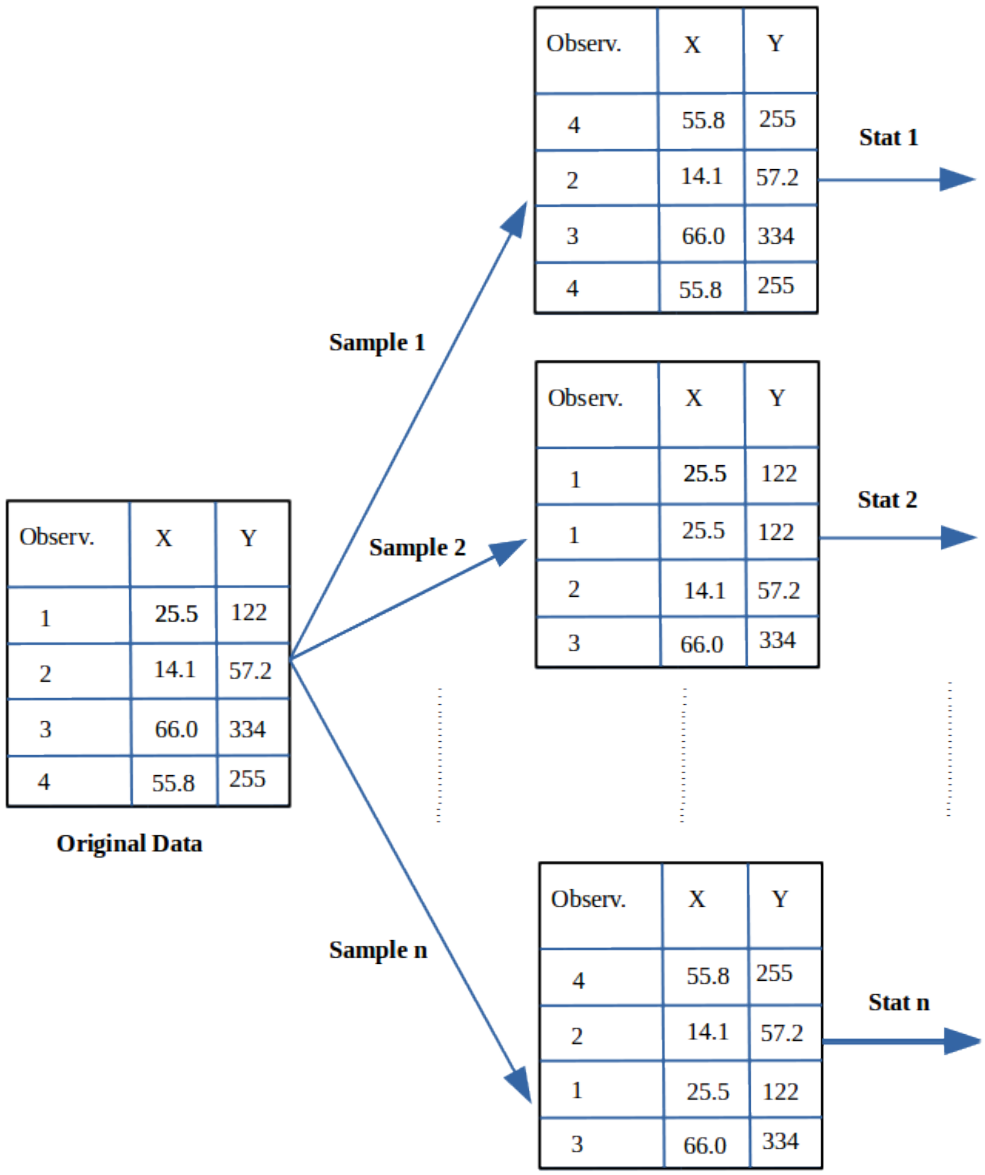
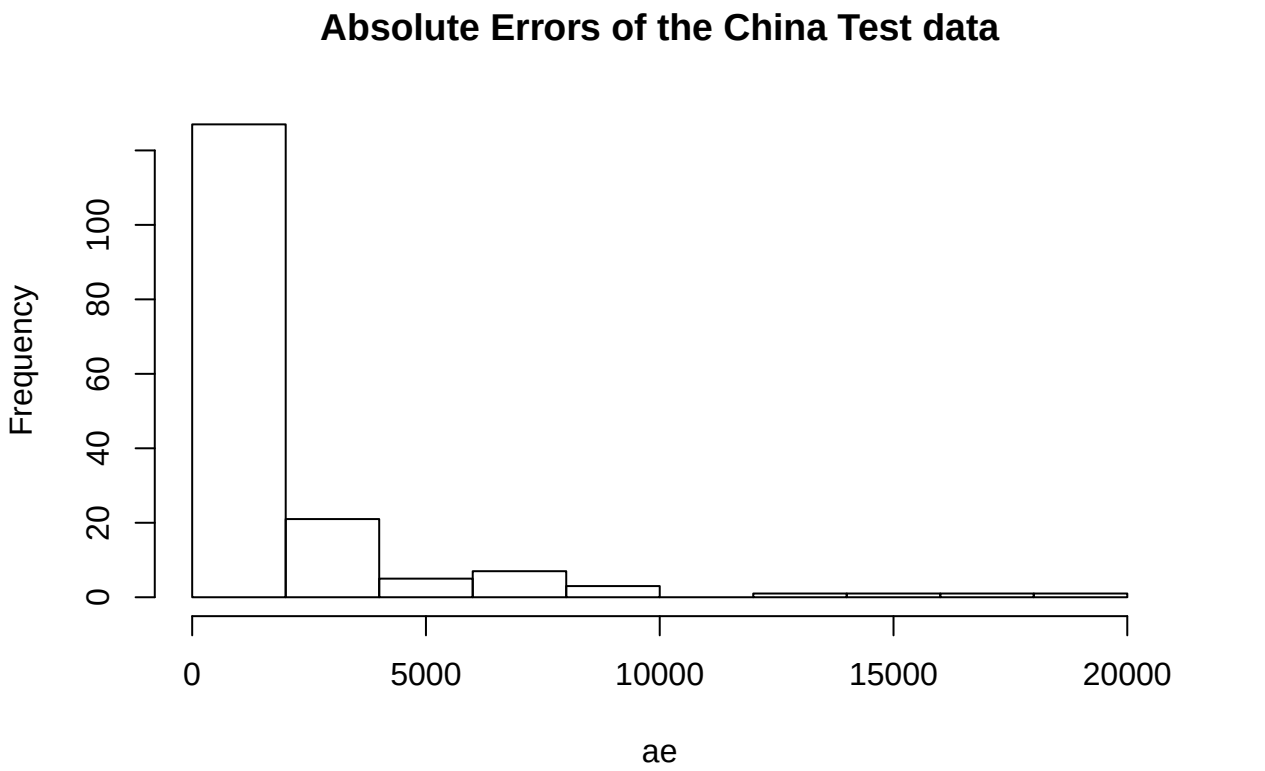

Observ. X Y
1 25.5 122
2 14.1 57.2
3 66.0 334
4 55.8 255
Original Data
Sample 1
Sample 2
Sample n
Observ. X Y
4 55.8 255
2 14.1 57.2
3 66.0 334
4 55.8 255
Stat 1
Observ. X Y
1 25.5 122
1 25.5 122
2 14.1 57.2
3 66.0 334
Stat 2
Observ. X Y
4 55.8 255
2 14.1 57.2
1 25.5 122
3 66.0 334
Stat n

Figure 12.1:

```
## 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%   (1415, 2309 )   (1389, 2282 )
##
## Level     Percentile            BCa
## 95%   (1451, 2345 )   (1488, 2411 )
## 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: 675.783 - 1012.201"
```

```
# 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 V

Advanced Topics

Chapter 13

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].*

13.1 Instance Selection

13.2 Missing Data Imputation

Chapter 14

Feature Selection Example

Feature Selection in R and Caret

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

```
## 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 object is masked from 'package:glmnet':
##
##     auc
```

```
## 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 15

Advanced Models

15.1 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.

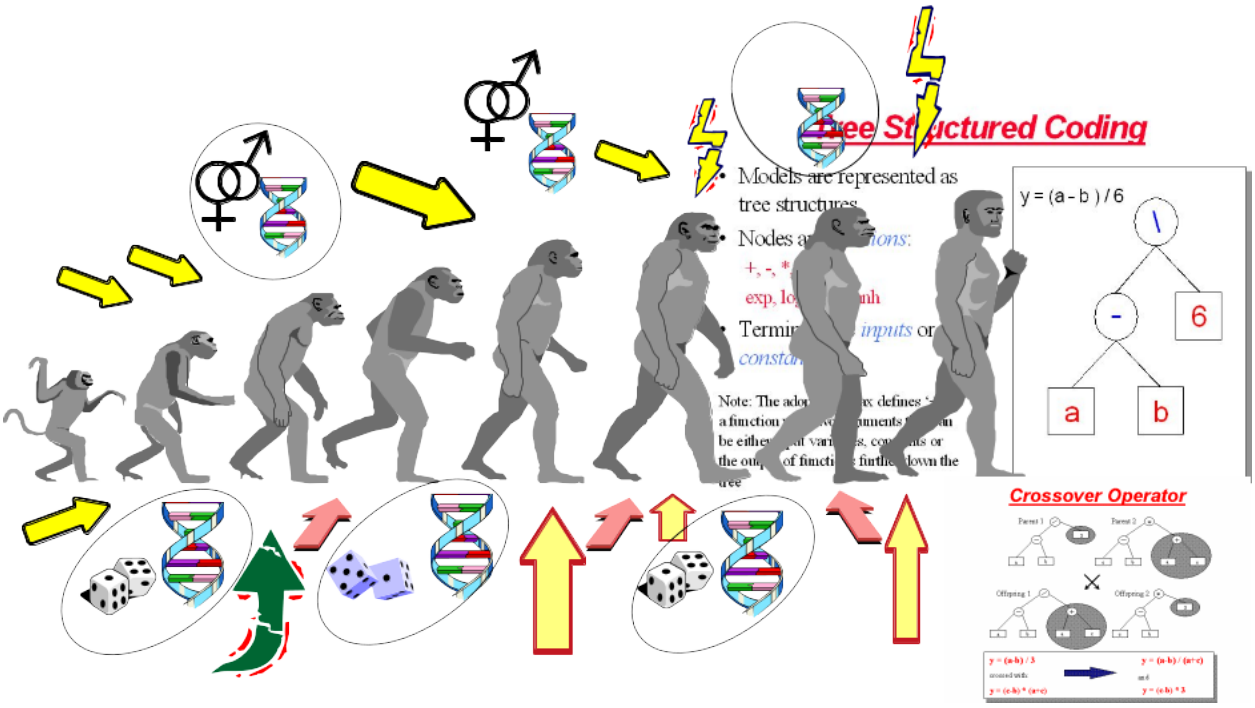
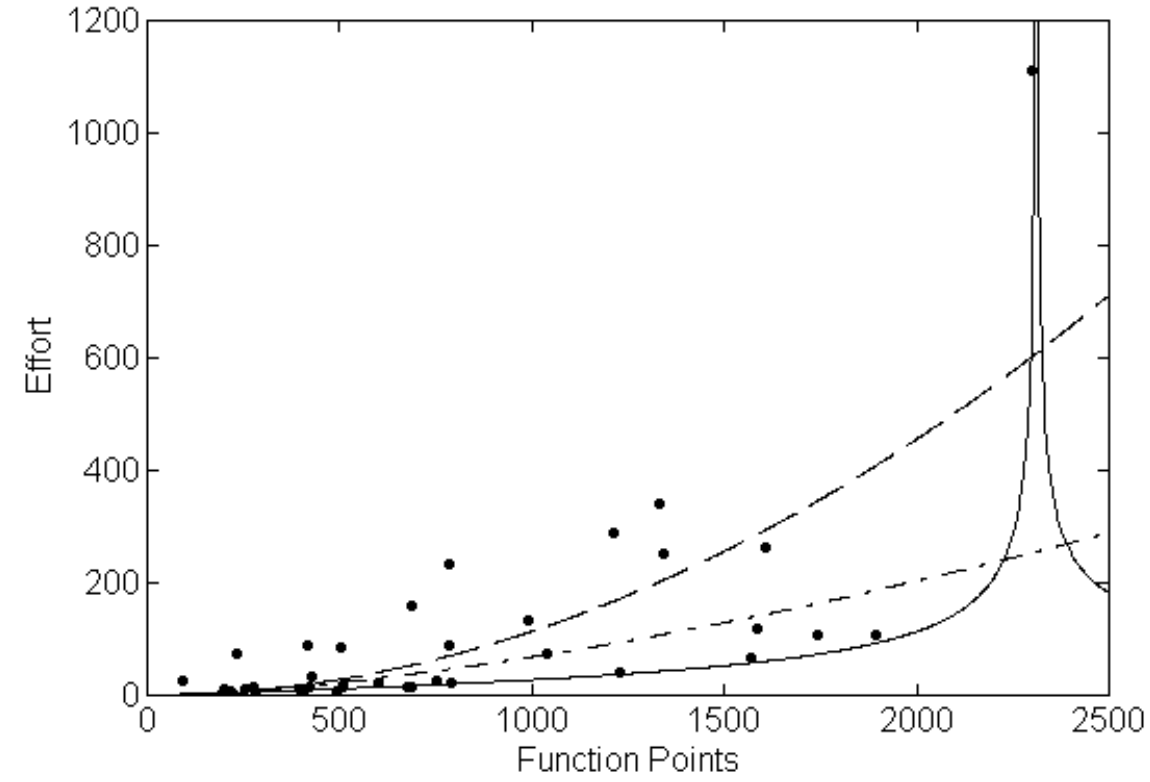
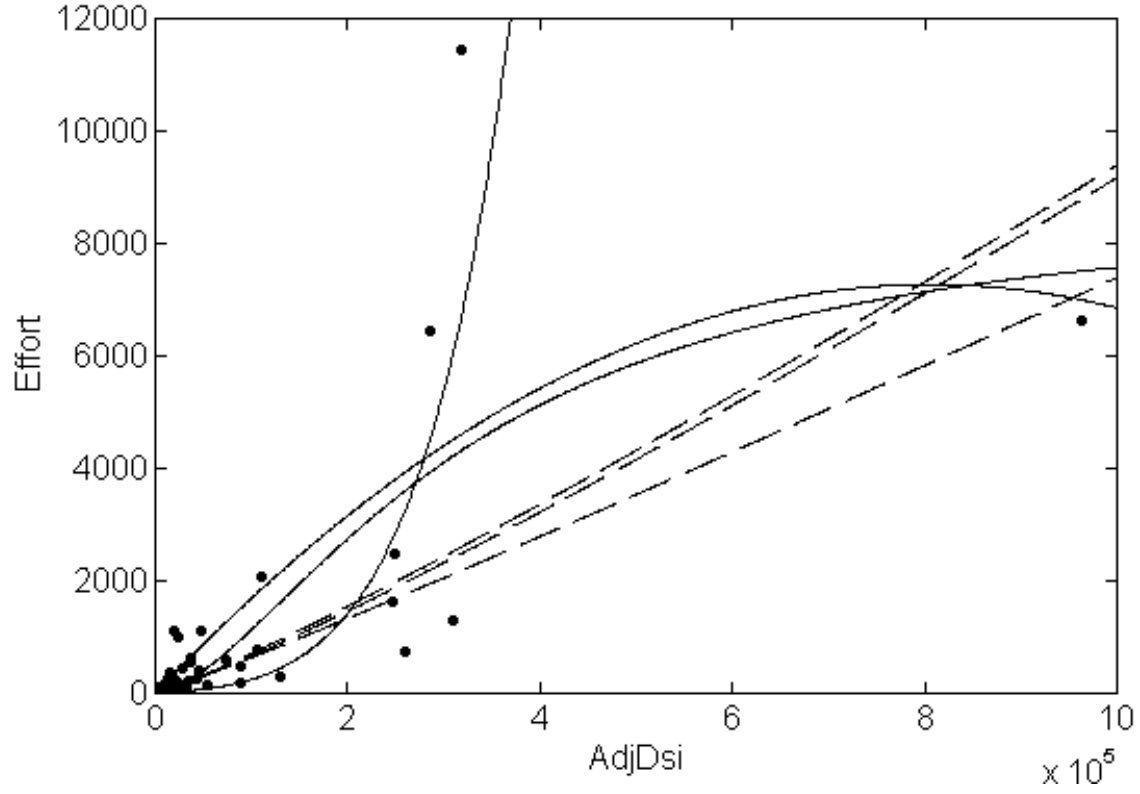


Figure 15.1:

- *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 15.2:

15.2 Genetic Programming Example

15.2.1 Load Data

```
library(foreign)

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

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
```

15.2.2 Genetic Programming for Symbolic Regression: China dataset.

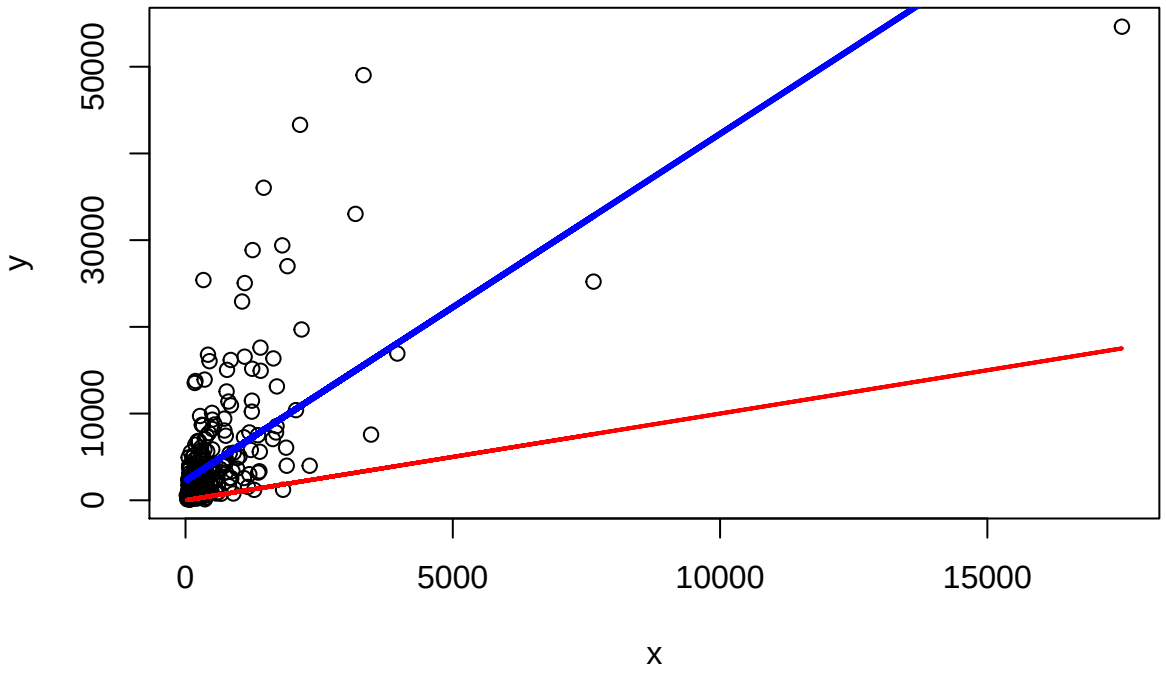
```
library("rgp")
```

```
## *** RGP version 0.4-1 initialized successfully.
##     Type 'help(package="rgp")' to bring up the RGP help pages,
##     or type 'vignette("rgp_introduction")' to show RGP's package vignette.
##     Type 'symbolicRegressionUi()' to bring up the symbolic regression UI if
##     the optional package 'rgpui' is installed.
```

```
options(digits = 5)
stepsGenerations <- 1000
initialPopulation <- 500
Steps <- c(1000)
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))
```

```
## STARTING genetic programming evolution run (Age/Fitness/Complexity Pareto GP search-heuristic) ...

## evolution step 100, fitness evaluations: 4950, best fitness: 6537.594992, time elapsed: 5.07 seconds

## evolution step 200, fitness evaluations: 9950, best fitness: 6537.594992, time elapsed: 8.32 seconds

## evolution step 300, fitness evaluations: 14950, best fitness: 6537.594992, time elapsed: 11.47 seconds

## evolution step 400, fitness evaluations: 19950, best fitness: 6012.656581, time elapsed: 15.11 seconds

## evolution step 500, fitness evaluations: 24950, best fitness: 5682.132944, time elapsed: 18.29 seconds

## evolution step 600, fitness evaluations: 29950, best fitness: 5682.132944, time elapsed: 21.25 seconds

## evolution step 700, fitness evaluations: 34950, best fitness: 5682.132944, time elapsed: 24.37 seconds

## evolution step 800, fitness evaluations: 39950, best fitness: 5682.132944, time elapsed: 27.51 seconds

## evolution step 900, fitness evaluations: 44950, best fitness: 5623.075872, time elapsed: 30.69 seconds

## evolution step 1000, fitness evaluations: 49950, best fitness: 5209.146640, time elapsed: 33.95 seconds

## Genetic programming evolution run FINISHED after 1000 evolution steps, 49950 fitness evaluations and 33.9
```

```
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] 2419.7
```

15.2.3 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)
```

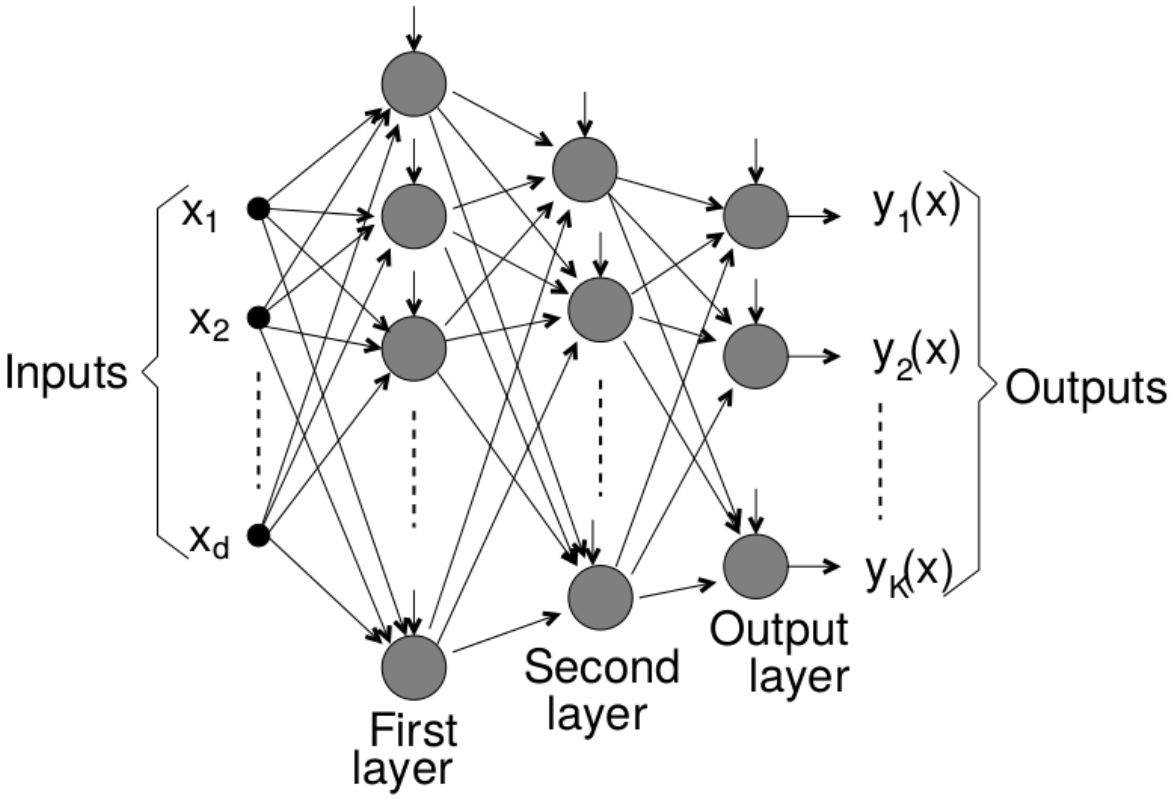
15.3 Neural Networks

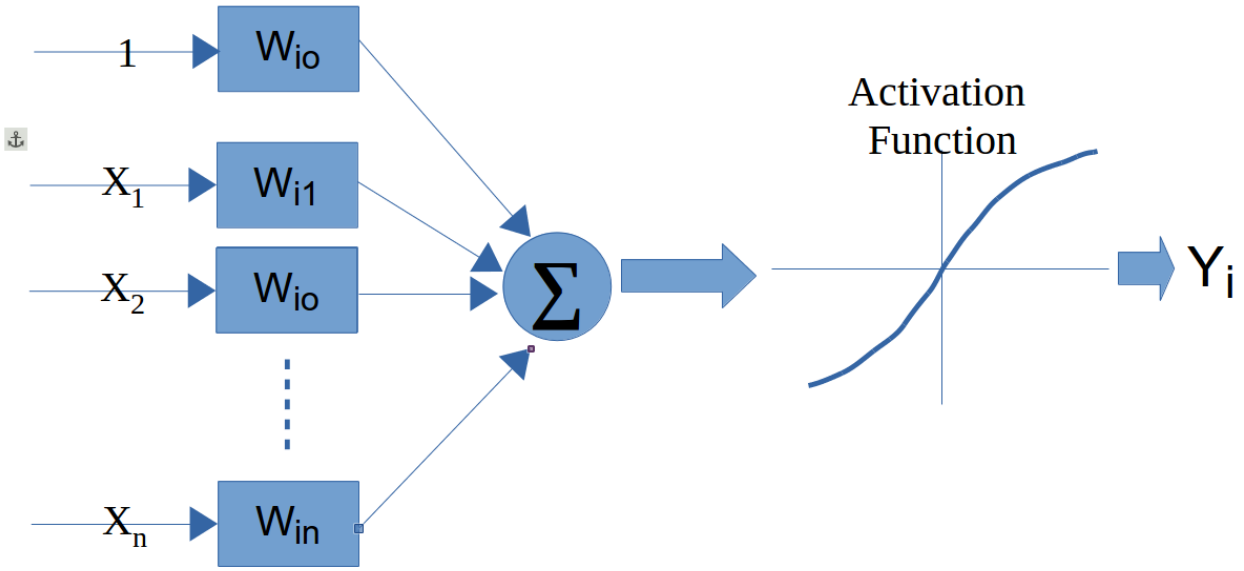
A neural network (NN) simulates some of the learning functions of the human brain.

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)
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

```
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)
```

15.4 Support Vector Machines

SVM

15.5 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).

15.5.1 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.

15.5.2 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.

15.5.3 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.

15.5.4 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")
                  )
```

```
## Loading required package: gbm

## Loading required package: splines

## Loaded gbm 2.1.1

## Loading required package: plyr

## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attaching package: 'plyr'

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

## The following objects are masked from 'package:Hmisc':
##
##     is.discrete, summarize
```

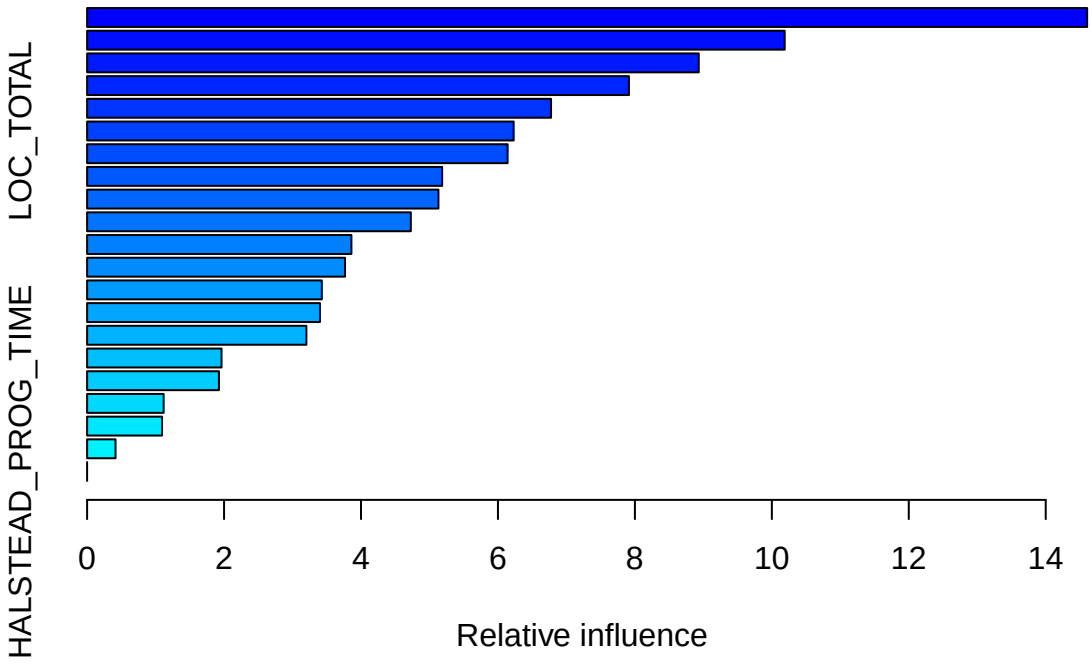
```
## The following object is masked from 'package:lubridate':
##
##     here

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following object is masked from 'package:DMwR':
##
##     join

## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8352            -nan     0.1000    0.0136
##      2        0.8085            -nan     0.1000    0.0106
##      3        0.7894            -nan     0.1000    0.0084
##      4        0.7738            -nan     0.1000    0.0074
##      5        0.7556            -nan     0.1000    0.0072
##      6        0.7466            -nan     0.1000    0.0029
##      7        0.7368            -nan     0.1000    0.0052
##      8        0.7279            -nan     0.1000    0.0046
##      9        0.7174            -nan     0.1000    0.0037
##     10        0.7132            -nan     0.1000    0.0014
##     20        0.6716            -nan     0.1000    0.0000
##     40        0.6480            -nan     0.1000   -0.0004
##     60        0.6366            -nan     0.1000   -0.0001
##     80        0.6229            -nan     0.1000   -0.0003
##    100        0.6118            -nan     0.1000   -0.0003
##    120        0.6050            -nan     0.1000   -0.0005
##    140        0.5969            -nan     0.1000   -0.0003
##    150        0.5925            -nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8270            -nan     0.1000    0.0161
##      2        0.8060            -nan     0.1000    0.0074
##      3        0.7785            -nan     0.1000    0.0091
##      4        0.7557            -nan     0.1000    0.0085
##      5        0.7423            -nan     0.1000    0.0064
##      6        0.7331            -nan     0.1000    0.0030
##      7        0.7223            -nan     0.1000    0.0019
##      8        0.7115            -nan     0.1000    0.0031
##      9        0.7009            -nan     0.1000    0.0041
##     10        0.6909            -nan     0.1000    0.0044
##     20        0.6449            -nan     0.1000   -0.0002
##     40        0.6005            -nan     0.1000   -0.0008
##     60        0.5696            -nan     0.1000   -0.0002
##     80        0.5462            -nan     0.1000   -0.0002
##    100        0.5274            -nan     0.1000   -0.0008
##    120        0.5110            -nan     0.1000   -0.0012
##    140        0.4952            -nan     0.1000   -0.0004
##    150        0.4880            -nan     0.1000   -0.0006
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8316            -nan     0.1000    0.0133
```

```
##      2        0.7987             -nan     0.1000    0.0126
##      3        0.7686             -nan     0.1000    0.0116
##      4        0.7471             -nan     0.1000    0.0086
##      5        0.7319             -nan     0.1000    0.0051
##      6        0.7163             -nan     0.1000    0.0049
##      7        0.7049             -nan     0.1000    0.0027
##      8        0.6934             -nan     0.1000    0.0038
##      9        0.6829             -nan     0.1000    0.0035
##     10        0.6755             -nan     0.1000    0.0027
##     20        0.6168             -nan     0.1000   -0.0007
##     40        0.5707             -nan     0.1000   -0.0018
##     60        0.5396             -nan     0.1000   -0.0012
##     80        0.5064             -nan     0.1000   -0.0011
##    100        0.4836             -nan     0.1000   -0.0009
##    120        0.4574             -nan     0.1000   -0.0003
##    140        0.4343             -nan     0.1000   -0.0011
##    150        0.4248             -nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8399             -nan     0.1000    0.0110
##      2        0.8180             -nan     0.1000    0.0107
##      3        0.8033             -nan     0.1000    0.0065
##      4        0.7887             -nan     0.1000    0.0073
##      5        0.7762             -nan     0.1000    0.0068
##      6        0.7630             -nan     0.1000    0.0048
##      7        0.7571             -nan     0.1000    0.0022
##      8        0.7497             -nan     0.1000    0.0026
##      9        0.7403             -nan     0.1000    0.0041
##     10        0.7344             -nan     0.1000    0.0022
##     20        0.7032             -nan     0.1000    0.0004
##     40        0.6799             -nan     0.1000   -0.0002
##     60        0.6617             -nan     0.1000   -0.0005
##     80        0.6481             -nan     0.1000   -0.0007
##    100        0.6392             -nan     0.1000   -0.0006
##    120        0.6302             -nan     0.1000   -0.0001
##    140        0.6217             -nan     0.1000   -0.0007
##    150        0.6171             -nan     0.1000   -0.0004
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8328             -nan     0.1000    0.0172
##      2        0.8082             -nan     0.1000    0.0096
##      3        0.7898             -nan     0.1000    0.0087
##      4        0.7758             -nan     0.1000    0.0035
##      5        0.7619             -nan     0.1000    0.0041
##      6        0.7492             -nan     0.1000    0.0049
##      7        0.7379             -nan     0.1000    0.0048
##      8        0.7311             -nan     0.1000    0.0011
##      9        0.7197             -nan     0.1000    0.0035
##     10        0.7160             -nan     0.1000    0.0007
##     20        0.6746             -nan     0.1000   -0.0001
##     40        0.6312             -nan     0.1000    0.0003
##     60        0.6019             -nan     0.1000   -0.0012
##     80        0.5818             -nan     0.1000   -0.0006
##    100        0.5623             -nan     0.1000   -0.0010
```

```
##    120        0.5455             -nan     0.1000   -0.0012
##    140        0.5317             -nan     0.1000   -0.0005
##    150        0.5227             -nan     0.1000   -0.0003
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8283             -nan     0.1000    0.0109
##      2        0.8016             -nan     0.1000    0.0122
##      3        0.7791             -nan     0.1000    0.0075
##      4        0.7607             -nan     0.1000    0.0072
##      5        0.7470             -nan     0.1000    0.0035
##      6        0.7352             -nan     0.1000    0.0054
##      7        0.7199             -nan     0.1000    0.0054
##      8        0.7111             -nan     0.1000    0.0030
##      9        0.7015             -nan     0.1000    0.0023
##     10        0.6925             -nan     0.1000    0.0016
##     20        0.6347             -nan     0.1000    0.0002
##     40        0.5882             -nan     0.1000   -0.0019
##     60        0.5546             -nan     0.1000   -0.0014
##     80        0.5269             -nan     0.1000   -0.0007
##    100        0.5025             -nan     0.1000   -0.0021
##    120        0.4782             -nan     0.1000   -0.0008
##    140        0.4555             -nan     0.1000   -0.0007
##    150        0.4442             -nan     0.1000   -0.0016
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8443             -nan     0.1000    0.0112
##      2        0.8248             -nan     0.1000    0.0098
##      3        0.8068             -nan     0.1000    0.0075
##      4        0.7962             -nan     0.1000    0.0042
##      5        0.7816             -nan     0.1000    0.0051
##      6        0.7711             -nan     0.1000    0.0044
##      7        0.7597             -nan     0.1000    0.0024
##      8        0.7528             -nan     0.1000    0.0028
##      9        0.7464             -nan     0.1000    0.0020
##     10        0.7384             -nan     0.1000    0.0036
##     20        0.7041             -nan     0.1000   -0.0007
##     40        0.6821             -nan     0.1000   -0.0012
##     60        0.6624             -nan     0.1000   -0.0006
##     80        0.6505             -nan     0.1000   -0.0003
##    100        0.6412             -nan     0.1000   -0.0003
##    120        0.6333             -nan     0.1000   -0.0004
##    140        0.6242             -nan     0.1000   -0.0004
##    150        0.6206             -nan     0.1000   -0.0009
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.8370             -nan     0.1000    0.0144
##      2        0.8118             -nan     0.1000    0.0099
##      3        0.7922             -nan     0.1000    0.0086
##      4        0.7748             -nan     0.1000    0.0081
##      5        0.7630             -nan     0.1000    0.0044
##      6        0.7521             -nan     0.1000    0.0037
##      7        0.7405             -nan     0.1000    0.0028
##      8        0.7317             -nan     0.1000    0.0010
##      9        0.7265             -nan     0.1000    0.0009
```

```
##     10         0.7196             -nan     0.1000    0.0019
##     20         0.6801             -nan     0.1000   -0.0000
##     40         0.6347             -nan     0.1000   -0.0006
##     60         0.6025             -nan     0.1000   -0.0004
##     80         0.5789             -nan     0.1000   -0.0007
##    100         0.5579             -nan     0.1000   -0.0015
##    120         0.5428             -nan     0.1000   -0.0005
##    140         0.5289             -nan     0.1000   -0.0011
##    150         0.5200             -nan     0.1000   -0.0003
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1         0.8322             -nan     0.1000    0.0125
##      2         0.8043             -nan     0.1000    0.0118
##      3         0.7784             -nan     0.1000    0.0087
##      4         0.7629             -nan     0.1000    0.0043
##      5         0.7465             -nan     0.1000    0.0065
##      6         0.7326             -nan     0.1000    0.0045
##      7         0.7215             -nan     0.1000    0.0017
##      8         0.7126             -nan     0.1000    0.0013
##      9         0.7047             -nan     0.1000   -0.0010
##     10         0.6936             -nan     0.1000    0.0031
##     20         0.6480             -nan     0.1000   -0.0008
##     40         0.5896             -nan     0.1000   -0.0005
##     60         0.5552             -nan     0.1000   -0.0007
##     80         0.5259             -nan     0.1000   -0.0005
##    100         0.4969             -nan     0.1000   -0.0007
##    120         0.4730             -nan     0.1000   -0.0008
##    140         0.4522             -nan     0.1000   -0.0003
##    150         0.4448             -nan     0.1000   -0.0009
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1         0.8364             -nan     0.1000    0.0147
##      2         0.8103             -nan     0.1000    0.0136
##      3         0.7938             -nan     0.1000    0.0081
##      4         0.7757             -nan     0.1000    0.0085
##      5         0.7617             -nan     0.1000    0.0071
##      6         0.7511             -nan     0.1000    0.0046
##      7         0.7415             -nan     0.1000    0.0032
##      8         0.7295             -nan     0.1000    0.0037
##      9         0.7198             -nan     0.1000    0.0031
##     10         0.7131             -nan     0.1000    0.0028
##     20         0.6773             -nan     0.1000   -0.0009
##     40         0.6413             -nan     0.1000   -0.0006
##     60         0.6180             -nan     0.1000   -0.0007
##     80         0.6015             -nan     0.1000   -0.0008
##    100         0.5866             -nan     0.1000   -0.0002
##    120         0.5695             -nan     0.1000   -0.0011
##    140         0.5569             -nan     0.1000   -0.0005
##    150         0.5518             -nan     0.1000   -0.0006
```

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




```
##                                       var       rel.inf
## HALSTEAD_CONTENT           HALSTEAD_CONTENT 14.6047519607
## HALSTEAD_DIFFICULTY     HALSTEAD_DIFFICULTY 10.1873906874
## NUM_OPERANDS                   NUM_OPERANDS  8.9325104200
## NUM_OPERATORS                 NUM_OPERATORS  7.9127746116
## NUM_UNIQUE_OPERATORS   NUM_UNIQUE_OPERATORS  6.7756582333
## LOC_TOTAL                         LOC_TOTAL  6.2296682519
## NUM_UNIQUE_OPERANDS     NUM_UNIQUE_OPERANDS  6.1410664847
## HALSTEAD_EFFORT             HALSTEAD_EFFORT  5.1853100309
## LOC_COMMENTS                   LOC_COMMENTS  5.1306009233
## LOC_EXECUTABLE               LOC_EXECUTABLE  4.7274594398
## LOC_CODE_AND_COMMENT   LOC_CODE_AND_COMMENT  3.8598106671
## HALSTEAD_LENGTH             HALSTEAD_LENGTH  3.7668756935
## HALSTEAD_VOLUME             HALSTEAD_VOLUME  3.4277318907
## ESSENTIAL_COMPLEXITY   ESSENTIAL_COMPLEXITY  3.4005292639
## BRANCH_COUNT                   BRANCH_COUNT  3.2023641850
## DESIGN_COMPLEXITY         DESIGN_COMPLEXITY  1.9630713543
## LOC_BLANK                         LOC_BLANK  1.9252046668
## HALSTEAD_LEVEL               HALSTEAD_LEVEL  1.1178129374
## CYCLOMATIC_COMPLEXITY CYCLOMATIC_COMPLEXITY  1.0945057892
## HALSTEAD_ERROR_EST       HALSTEAD_ERROR_EST  0.4149025082
## 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      0.8054441609  0.9826224329  0.1282051282
##   1                  100      0.8090442338  0.9802527646  0.1581196581
##   1                  150      0.8050492162  0.9755134281  0.1794871795
##   2                   50      0.8068619111  0.9739336493  0.1623931624
##   2                  100      0.8085834650  0.9763033175  0.2051282051
##   2                  150      0.8108518654  0.9684044234  0.2307692308
##   3                   50      0.8042340098  0.9691943128  0.1837606838
##   3                  100      0.8054694779  0.9668246445  0.2094017094
##   3                  150      0.8047859197  0.9636650869  0.2478632479
##
## 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 = 2, 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
## 0.8691275168 0.2982095951
```

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

```
##              N             Y
## 1 0.9135315703 0.08646842967
## 2 0.9756990212 0.02430097876
## 3 0.8465454841 0.15345451590
## 4 0.8366086073 0.16339139270
## 5 0.8333283490 0.16667165098
## 6 0.9344452820 0.06555471799
```

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

```
##         RMSE     Rsquared
## 0.2143954918           NA
```

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

```
## Area under the curve: 0.8049179
```

Chapter 16

Further Classification Models

16.1 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.)

16.2 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         13          0         0
##   versicolor      0         17         1
##   virginica       0          1        18
```

```
## 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         13          0         0
##   versicolor      0         13         0
##   virginica       0          5        19
```

```
## 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         13          0         0
##   versicolor      0         13         0
##   virginica       0          5        19
```

Chapter 17

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:

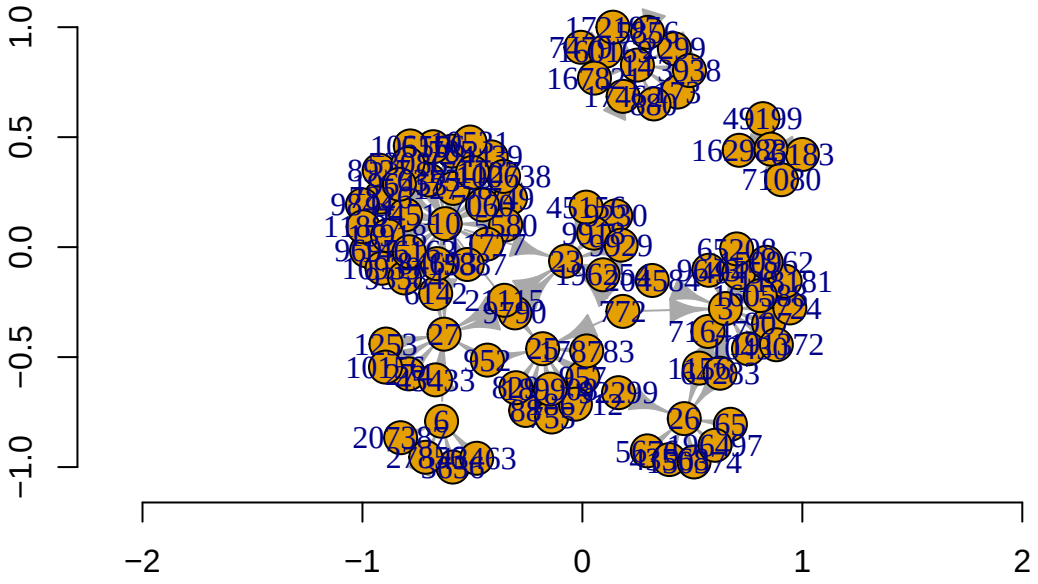
```
summary(g);
```

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

Plotting the graph:

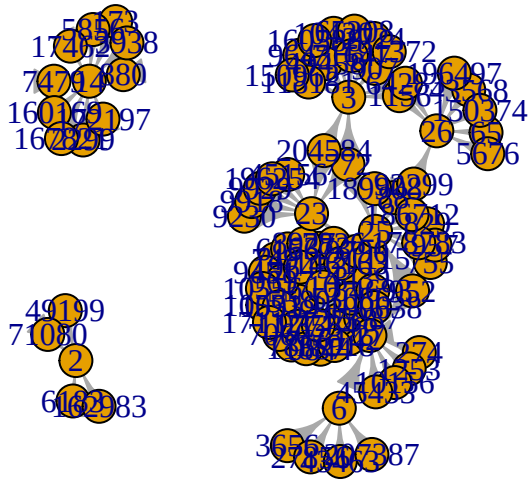
```
layout1 <-  layout.fruchterman.reingold(g)
plot(g, layout1)
```

```
## 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)
)

#
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
```

Chapter 18

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 19

Text Mining Soft Eng Data

In software engineering, there is a lot of information that can be extracted from Software Configuration Management System (SCM), or Bug Tracking Systems such as Bugzilla.

19.1 Example of classifying bugs from Bugzilla

Bugzilla is Issue Tracking System that allow us to maintain and track 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).

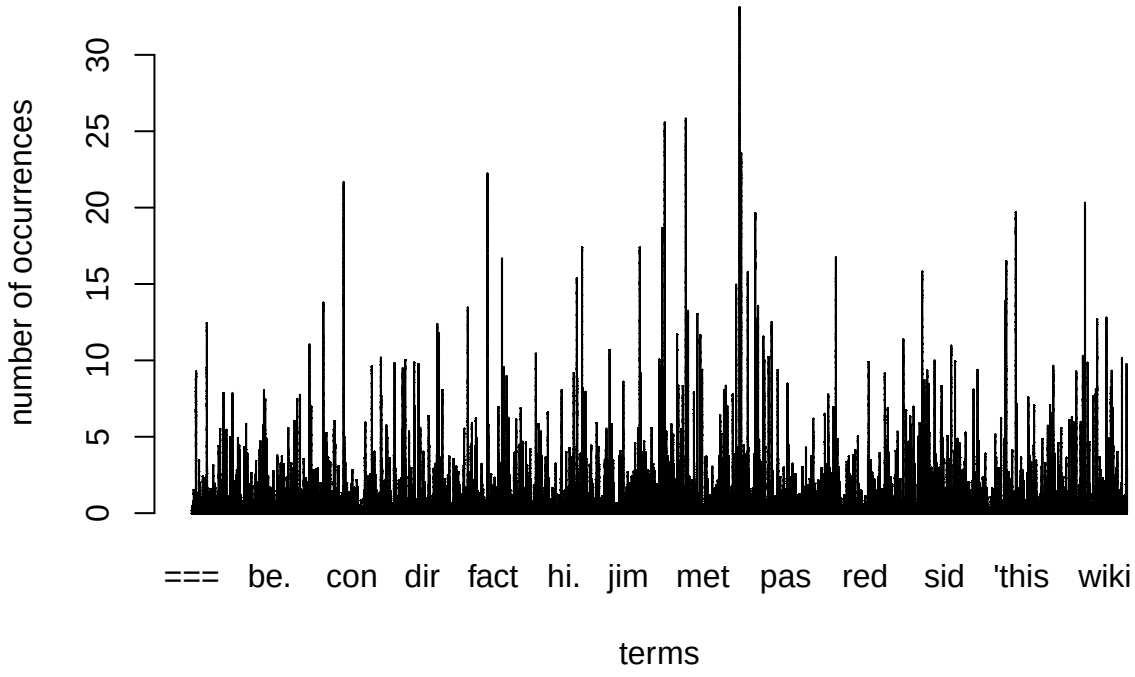
```
library(foreign)

d <- read.arff("./datasets/textMining/compendium.arff")
head(d,2)
```

```
##
## 1                                                                    Drag drop learning design zip
## 2 Error Launching Compendium install team Error message launching Compendium initial install Java Virtual
##   Category           Ann1         Ann2           Ann3           Ann4
## 1      Bug     Capability   Usability     Capability     Capability
## 2  Support Installability Reliability Installability Installability
##              Ann5     MostCommon
## 1     Capability     Capability
## 2 Installability Installability
```

Creating a Document-Term Matrix (DTM)

```
library(tm)

dfcomp <- data.frame(textCol = d$Description)  # , d$Category)

ds <- DataframeSource(dfcomp)
dsc <-Corpus(ds)

# weighting=TfIdf weighting is Tf-Idf
# minWordLength=WL the minimum word length is WL
# minDocFreq=ND each word must appear at least in ND docs

# Other options of DTM
```

```
# These are not really needed, if preprocessing has been carried out:
# stemming=TRUE stemming is applied
# stopwords=TRUE stopwords are eliminated
# removeNumbers=TRUE numbers are eliminated


dtm<- DocumentTermMatrix(dsc, control = list(weighting = weightTfIdf, minDocFreq=3, stopwords = TRUE, r

# dim(dtm)
# inspect(dtm) #[1:10,1:10])

# dtm.70=removeSparseTerms(dtm,sparse=0.7)
# dtm.70 # or dim(dtm.70)
# note that the term-document matrix needs to be transformed (casted)
# to a matrix form in the following barplot command

dtm.90=removeSparseTerms(dtm,sparse=0.9)

barplot(as.matrix(dtm),xlab="terms",ylab="number of occurrences",main="Most frequent terms (sparseness=
```



As data frame:

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


class <- d$Category
dtmdf <- cbind(dtmdf,class)
```

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

```
# which words are associated with "feature"?
findAssocs(dtm, 'feature', 0.40)
```

```
## $feature
## requests together   global
##     0.61     0.43     0.41
```

And find frequent terms.

```
findFreqTerms(dtm,5)
```

```
##   [1] "ability"     "able"        "add"         "allow"       "already"
##   [6] "also"        "always"      "another"     "arrange"     "back"
##  [11] "background"  "backup"      "box"         "bug"         "button"
##  [16] "can"         "change"      "changing"    "characters"  "click"
##  [21] "clone"       "color"       "colour"      "compendium"  "compendium."
##  [26] "contents"    "copy"        "create"      "created"     "current"
##  [31] "database"    "deep"        "default"     "delete"      "derby"
##  [36] "detail"      "details"     "dialog"      "different"   "display"
##  [41] "download"    "drag"        "drop"        "entered"     "error"
##  [46] "even"        "excel"       "export"      "exporting"   "exports"
##  [51] "feature"     "file"        "files"       "find"        "first"
##  [56] "font"        "formatting"  "get"         "global"      "good"
##  [61] "group"       "help"        "html"        "icon"        "icons"
##  [66] "image"       "images"      "import"      "indicator"   "installed"
##  [71] "integration" "interface"   "java"        "know"        "label"
##  [76] "labels"      "left"        "like"        "line"        "link"
##  [81] "links"       "list"        "lists"       "log"         "long"
##  [86] "low"         "mac"         "main"        "make"        "map"
##  [91] "maps"        "may"         "medium"      "menu"        "message"
##  [96] "move"        "movie"       "multiple"    "mysql"       "name"
## [101] "need"        "new"         "node"        "nodes"       "n't"
## [106] "old"         "one"         "open"        "option"      "options"
## [111] "originally"  "outline"     "page"        "paste"       "possible"
## [116] "print"       "problem"     "program"     "project"     "really"
## [121] "reference"   "reported"    "requested"   "results"     "right"
## [126] "rollover"    "run"         "running"     "save"        "screen"
## [131] "scribble"    "search"      "see"         "seems"       "select"
## [136] "selected"    "selvin"      "send"        "set"         "shortcut"
## [141] "shortcuts"   "show"        "simon"       "size"        "smartboard"
## [146] "specify"     "start"       "stencil"     "stencils"    "support"
## [151] "system"      "tag"         "tags"        "text"        "thanks"
## [156] "time"        "timeline"    "toolbar"     "tried"       "try"
## [161] "two"         "type"        "undo"        "update"      "url"
## [166] "used"        "users"       "using"       "'ve"         "version"
## [171] "view"        "views"       "want"        "way"         "web"
## [176] "window"      "windows"     "word"        "work"        "working"
## [181] "xml"         "zoom"
```

Use any classifier now:

```
library(caret)
library(randomForest)
```

```
## randomForest 4.6-12
```

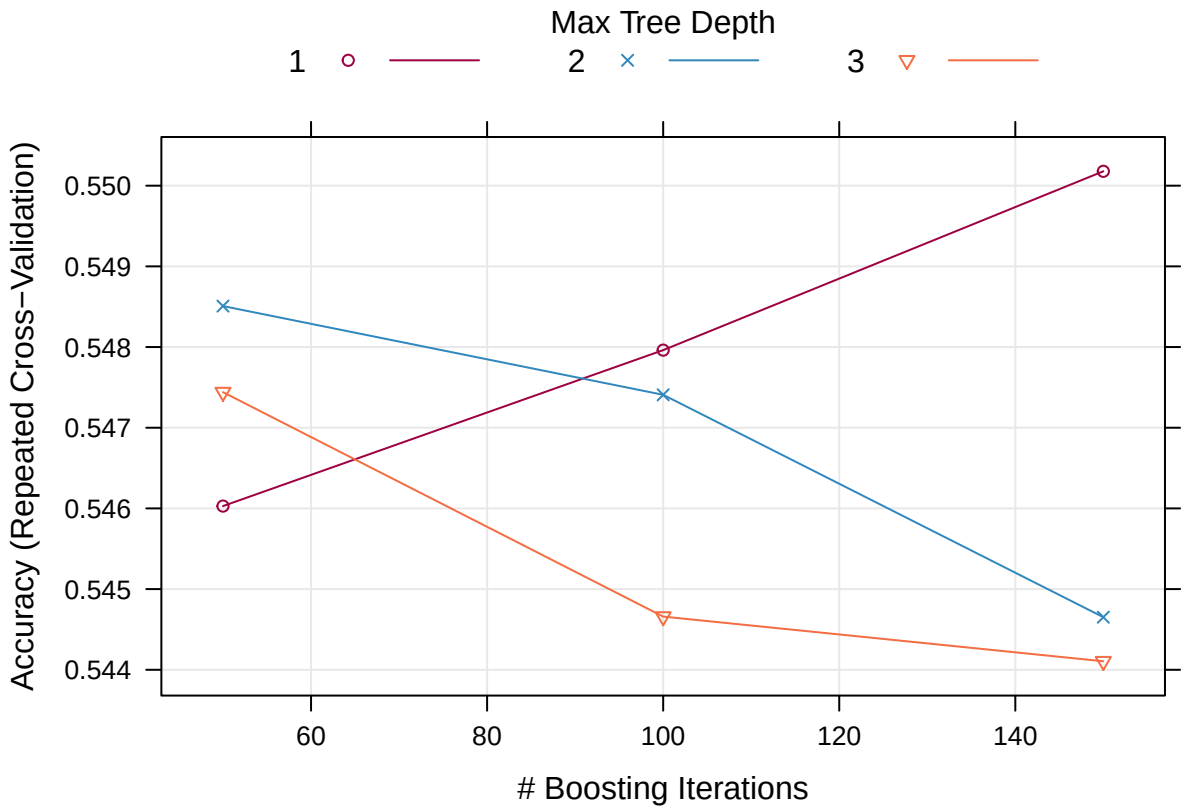
```
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:Hmisc':
##
##     combine
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
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
```

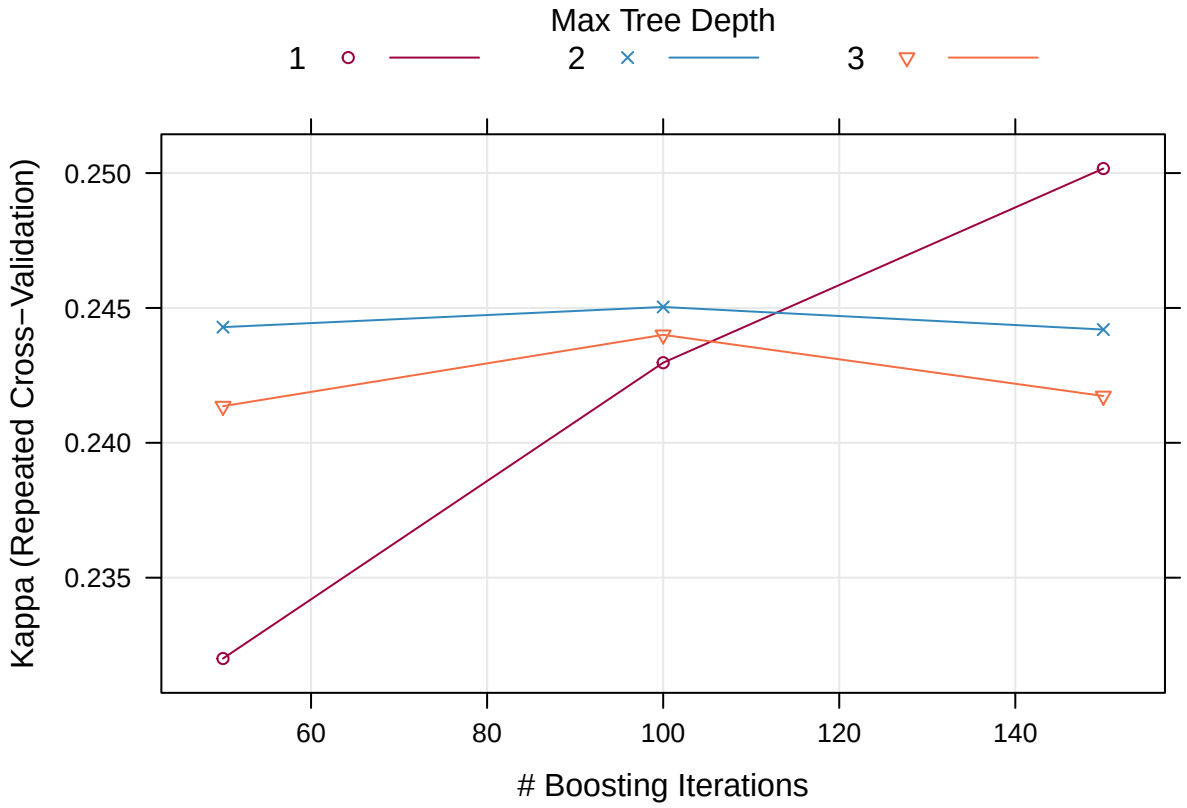
```
## Stochastic Gradient Boosting
##
## 723 samples
##  21 predictors
##   3 classes: 'Bug', 'Feature', 'Support'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 578, 579, 578, 579, 578, 578, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy      Kappa
##   1                   50      0.5460288294  0.2320030071
##   1                  100      0.5479617796  0.2429697978
##   1                  150      0.5501782815  0.2501673968
##   2                   50      0.5485077834  0.2442889212
##   2                  100      0.5474082201  0.2450358030
##   2                  150      0.5446515955  0.2441993226
##   3                   50      0.5474407604  0.2413583825
##   3                  100      0.5446591244  0.2440032591
```

```
##   3                   150      0.5441054578  0.2417363346
##
## 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"))
```

```
##             Bug      Feature       Support
## 1 0.67793156071 0.1275266302 0.19454180907
## 2 0.25335759991 0.3835160779 0.36312632218
## 3 0.31883175120 0.6276651785 0.05350307033
## 4 0.14557674137 0.7871140592 0.06730919948
## 5 0.03041910342 0.9488591442 0.02072175243
## 6 0.07188165781 0.8969776102 0.03114073197
```

```
confusionMatrix(testing$class, predict(gbmFit1, testing))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bug Feature Support
##    Bug      34      39       6
##    Feature  30      85       1
##    Support  11      18      15
##
## Overall Statistics
##
##                Accuracy : 0.5606695
##                  95% CI : (0.4952344, 0.6245808)
##     No Information Rate : 0.5941423
##     P-Value [Acc > NIR] : 0.8683772095
##
##                   Kappa : 0.2565766
##  Mcnemar's Test P-Value : 0.0004711919
```

```
##
## Statistics by Class:
##
##                      Class: Bug Class: Feature Class: Support
## Sensitivity           0.4533333      0.5985915     0.68181818
## Specificity           0.7256098      0.6804124     0.86635945
## Pos Pred Value        0.4303797      0.7327586     0.34090909
## Neg Pred Value        0.7437500      0.5365854     0.96410256
## Prevalence            0.3138075      0.5941423     0.09205021
## Detection Rate        0.1422594      0.3556485     0.06276151
## Detection Prevalence  0.3305439      0.4853556     0.18410042
## Balanced Accuracy     0.5894715      0.6395020     0.77408881
```

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

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
# calculate the frequency of words and sort in descending order.
wordFreqs=sort(colSums(as.matrix(dtm.90)),decreasing=TRUE)
wordcloud(words=names(wordFreqs),freq=wordFreqs)
```

19.2 Extracting data from Twitter

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

Chapter 20

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)
```

```
## Loading required package: timeDate
```

```
##
## Attaching package: 'timeDate'
```

```
## The following objects are masked from 'package:e1071':
##
##     kurtosis, skewness
```

```
## This is forecast 7.3
```

```
# 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)
```

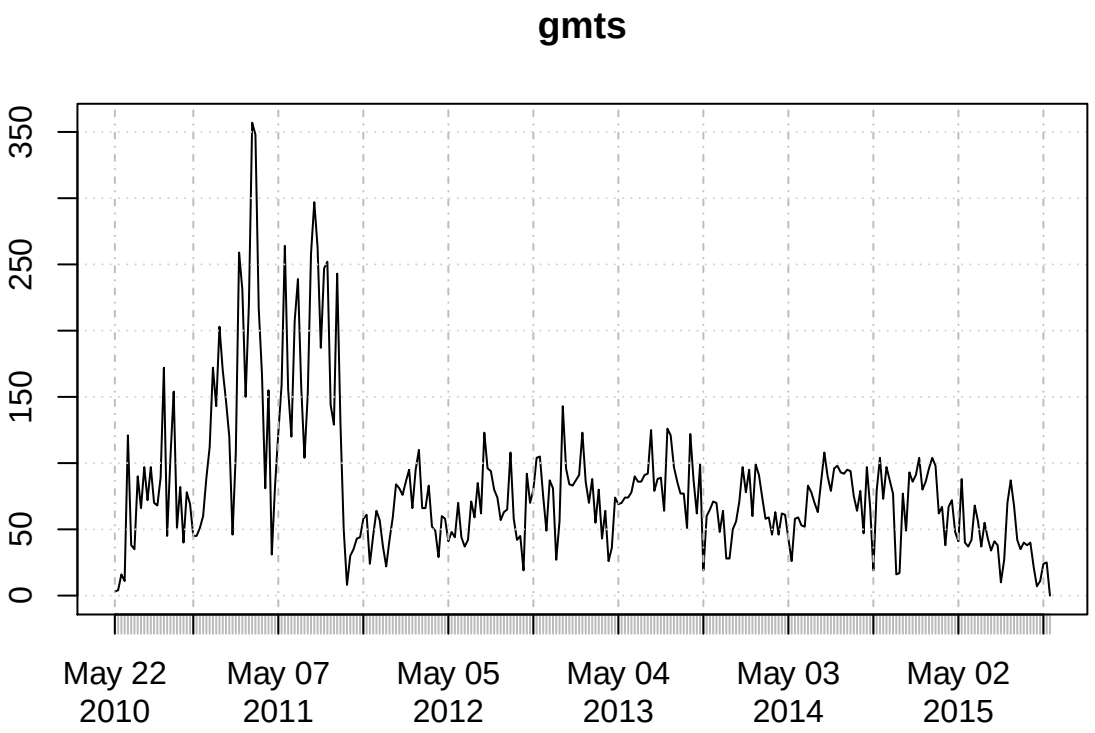
```
## An 'xts' object on 2010-05-22/2015-11-14 containing:
##   Data: int [1:287, 1] 3 4 16 11 121 38 35 90 66 97 ...
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
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:

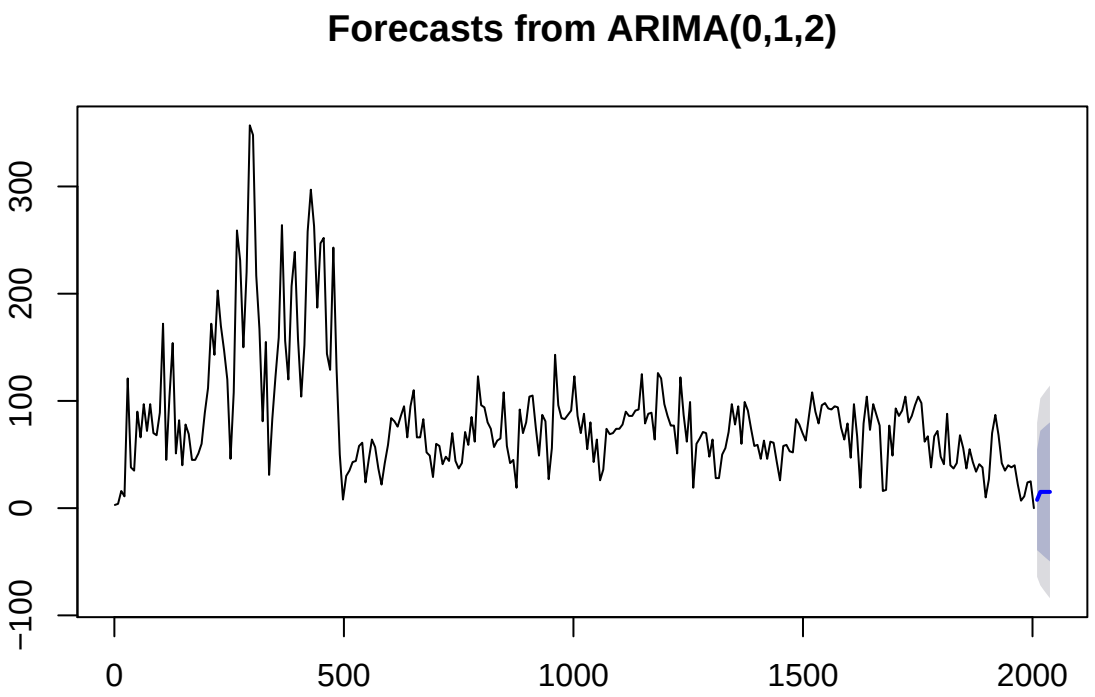
```
fit <- auto.arima(gmts)
fit
```

```
## 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))
```



20.1 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/

Bibliography

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.*

García, S., Luengo, J., and Herrera, F. (2015). Data Preprocessing in Data Mining*. Springer.*

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.*

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).*

Torgo, L. (2010). Data Mining with R: Learning with Case Studies*. Chapman & Hall/CRC, 1st edt. edition.*