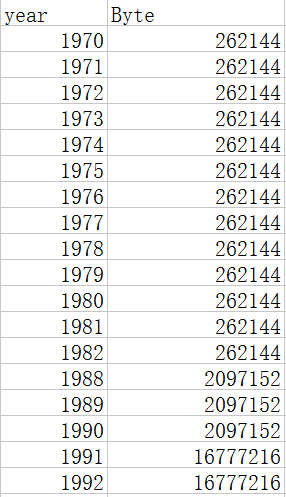
Homework 1

**1. Calculate the increase of memory of PCs over the last 30 years and check whether the FMRI analysis could have been done 20 years ago.**

![~DNYK)B@UP4H{UK5E@](3@A](data:image/png;base64,)

**2. Logistic regression**

In statistics, logistic regression, or logit regression, is a mathematical model used in statistics to estimate the probability of an event occuring having been given some previous data. Logistic Regression works with binary data, where either the event happens (1) or the event does not happen (0). So given some feature x it tries to find out whether some event y happens or not.

For example, if y represents whether a sports team wins a match, then y will be 1 if they win the match or y will be 0 if they do not. This is known as Binomial Logistic Regression. There is also another form of Logistic Regression which uses multiple values for the variable y. This form of Logistic Regression is known as Multinomial Logistic Regression.

Logistic regression does not look at the relationship between the two variables as a straight line. Instead, Logistic regression uses the natural logarithm function to find the relationship between the variables and uses test data to find the coefficients. The function can then predict the future results using these coefficients in the logistic equation.

Logistic regression uses the concept of odds ratios to calculate the probability. This is defined as the ratio of the odds of an event happening to its not happening.



The natural logarithm of the odds ratio is then taken in order to create the logistic equation is know as the logit:



In Logistic regression the Logit of the probability is said to be linear with respect to x, so the logit becomes:



Using the two equations together then gives the following:



Using the two equations together then gives the following:



This then leads to the probability:



The final equation is the logistic regression.

We can use the vector form and the logistic equation can be changed to:



Homework 2

**Code and result:**

################HW2###################

# EX1&EX2

cpu.df = read.csv("byte.csv",header = TRUE)

plot(cpu.df$Byte~cpu.df$year)

splines.reg.l1 = smooth.spline(x = cpu.df$year, y = cpu.df$Byte, spar = 0.2)

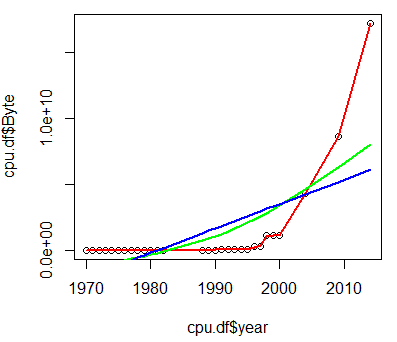
splines.reg.l2 = smooth.spline(x = cpu.df$year, y = cpu.df$Byte, spar = 1)

splines.reg.l3= smooth.spline(x = cpu.df$year, y = cpu.df$Byte, spar = 2)

lines(splines.reg.l1, col = "red", lwd = 2) # regression line with lambda = 0.2

lines(splines.reg.l2, col = "green", lwd = 2) # regression line with lambda = 1

lines(splines.reg.l3, col = "blue", lwd = 2) # regression line with lambda = 2



# EX3

lambda=4

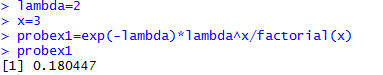
x=6

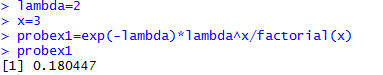
probex1=exp(-lambda)\*lambda^x/factorial(x)

lambda=5

x=0

probex2=exp(-lambda)\*lambda^x/factorial(x)





Homework 3

**Code and result:**

############# HW3 ##############

#1

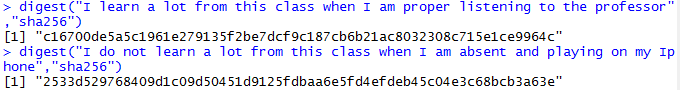
library(digest)

digest("I learn a lot from this class when I am

proper listening to the professor","sha256")

digest("I do not learn a lot from this class when

I am absent and playing on my Iphone","sha256")



#3

library(rjson)

json\_file = "http://crix.hu-berlin.de/data/crix.json"

json\_data = fromJSON(file=json\_file)

x = as.data.frame(json\_data)

date1=c(json\_data[[1]]$date)

for (i in 1:50){

date1[i]=c(json\_data[[i]]$date)

}

price1=c(json\_data[[1]]$price)

for (i in 1:50){

price1[i]=c(json\_data[[i]]$price)

}

date=date1

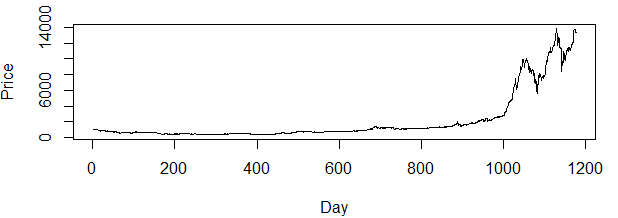
price=price1

crix=data.frame(date,price)

plot(crix$price~as.Date(crix$date))

plot(crix$price~crix$date,type="b")

plot(ts(crix$price,freq=1),type='l',xlab='Day',ylab='Price')

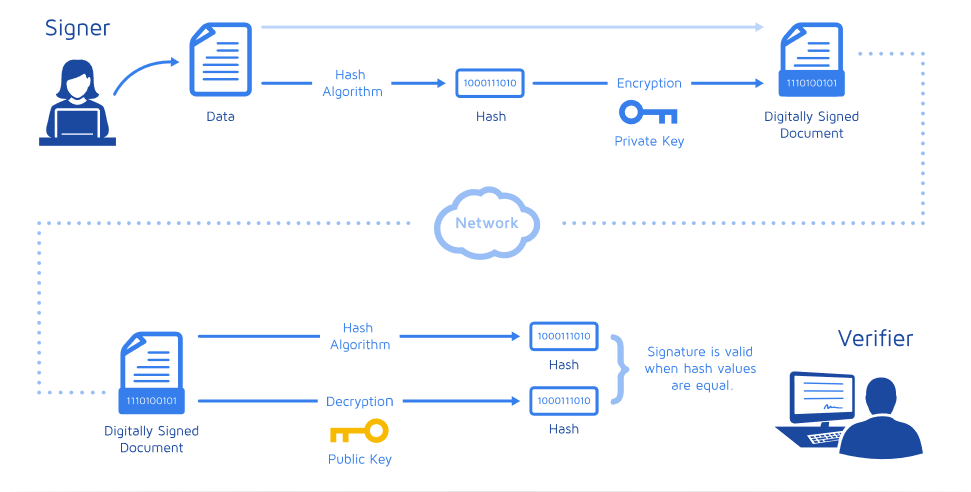


**2. DSA**

The Digital Signature Algorithm (DSA) is a United States Federal Government standard for digital signatures.

DSA was proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their Digital Signature Standard (DSS), specified in FIPS 186.

DSA can be used by the recipient of a message to verify that the message has not been altered during transit as well as as certain the originator’s identity. How do digital signatures work?



The Digital Signature is usually performed in several steps:

1. Calculate the Message Digest(hash-value of the message )

In the first step of the process, a hash-value of the message (often called the message digest) is calculated by applying some cryptographic hashing algorithm

2. Calculate the Digital Signature

In the second step of digitally signing a message, the information obtained in the first step hash-value of the message (the message digest) is encrypted with the private key of the person who signs the message and thus an encrypted hash-value, also called digital signature, is obtained. For this purpose, some mathematical cryptographic encrypting algorithm for calculating digital signatures from given message digest is used, which includes DSA, TSA, ECDSA and so on.

3. Verifying Digital Signatures

The public key is used in the signature verification process to verify the authenticity of the signature

Homework 4

############# HW4 ##############

#1&2

library(rjson)

json\_file = "http://crix.hu-berlin.de/data/crix.json"

json\_data = fromJSON(file=json\_file)

x = as.data.frame(json\_data)

date1=c(json\_data[[1]]$date)

for (i in 1:1177){

date1[i]=c(json\_data[[i]]$date)

}

price1=c(json\_data[[1]]$price)

for (i in 1:1177){

price1[i]=c(json\_data[[i]]$price)

}

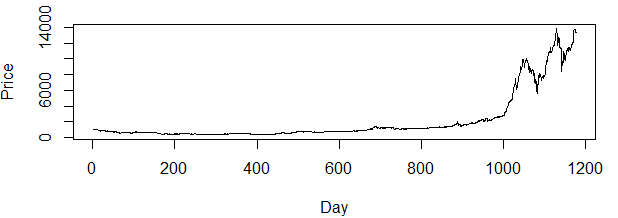
date=date1

price=price1

crix=data.frame(date,price)

##figure 3:crix&ecrix%efcrix

plot(ts(crix$price,freq=1),type='l',xlab='Day',ylab='Price')



library(caschrono)

library(TTR)

library(fGarch)

library(rugarch)

library(forecast)

library(TSA)

#\*\*\*\*\*\*\*\*\*ARIMA medel\*\*\*\*\*\*\*\*\*\*

xy.acfb(crix$price,numer=FALSE)

adf.test(crix$price)

#Augmented Dickey-Fuller Test:not stationary

##\*\*\*\*\*1)return

r=diff(log(crix$price))

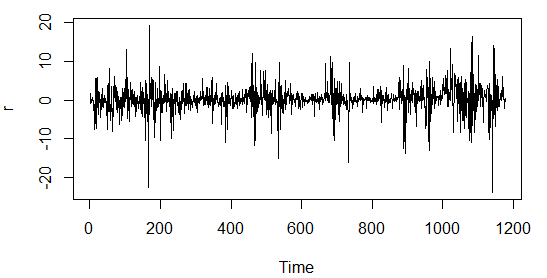
plot(r,type="b")

abline(h = 0)

plot(r,type="l")

#figure4

ts.plot(r)



#figure5

mean(r)

var(r)

sd(r)

hist(r, col = "grey", breaks = 20, freq = FALSE, ylim = c(0, 25),

xlab = "ret")

lines(density(r), lwd = 2)

mu = mean(r)

sigma = sd(r)

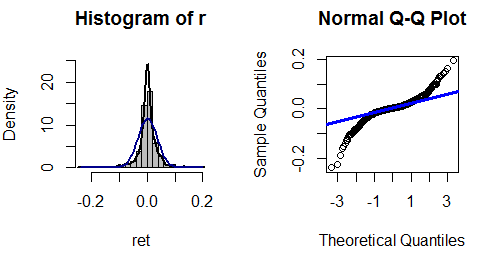
x = seq(-4, 4, length = 100)

curve(dnorm(x, mean = mean(r), sd = sd(r)), add = TRUE,

col = "darkblue", lwd = 2)

qqnorm(r)

qqline(r, col = "blue", lwd = 3)

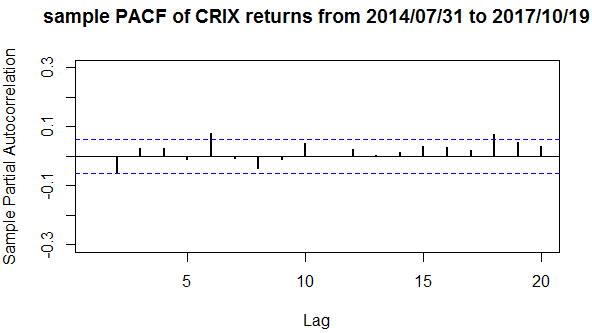


#figure6

libraries = c("zoo", "tseries")

autocorr = acf(r, lag.max = 20, ylab = "Sample Autocorrelation", main = "sample ACF of CRIX returns from 2014/07/31 to 2017/10/19" , lwd = 2, ylim = c(-0.3, 1))

autopcorr = pacf(r, lag.max = 20, ylab = "Sample Partial Autocorrelation", main = "sample PACF of CRIX returns from 2014/07/31 to 2017/10/19" , ylim = c(-0.3, 0.3), lwd = 2)



#figure7

par(mfrow = c(1, 1))

auto.arima(r)

fit202 = arima(r, order = c(2, 0, 2))

tsdiag(fit202)

fit202 = arima(r, order = c(2, 0, 2))

crpre = predict(fit202, n.ahead = 30)

dates = seq(as.Date("31/07/2014", format = "%d/%m/%Y"), by = "days", length = length(ret))

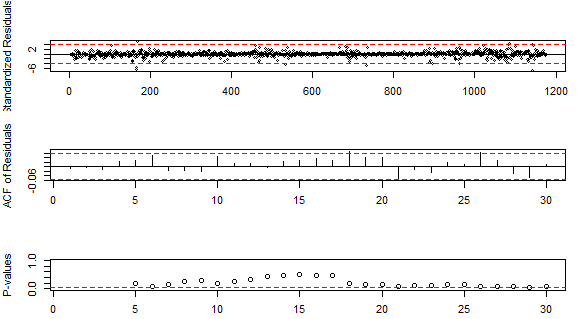
plot(ret, type = "l", ylab = "log return", xlab = "days",

lwd = 1.5, main = "CRIX returns and predicted values")

lines(crpre$pred, col = "red", lwd = 3)

lines(crpre$pred + 2 \* crpre$se, col = "red", lty = 3, lwd = 3)

lines(crpre$pred - 2 \* crpre$se, col = "red", lty = 3, lwd = 3)



Homework 5

**1. Do a word cloud for Shakesspeare's dramas. Romeo and Julia, Julius Caesar, Hamlet.**

rm(list = ls())

#install.packages("RCurl")

#install.packages("XML")

library(RCurl)

library(XML)

url1 = "http://shakespeare.mit.edu/romeo\_juliet/full.html"

url2 = "http://shakespeare.mit.edu/julius\_caesar/full.html"

url3 = "http://shakespeare.mit.edu/hamlet/full.html"

html1 = readLines(url1, encoding = "UTF-8")

html2 = readLines(url2, encoding = "UTF-8")

html3 = readLines(url3, encoding = "UTF-8")

html1 = htmlParse(html1, encoding = "UTF-8")

html2 = htmlParse(html2, encoding = "UTF-8")

html3 = htmlParse(html3, encoding = "UTF-8")

#HW5.1 Wordcloud

#install.packages("bitops")

#install.packages("stringr")

library(bitops)

library(stringr)

abs1 = lapply(url1, FUN = function(x) htmlParse(x, encoding = "Latin-1"))

abs2 = lapply(url2, FUN = function(x) htmlParse(x, encoding = "Latin-1"))

abs3 = lapply(url3, FUN = function(x) htmlParse(x, encoding = "Latin-1"))

clean\_txt = function(x) {

cleantxt = xpathApply(x, "//body//text()

[not(ancestor :: script)][ not(ancestor :: style)]

[not(ancestor :: noscript)] " ,xmlValue)

cleantxt = paste(cleantxt, collapse="\n")

cleantxt = str\_replace\_all(cleantxt, "\n", " ")

cleantxt = str\_replace\_all(cleantxt, "\r", "")

cleantxt = str\_replace\_all(cleantxt, "\t", "")

cleantxt = str\_replace\_all(cleantxt, "<br>", "")

return(cleantxt)

}

cleantxt1 = lapply(abs1,clean\_txt)

cleantxt2 = lapply(abs2,clean\_txt)

cleantxt3 = lapply(abs3,clean\_txt)

vec\_abs1 = unlist(cleantxt1)

vec\_abs2 = unlist(cleantxt2)

vec\_abs3 = unlist(cleantxt3)

###Text Mining

#install.packages("tm")

#install.packages("SnowballC")

library(tm)

library(SnowballC)

abs1 = Corpus(VectorSource(vec\_abs1))

abs2 = Corpus(VectorSource(vec\_abs2))

abs3 = Corpus(VectorSource(vec\_abs3))

abs\_dtm1 = DocumentTermMatrix(abs1, control = list(

stemming = TRUE, stopwords = TRUE, minWordLength = 3,

removeNumbers = TRUE, removePunctuation = TRUE))

abs\_dtm2 = DocumentTermMatrix(abs2, control = list(

stemming = TRUE, stopwords = TRUE, minWordLength = 3,

removeNumbers = TRUE, removePunctuation = TRUE))

abs\_dtm3 = DocumentTermMatrix(abs3, control = list(

stemming = TRUE, stopwords = TRUE, minWordLength = 3,

removeNumbers = TRUE, removePunctuation = TRUE))

##WordCloud

#instal.packages("ggplot2")

#install.packages("wordcloud")

library(ggplot2)

library(wordcloud)

freq1 = colSums(as.matrix(abs\_dtm1))

freq2 = colSums(as.matrix(abs\_dtm2))

freq3 = colSums(as.matrix(abs\_dtm3))

wf1 = data.frame(word=names(freq1), freq=freq1)

wf2 = data.frame(word=names(freq2), freq=freq2)

wf3 = data.frame(word=names(freq3), freq=freq3)

#Romeo and Juliet

plot1 = ggplot(subset(wf1, freq>15), aes(word, freq1))

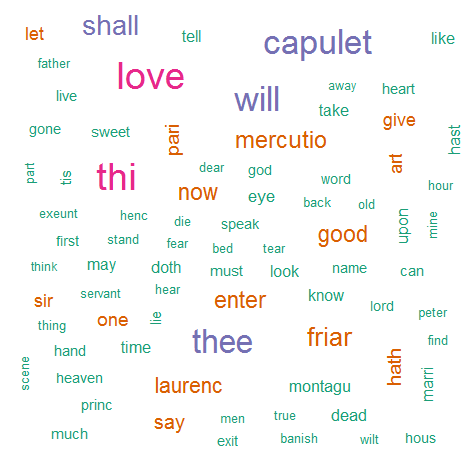
plot1 = plot1 + geom\_bar(stat="identity")

plot1 = plot1 + theme(axis.text.x=element\_text(angle=45, hjust=1))

freq1 = colSums(as.matrix(abs\_dtm1))

dark2\_1 = brewer.pal(6, "Dark2")

wordcloud(names(freq1), freq1, max.words=100, rot.per=0.2, colors=dark2\_1)



#Julius Caeser

plot2 = ggplot(subset(wf2, freq>15), aes(word, freq2))

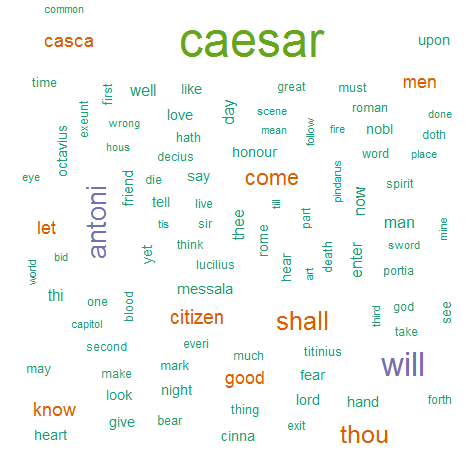
plot2 = plot2 + geom\_bar(stat="identity")

plot2 = plot2 + theme(axis.text.x=element\_text(angle=45, hjust=1))

freq2 = colSums(as.matrix(abs\_dtm2))

dark2\_2 = brewer.pal(6, "Dark2")

wordcloud(names(freq2), freq2, max.words=100, rot.per=0.2, colors=dark2\_2)



#Hamlet

plot3 = ggplot(subset(wf3, freq>15), aes(word, freq3))

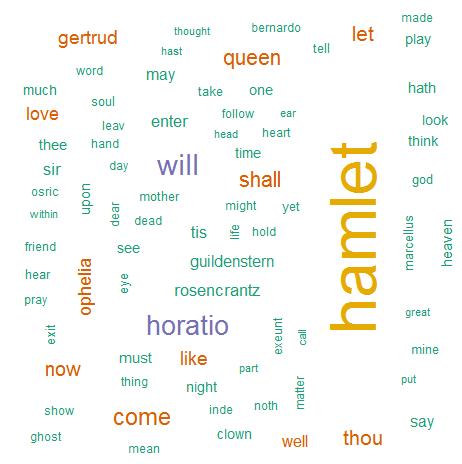
plot3 = plot3 + geom\_bar(stat="identity")

plot3 = plot3 + theme(axis.text.x=element\_text(angle=45, hjust=1))

freq3 = colSums(as.matrix(abs\_dtm3))

dark2\_3 = brewer.pal(6, "Dark2")

wordcloud(names(freq3), freq3, max.words=100, rot.per=0.2, colors=dark2\_3)



1. **Calculate the histogram of words**

#Romeo and Juliet

wf1 <- wf1[order(-wf1$freq),]

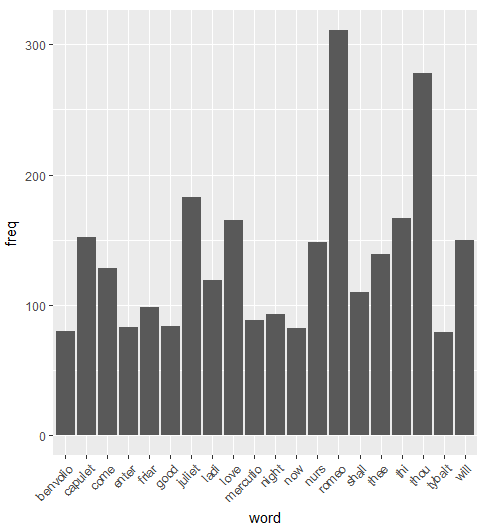
wf1 <- wf1[c(1:20),]

p1 = ggplot(subset(wf1, freq > 15), aes(word, freq))

p1 = p1 + geom\_bar(stat = "identity")

p1 = p1 + theme(axis.text.x = element\_text(angle = 45, hjust = 1))

p1



#Julius Caeser

wf2 <- wf2[order(-wf2$freq),]

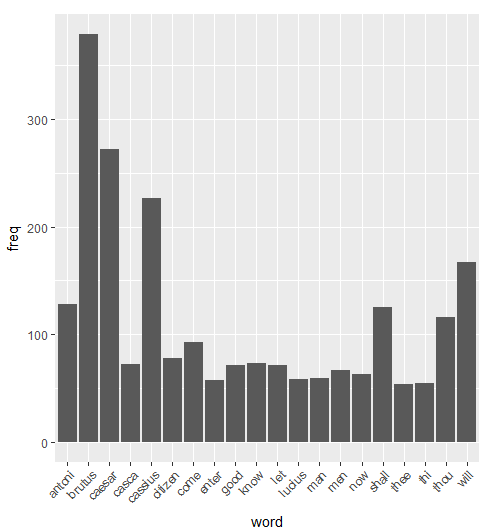
wf2 <- wf2[c(1:20),]

p2 = ggplot(subset(wf2, freq > 15), aes(word, freq))

p2 = p2 + geom\_bar(stat = "identity")

p2 = p2 + theme(axis.text.x = element\_text(angle = 45, hjust = 1))

p2



#Hamlet

wf3 <- wf3[order(-wf3$freq),]

wf3 <- wf3[c(1:20),]

p3 = ggplot(subset(wf3, freq > 15), aes(word, freq))

p3 = p3 + geom\_bar(stat = "identity")

p3 = p3 + theme(axis.text.x = element\_text(angle = 45, hjust = 1))

p3

