Machine Learning basic concepts

Rodrigo Azuero: razuero@iadb.org

February 8, 2019

In this document we will cover the following topics:

1. Linear Regression: Basic concepts
2. Logistic regression: Basic concepts
3. Linear Regression: Advanced topics
4. Logistic Regression: Advanced topics
5. Bayesian classification
6. Maximum Likelihood Estimation
7. Gradient Descent

Data used:

a. WageCSV.csv-> Data with information about wages, education, and ages of individuals in Chile.

b. spam.csv-> Email dataset with information about emails and their spam vs non spam classification

c. EmailCategory.csv-> Second email dataset with information about emails and their spam vs non-spam classification.

1. Linear regression

1.1. Setting up R to run some linear regressions

To change your working directory you can run the code

```
#setwd()
```

The '#' symbol is used as a comment. R will not interpret this as a command unless you remove that symbol.

To check your directory, you can run:

```
getwd()
```

```
## [1] "/Users/rodrigoazuero/Dropbox/BACKUPRODRIGO/MachineLearningClassIDB/Session2"
```

We will clear our environment and all the data stored before every session with the command

```
rm(list = ls())
```

1.1. Exploratory data analysis

Now we will open a dataset containing information about wages, years of schooling, and age of 800 individuals:

```
WageD<-read.table("WageCSV.csv",sep = ",")
WageD<-as.data.frame(WageD)
head(WageD)
```

```
##   V1 V2       V3
## 1 46 12 1.431757
## 2 43 13 1.386294
## 3 41 15 3.180957
```

```
## 4 31 12 1.973354
## 5 30 12 2.484180
## 6 38 12 2.124904
```

Data frame is a format to store data. There are various formats but for the moment we will work with this one.

Assigning the names to the WageD dataset:

```r
colnames(WageD)<-c("age","schooling","wage")
```

and we are ready to start with some calculations and graph with the data. Let's compute the averages and standard deviations of our series:

```r
head(WageD)
```

```
##   age schooling     wage
## 1  46        12 1.431757
## 2  43        13 1.386294
## 3  41        15 3.180957
## 4  31        12 1.973354
## 5  30        12 2.484180
## 6  38        12 2.124904
```

```r
tail(WageD)
```

```
##     age schooling     wage
## 795  23        12 1.878044
## 796  24        12 1.760261
## 797  35        12 2.412586
## 798  29        12 2.165726
## 799  31        12 3.552020
## 800  33        17 2.923412
```

```r
dim(WageD)
```

```
## [1] 800   3
```

```r
nrow(WageD)
```

```
## [1] 800
```

```r
ncol(WageD)
```

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

```r
class(WageD)
```

```
## [1] "data.frame"
```

```r
names(WageD)
```

```
## [1] "age"       "schooling" "wage"
```

```r
#Structure of an object
str(WageD)
```

```
## 'data.frame':    800 obs. of  3 variables:
##  $ age      : int  46 43 41 31 30 38 60 27 28 32 ...
##  $ schooling: int  12 13 15 12 12 12 6 12 12 8 ...
##  $ wage     : num  1.43 1.39 3.18 1.97 2.48 ...
```

```r
#Summary
summary(WageD)
```

```
##       age          schooling          wage        
##  Min.   :20.00   Min.   : 0.00   Min.   :-0.7655  
##  1st Qu.:31.00   1st Qu.: 9.00   1st Qu.: 1.4587  
##  Median :36.00   Median :12.00   Median : 1.8736  
##  Mean   :36.81   Mean   :11.09   Mean   : 1.9304  
##  3rd Qu.:42.00   3rd Qu.:12.00   3rd Qu.: 2.2427  
##  Max.   :60.00   Max.   :20.00   Max.   : 5.1471  
```

```r
#Some additional analysis of the data
#We can ask R to return specifics of the dataset.
#When we execute WageD[i,j] we should obtain the
#element stored in the i-th row and j-th column of the dataframe
#WageD

WageD[2,3]
```

```
## [1] 1.386294
```

```r
WageD[1,]
```

```
##   age schooling     wage
## 1  46        12 1.431757
```

```r
#We can call the columns by their name via two ways:

WageD$age
```

```
##   [1] 46 43 41 31 30 38 60 27 28 32 33 35 27 41 28 44 34 31 42 57 39 37 47
##  [24] 50 34 24 32 33 44 43 31 36 38 39 32 37 37 38 36 33 39 44 45 44 41 33
##  [47] 45 45 35 48 30 29 42 35 35 30 44 38 30 42 46 37 28 41 33 28 36 26 26
##  [70] 48 38 52 49 36 45 38 29 43 47 34 41 42 35 36 43 41 47 34 33 25 42 42
##  [93] 38 27 44 44 33 46 29 31 31 22 45 21 48 39 38 34 40 32 26 36 27 23 36
## [116] 30 39 38 32 37 29 38 30 36 35 27 32 45 44 31 47 33 33 51 32 38 32 34
## [139] 39 29 38 29 30 42 37 35 40 42 26 24 32 51 46 30 31 36 30 36 42 49 36
## [162] 41 30 30 51 33 49 38 37 32 41 35 31 53 49 37 31 26 29 30 35 43 39 49
## [185] 43 40 23 47 51 24 37 38 44 44 34 50 27 41 42 35 29 32 30 31 31 46 40
## [208] 38 37 43 36 37 37 30 33 43 26 40 29 28 38 32 51 55 52 31 28 38 59 32
## [231] 26 46 41 41 28 36 44 32 39 32 32 40 38 47 38 42 38 36 39 41 36 45 47
## [254] 30 39 47 36 34 38 32 54 44 34 41 23 23 49 30 36 39 31 40 28 38 52 46
## [277] 33 40 33 33 43 29 33 35 34 34 27 37 33 43 35 29 31 30 49 46 31 26 40
## [300] 36 43 42 36 31 48 24 37 34 48 34 53 38 56 26 48 42 37 46 29 33 42 38
## [323] 23 32 41 30 37 34 34 38 43 36 39 26 42 42 44 40 30 26 42 48 33 46 41
## [346] 47 44 51 39 41 39 26 40 34 36 46 31 28 40 49 31 37 38 50 35 38 47 34
## [369] 33 31 20 40 49 47 25 39 35 30 36 33 36 27 29 38 35 33 43 35 34 28 39
## [392] 34 42 41 37 36 47 23 27 24 34 34 32 36 42 38 45 26 23 31 37 37 36 56
## [415] 34 24 38 40 33 35 44 56 41 39 30 35 39 27 34 47 38 40 44 36 26 23 37
## [438] 39 32 33 45 50 37 46 33 26 29 28 30 29 33 32 28 23 35 37 39 46 48 27
## [461] 41 42 56 52 48 36 36 33 50 43 35 41 36 40 31 26 49 30 43 26 26 32 39
## [484] 25 37 30 29 30 52 39 30 30 56 34 36 51 40 37 49 38 32 24 34 37 35 21
## [507] 44 36 42 36 42 42 40 37 43 26 42 39 32 58 45 42 35 51 48 44 40 31 25
## [530] 33 38 24 40 27 35 29 42 30 30 29 31 44 39 41 40 43 41 36 41 30 29 30
## [553] 60 48 40 39 38 33 34 32 46 28 22 34 40 38 44 35 34 31 53 35 36 47 23
## [576] 42 31 35 28 40 32 39 36 42 31 50 38 42 49 40 29 34 45 33 24 29 33 21
```

```
## [599] 40 39 40 35 36 47 39 39 48 43 33 39 36 31 49 37 35 43 39 39 36 30 22
## [622] 50 32 37 39 41 33 35 58 31 23 35 48 57 34 31 47 48 41 22 28 49 48 49
## [645] 36 37 38 45 46 26 44 26 45 34 28 39 27 32 37 39 31 30 40 30 30 30 43
## [668] 28 37 30 42 33 38 29 36 37 32 30 26 44 29 45 34 37 37 42 30 39 39 39
## [691] 37 33 41 32 41 34 34 35 45 33 47 47 39 26 51 40 29 34 43 38 37 28 45
## [714] 24 30 38 25 27 37 44 31 32 31 28 43 23 39 36 28 35 32 25 37 35 35 40
## [737] 39 28 40 45 41 39 37 34 33 35 28 39 42 35 35 42 25 34 28 36 32 25 28
## [760] 29 37 29 39 50 46 38 28 38 30 24 27 43 26 45 23 42 37 42 35 33 34 39
## [783] 45 45 49 34 39 31 30 48 52 33 39 52 23 24 35 29 31 33
```

```r
WageD[,"age"]
```

```
##   [1] 46 43 41 31 30 38 60 27 28 32 33 35 27 41 28 44 34 31 42 57 39 37 47
##  [24] 50 34 24 32 33 44 43 31 36 38 39 32 37 37 38 36 33 39 44 45 44 41 33
##  [47] 45 45 35 48 30 29 42 35 35 30 44 38 30 42 46 37 28 41 33 28 36 26 26
##  [70] 48 38 52 49 36 45 38 29 43 47 34 41 42 35 36 43 41 47 34 33 25 42 42
##  [93] 38 27 44 44 33 46 29 31 31 22 45 21 48 39 38 34 40 32 26 36 27 23 36
## [116] 30 39 38 32 37 29 38 30 36 35 27 32 45 44 31 47 33 33 51 32 38 32 34
## [139] 39 29 38 29 30 42 37 35 40 42 26 24 32 51 46 30 31 36 30 36 42 49 36
## [162] 41 30 30 51 33 49 38 37 32 41 35 31 53 49 37 31 26 29 30 35 43 39 49
## [185] 43 40 23 47 51 24 37 38 44 44 34 50 27 41 42 35 29 32 30 31 31 46 40
## [208] 38 37 43 36 37 37 30 33 43 26 40 29 28 38 32 51 55 52 31 28 38 59 32
## [231] 26 46 41 41 28 36 44 32 39 32 32 40 38 47 38 42 38 36 39 41 36 45 47
## [254] 30 39 47 36 34 38 32 54 44 34 41 23 23 49 30 36 39 31 40 28 38 52 46
## [277] 33 40 33 33 43 29 33 35 34 34 27 37 33 43 35 29 31 30 49 46 31 26 40
## [300] 36 43 42 36 31 48 24 37 34 48 34 53 38 56 26 48 42 37 46 29 33 42 38
## [323] 23 32 41 30 37 34 34 38 43 36 39 26 42 42 44 40 30 26 42 48 33 46 41
## [346] 47 44 51 39 41 39 26 40 34 36 46 31 28 40 49 31 37 38 50 35 38 47 34
## [369] 33 31 20 40 49 47 25 39 35 30 36 33 36 27 29 38 35 33 43 35 34 28 39
## [392] 34 42 41 37 36 47 23 27 24 34 34 32 36 42 38 45 26 23 31 37 37 36 56
## [415] 34 24 38 40 33 35 44 56 41 39 30 35 39 27 34 47 38 40 44 36 26 23 37
## [438] 39 32 33 45 50 37 46 33 26 29 28 30 29 33 32 28 23 35 37 39 46 48 27
## [461] 41 42 56 52 48 36 36 33 50 43 35 41 36 40 31 26 49 30 43 26 26 32 39
## [484] 25 37 30 29 30 52 39 30 30 56 34 36 51 40 37 49 38 32 24 34 37 35 21
## [507] 44 36 42 36 42 42 40 37 43 26 42 39 32 58 45 42 35 51 48 44 40 31 25
## [530] 33 38 24 40 27 35 29 42 30 30 29 31 44 39 41 40 43 41 36 41 30 29 30
## [553] 60 48 40 39 38 33 34 32 46 28 22 34 40 38 44 35 34 31 53 35 36 47 23
## [576] 42 31 35 28 40 32 39 36 42 31 50 38 42 49 40 29 34 45 33 24 29 33 21
## [599] 40 39 40 35 36 47 39 39 48 43 33 39 36 31 49 37 35 43 39 39 36 30 22
## [622] 50 32 37 39 41 33 35 58 31 23 35 48 57 34 31 47 48 41 22 28 49 48 49
## [645] 36 37 38 45 46 26 44 26 45 34 28 39 27 32 37 39 31 30 40 30 30 30 43
## [668] 28 37 30 42 33 38 29 36 37 32 30 26 44 29 45 34 37 37 42 30 39 39 39
## [691] 37 33 41 32 41 34 34 35 45 33 47 47 39 26 51 40 29 34 43 38 37 28 45
## [714] 24 30 38 25 27 37 44 31 32 31 28 43 23 39 36 28 35 32 25 37 35 35 40
## [737] 39 28 40 45 41 39 37 34 33 35 28 39 42 35 35 42 25 34 28 36 32 25 28
## [760] 29 37 29 39 50 46 38 28 38 30 24 27 43 26 45 23 42 37 42 35 33 34 39
## [783] 45 45 49 34 39 31 30 48 52 33 39 52 23 24 35 29 31 33
```

```r
#

WageD[1:10,]
```

```
##    age schooling     wage
## 1   46        12 1.431757
## 2   43        13 1.386294
```

```
## 3   41         15 3.180957
## 4   31         12 1.973354
## 5   30         12 2.484180
## 6   38         12 2.124904
## 7   60          6 1.326396
## 8   27         12 1.654900
## 9   28         12 2.417040
## 10  32          8 1.378268
```

```r
WageD[,1]
```

```
##   [1] 46 43 41 31 30 38 60 27 28 32 33 35 27 41 28 44 34 31 42 57 39 37 47
##  [24] 50 34 24 32 33 44 43 31 36 38 39 32 37 37 38 36 33 39 44 45 44 41 33
##  [47] 45 45 35 48 30 29 42 35 35 30 44 38 30 42 46 37 28 41 33 28 36 26 26
##  [70] 48 38 52 49 36 45 38 29 43 47 34 41 42 35 36 43 41 47 34 33 25 42 42
##  [93] 38 27 44 44 33 46 29 31 31 22 45 21 48 39 38 34 40 32 26 36 27 23 36
## [116] 30 39 38 32 37 29 38 30 36 35 27 32 45 44 31 47 33 33 51 32 38 32 34
## [139] 39 29 38 29 30 42 37 35 40 42 26 24 32 51 46 30 31 36 30 36 42 49 36
## [162] 41 30 30 51 33 49 38 37 32 41 35 31 53 49 37 31 26 29 30 35 43 39 49
## [185] 43 40 23 47 51 24 37 38 44 44 34 50 27 41 42 35 29 32 30 31 31 46 40
## [208] 38 37 43 36 37 37 30 33 43 26 40 29 28 38 32 51 55 52 31 28 38 59 32
## [231] 26 46 41 41 28 36 44 32 39 32 32 40 38 47 38 42 38 36 39 41 36 45 47
## [254] 30 39 47 36 34 38 32 54 44 34 41 23 23 49 30 36 39 31 40 28 38 52 46
## [277] 33 40 33 33 43 29 33 35 34 34 27 37 33 43 35 29 31 30 49 46 31 26 40
## [300] 36 43 42 36 31 48 24 37 34 48 34 53 38 56 26 48 42 37 46 29 33 42 38
## [323] 23 32 41 30 37 34 34 38 43 36 39 26 42 42 44 40 30 26 42 48 33 46 41
## [346] 47 44 51 39 41 39 26 40 34 36 46 31 28 40 49 31 37 38 50 35 38 47 34
## [369] 33 31 20 40 49 47 25 39 35 30 36 33 36 27 29 38 35 33 43 35 34 28 39
## [392] 34 42 41 37 36 47 23 27 24 34 34 32 36 42 38 45 26 23 31 37 37 36 56
## [415] 34 24 38 40 33 35 44 56 41 39 30 35 39 27 34 47 38 40 44 36 26 23 37
## [438] 39 32 33 45 50 37 46 33 26 29 28 30 29 33 32 28 23 35 37 39 46 48 27
## [461] 41 42 56 52 48 36 36 33 50 43 35 41 36 40 31 26 49 30 43 26 26 32 39
## [484] 25 37 30 29 30 52 39 30 30 56 34 36 51 40 37 49 38 32 24 34 37 35 21
## [507] 44 36 42 36 42 42 40 37 43 26 42 39 32 58 45 42 35 51 48 44 40 31 25
## [530] 33 38 24 40 27 35 29 42 30 30 29 31 44 39 41 40 43 41 36 41 30 29 30
## [553] 60 48 40 39 38 33 34 32 46 28 22 34 40 38 44 35 34 31 53 35 36 47 23
## [576] 42 31 35 28 40 32 39 36 42 31 50 38 42 49 40 29 34 45 33 24 29 33 21
## [599] 40 39 40 35 36 47 39 39 48 43 33 39 36 31 49 37 35 43 39 39 36 30 22
## [622] 50 32 37 39 41 33 35 58 31 23 35 48 57 34 31 47 48 41 22 28 49 48 49
## [645] 36 37 38 45 46 26 44 26 45 34 28 39 27 32 37 39 31 30 40 30 30 30 43
## [668] 28 37 30 42 33 38 29 36 37 32 30 26 44 29 45 34 37 37 42 30 39 39 39
## [691] 37 33 41 32 41 34 34 35 45 33 47 47 39 26 51 40 29 34 43 38 37 28 45
## [714] 24 30 38 25 27 37 44 31 32 31 28 43 23 39 36 28 35 32 25 37 35 35 40
## [737] 39 28 40 45 41 39 37 34 33 35 28 39 42 35 35 42 25 34 28 36 32 25 28
## [760] 29 37 29 39 50 46 38 28 38 30 24 27 43 26 45 23 42 37 42 35 33 34 39
## [783] 45 45 49 34 39 31 30 48 52 33 39 52 23 24 35 29 31 33
```

```r
head(WageD$age)
```

```
## [1] 46 43 41 31 30 38
```

```r
head(WageD[,"age"])
```

```
## [1] 46 43 41 31 30 38
```

```r
WageD[WageD$age>55,]
```

```
##     age schooling      wage
## 7    60         6 1.3263960
## 20   57        17 2.8870440
## 229  59        17 2.7535130
## 313  56         8 0.9617532
## 414  56         6 1.3857130
## 422  56        15 1.2902570
## 463  56         8 1.6957220
## 493  56         6 1.5495400
## 520  58        13 2.6357300
## 553  60        10 1.7194390
## 629  58        18 2.4534080
## 634  57        14 3.5520200
```

```
subset(WageD,schooling<3)
```

```
##     age schooling      wage
## 15   28         0 1.6957220
## 61   46         2 2.3480470
## 113  27         2 1.3184280
## 278  40         0 1.4428070
## 316  42         2 0.7386096
## 464  52         0 1.0262920
## 616  43         0 3.5127590
```

```
mean(WageD$wage)
```

```
## [1] 1.930354
```

```
sd(WageD$wage)
```

```
## [1] 0.7232198
```

```
mean(WageD$age)
```

```
## [1] 36.80625
```

```
sd(WageD$age)
```

```
## [1] 7.487409
```

```
mean(WageD$schooling)
```

```
## [1] 11.08875
```

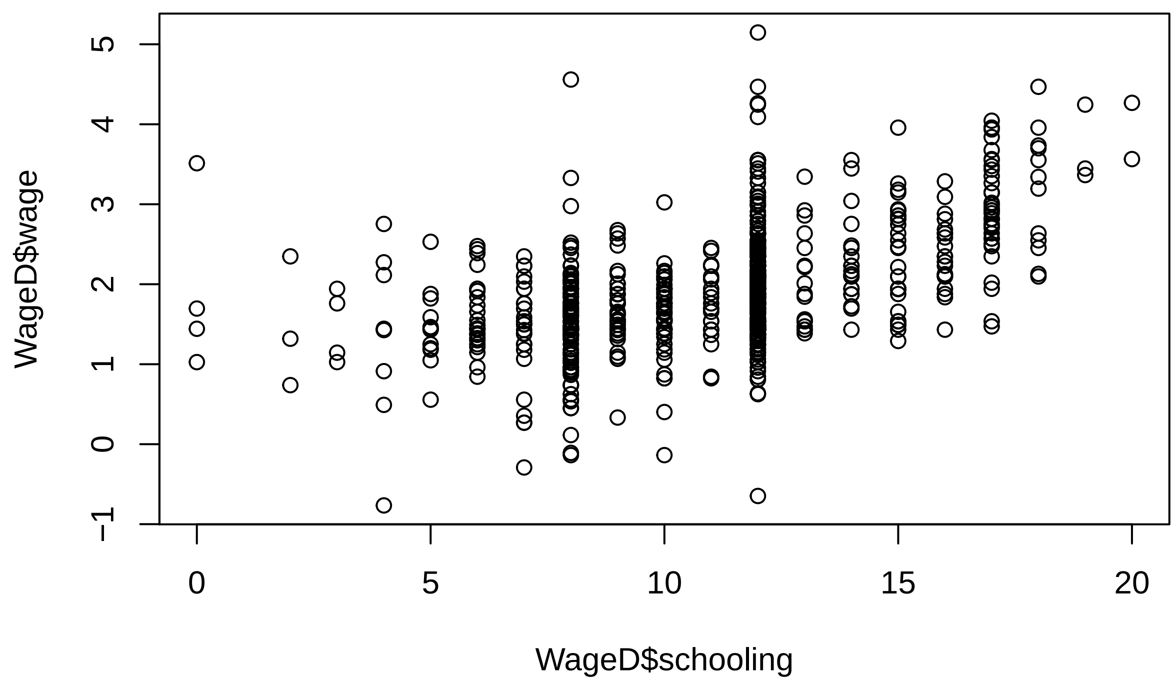
```
sd(WageD$schooling)
```

```
## [1] 3.190784
```

Let's plot some relationships in the data.

```
hist(WageD$age)
```


```
plot(WageD$schooling,WageD$wage)
```



1.3. Basic concepts of linear regression

The goal of OLS is to fit a linear function to the data by minimizing the sum of squared errors. Suppose we have data on log-wages, denoted by y_i, for n individuals. We also have information about education $(x_{1,i})$ and age $(x_{2,i})$ and we want to analyze the relationship between education, age, and wages. For each

individual, our predicted wage
will be:

$$\hat{y}_i = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} \tag{1}$$

The goal in OLS is to minimize the sum of squared errors. That is, we want to find $\beta_0, \beta_1, \beta_2$ that minimize the following loss function defined in Equation 2

$$SSR(\beta_0, \beta_1, \beta_2) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$= \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{1,i} - \beta_2 x_{2,i})^2 \tag{2}$$

We can use the lm function in R to obtain estimates of our parameters minimizing the SSR. These will be our linear regression coefficients:

```r
mod<-lm(wage~ age+schooling, data=WageD)

#let us see what we have stored in 'mod'
summary(mod)
```

```
##
## Call:
## lm(formula = wage ~ age + schooling, data = WageD)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7413 -0.3660 -0.0476  0.3095  3.1875
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.343963   0.137503   2.501  0.01257 *
## age         0.007881   0.002933   2.687  0.00735 **
## schooling   0.116906   0.006882  16.988  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6196 on 797 degrees of freedom
## Multiple R-squared:  0.2678, Adjusted R-squared:  0.266
## F-statistic: 145.8 on 2 and 797 DF,  p-value: < 2.2e-16
```

```r
#Mod stores a series of objects with different names. We can see the list of names
#with the 'names' function:
names(summary(mod))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```r
#We can call each object with the '$' sign:
summary(mod)$coefficients
```

```
##                Estimate  Std. Error   t value     Pr(>|t|)
## (Intercept) 0.343963408 0.137502893  2.501499 1.256636e-02
```

```
## age         0.007880537 0.002932586  2.687231 7.354874e-03
## schooling   0.116905614 0.006881529 16.988321 1.792663e-55
```

```r
summary(mod)$call
```

```
## lm(formula = wage ~ age + schooling, data = WageD)
```

```r
#Do you want to know more about the 'lm'  command?
?lm
```

We can see the predicted coefficients

```r
print(coef(summary(mod)))
```

```
##                Estimate  Std. Error   t value     Pr(>|t|)
## (Intercept) 0.343963408 0.137502893  2.501499 1.256636e-02
## age         0.007880537 0.002932586  2.687231 7.354874e-03
## schooling   0.116905614 0.006881529 16.988321 1.792663e-55
```

And see the predicted values.

```r
head(predict(mod))
```

```
##        1        2        3        4        5        6
## 2.109335 2.202599 2.420650 1.991127 1.983247 2.046291
```

```r
head(WageD$wage)
```

```
## [1] 1.431757 1.386294 3.180957 1.973354 2.484180 2.124904
```

And if we want to predict based on this model with new data:

```r
newdata <- data.frame(age=32,schooling=12)
predict(mod,newdata)
```

```
##        1
## 1.999008
```

3. Classification-Logistic regression: I.

Let us see a basic problem of classification. We have a dataset containing spam and non-spam emails with some characteristics of those emails. Let us inspect the dataset.

```r
email<- read.csv("spam.csv")
email<-data.frame(email)
head(email)
```

```
##   word_freq_make word_freq_address word_freq_all word_freq_3d
## 1           0.00              0.64          0.64            0
## 2           0.21              0.28          0.50            0
## 3           0.06              0.00          0.71            0
## 4           0.00              0.00          0.00            0
## 5           0.00              0.00          0.00            0
## 6           0.00              0.00          0.00            0
##   word_freq_our word_freq_over word_freq_remove word_freq_internet
## 1          0.32           0.00             0.00               0.00
## 2          0.14           0.28             0.21               0.07
## 3          1.23           0.19             0.19               0.12
## 4          0.63           0.00             0.31               0.63
## 5          0.63           0.00             0.31               0.63
```

```
## 6         1.85          0.00             0.00                1.85
##   word_freq_order word_freq_mail word_freq_receive word_freq_will
## 1            0.00           0.00              0.00           0.64
## 2            0.00           0.94              0.21           0.79
## 3            0.64           0.25              0.38           0.45
## 4            0.31           0.63              0.31           0.31
## 5            0.31           0.63              0.31           0.31
## 6            0.00           0.00              0.00           0.00
##   word_freq_people word_freq_report word_freq_addresses word_freq_free
## 1             0.00             0.00                0.00           0.32
## 2             0.65             0.21                0.14           0.14
## 3             0.12             0.00                1.75           0.06
## 4             0.31             0.00                0.00           0.31
## 5             0.31             0.00                0.00           0.31
## 6             0.00             0.00                0.00           0.00
##   word_freq_business word_freq_email word_freq_you word_freq_credit
## 1               0.00            1.29          1.93             0.00
## 2               0.07            0.28          3.47             0.00
## 3               0.06            1.03          1.36             0.32
## 4               0.00            0.00          3.18             0.00
## 5               0.00            0.00          3.18             0.00
## 6               0.00            0.00          0.00             0.00
##   word_freq_your word_freq_font word_freq_000 word_freq_money word_freq_hp
## 1           0.96              0          0.00            0.00            0
## 2           1.59              0          0.43            0.43            0
## 3           0.51              0          1.16            0.06            0
## 4           0.31              0          0.00            0.00            0
## 5           0.31              0          0.00            0.00            0
## 6           0.00              0          0.00            0.00            0
##   word_freq_hpl word_freq_george word_freq_650 word_freq_lab
## 1             0                0             0             0
## 2             0                0             0             0
## 3             0                0             0             0
## 4             0                0             0             0
## 5             0                0             0             0
## 6             0                0             0             0
##   word_freq_labs word_freq_telnet word_freq_857 word_freq_data
## 1              0                0             0              0
## 2              0                0             0              0
## 3              0                0             0              0
## 4              0                0             0              0
## 5              0                0             0              0
## 6              0                0             0              0
##   word_freq_415 word_freq_85 word_freq_technology word_freq_1999
## 1             0            0                    0           0.00
## 2             0            0                    0           0.07
## 3             0            0                    0           0.00
## 4             0            0                    0           0.00
## 5             0            0                    0           0.00
## 6             0            0                    0           0.00
##   word_freq_parts word_freq_pm word_freq_direct word_freq_cs
## 1               0            0             0.00            0
## 2               0            0             0.00            0
## 3               0            0             0.06            0
```

```
## 4               0             0             0.00             0
## 5               0             0             0.00             0
## 6               0             0             0.00             0
##   word_freq_meeting word_freq_original word_freq_project word_freq_re
## 1                 0               0.00                 0         0.00
## 2                 0               0.00                 0         0.00
## 3                 0               0.12                 0         0.06
## 4                 0               0.00                 0         0.00
## 5                 0               0.00                 0         0.00
## 6                 0               0.00                 0         0.00
##   word_freq_edu word_freq_table word_freq_conference char_freq_semicolon
## 1          0.00               0                    0                0.00
## 2          0.00               0                    0                0.00
## 3          0.06               0                    0                0.01
## 4          0.00               0                    0                0.00
## 5          0.00               0                    0                0.00
## 6          0.00               0                    0                0.00
##   char_freq_leftbrac char_freq_leftsquarebrac char_freq_exclaim
## 1              0.000                        0             0.778
## 2              0.132                        0             0.372
## 3              0.143                        0             0.276
## 4              0.137                        0             0.137
## 5              0.135                        0             0.135
## 6              0.223                        0             0.000
##   char_freq_dollar char_freq_pound capital_run_length_average
## 1            0.000           0.000                      3.756
## 2            0.180           0.048                      5.114
## 3            0.184           0.010                      9.821
## 4            0.000           0.000                      3.537
## 5            0.000           0.000                      3.537
## 6            0.000           0.000                      3.000
##   capital_run_length_longest capital_run_length_total spam
## 1                         61                      278    1
## 2                        101                     1028    1
## 3                        485                     2259    1
## 4                         40                      191    1
## 5                         40                      191    1
## 6                         15                       54    1
```

```
Spam <- subset(email, spam == 1)
NoSpam <- subset(email, spam == 0)

mean(Spam$word_freq_make)
```

```
## [1] 0.1523387
```

```
mean(NoSpam$word_freq_make)
```

```
## [1] 0.0734792
```

We want to predict the probability that an email is spam or non-spam based on the characteristics. If we were to apply ordinarly least squares, we would predict probabilites that are greater than one or less than zero.

```
mod<-lm(spam~ ., email)
print(coef(summary(mod)))
```

```
##                                   Estimate   Std. Error     t value
```

```
## (Intercept)                2.002786e-01 1.151670e-02 17.39027603
## word_freq_make            -4.981903e-02 1.678205e-02 -2.96858964
## word_freq_address         -1.204624e-02 3.790254e-03 -3.17821321
## word_freq_all              3.927858e-02 1.005551e-02  3.90617367
## word_freq_3d               1.191729e-02 3.460758e-03  3.44354827
## word_freq_our              8.420772e-02 7.570049e-03 11.12380096
## word_freq_over             1.188405e-01 1.842126e-02  6.45127164
## word_freq_remove           2.129405e-01 1.303034e-02 16.34190749
## word_freq_internet         9.398890e-02 1.256983e-02  7.47734086
## word_freq_order            7.247403e-02 1.896365e-02  3.82173435
## word_freq_mail             1.506740e-02 7.779864e-03  1.93671712
## word_freq_receive          5.685672e-02 2.623225e-02  2.16743606
## word_freq_will            -2.785949e-02 5.900618e-03 -4.72145331
## word_freq_people           1.190404e-02 1.667835e-02  0.71374200
## word_freq_report           4.859980e-03 1.473798e-02  0.32975882
## word_freq_addresses        1.852452e-02 2.196416e-02  0.84339772
## word_freq_free             7.506136e-02 6.024203e-03 12.45996544
## word_freq_business         5.171571e-02 1.186399e-02  4.35905001
## word_freq_email            5.539792e-02 9.763122e-03  5.67420094
## word_freq_you              1.413337e-02 3.103227e-03  4.55441058
## word_freq_credit           6.172198e-02 9.840262e-03  6.27239142
## word_freq_your             5.269373e-02 4.663475e-03 11.29924101
## word_freq_font             4.476700e-02 5.345677e-03  8.37442969
## word_freq_000              1.748004e-01 1.590924e-02 10.98734934
## word_freq_money            9.089079e-02 1.142815e-02  7.95323935
## word_freq_hp              -2.317497e-02 3.655305e-03 -6.34009201
## word_freq_hpl             -2.162895e-02 6.735746e-03 -3.21107027
## word_freq_george          -1.220160e-02 1.508024e-03 -8.09112110
## word_freq_650              3.987416e-03 1.267117e-02  0.31468418
## word_freq_lab             -7.449544e-03 1.118318e-02 -0.66613856
## word_freq_labs            -5.194838e-02 1.609822e-02 -3.22696508
## word_freq_telnet          -2.329398e-02 1.939458e-02 -1.20105585
## word_freq_857              6.331760e-03 1.686705e-01  0.03753924
## word_freq_data            -4.198342e-02 8.845309e-03 -4.74640451
## word_freq_415              5.114177e-02 1.660001e-01  0.30808280
## word_freq_85              -3.116872e-02 1.221121e-02 -2.55246779
## word_freq_technology       2.648006e-02 1.962524e-02  1.34928637
## word_freq_1999            -3.321252e-02 1.271459e-02 -2.61215750
## word_freq_parts           -5.343861e-02 2.247231e-02 -2.37797614
## word_freq_pm              -1.975460e-02 1.171643e-02 -1.68605905
## word_freq_direct           4.076089e-02 2.723716e-02  1.49651783
## word_freq_cs              -8.364086e-03 1.438289e-02 -0.58153042
## word_freq_meeting         -3.692652e-02 7.602841e-03 -4.85693702
## word_freq_original        -6.323898e-02 2.356276e-02 -2.68385327
## word_freq_project         -3.238003e-02 7.863416e-03 -4.11780726
## word_freq_re              -3.525328e-02 4.958248e-03 -7.11002819
## word_freq_edu             -3.781411e-02 5.776719e-03 -6.54594958
## word_freq_table           -1.951774e-01 6.344529e-02 -3.07631016
## word_freq_conference      -5.822294e-02 1.696589e-02 -3.43176479
## char_freq_semicolon       -1.401025e-01 2.204915e-02 -6.35410003
## char_freq_leftbrac        -5.995936e-02 2.231302e-02 -2.68719127
## char_freq_leftsquarebrac  -5.905158e-02 4.487715e-02 -1.31584952
## char_freq_exclaim          6.805299e-02 6.136288e-03 11.09025330
## char_freq_dollar           2.331780e-01 2.170978e-02 10.74068781
```

```
## char_freq_pound            2.769343e-02 1.161990e-02  2.38327556
## capital_run_length_average  2.326708e-04 1.831339e-04  1.27049544
## capital_run_length_longest  6.675382e-05 3.712622e-05  1.79802357
## capital_run_length_total    7.986219e-05 9.656463e-06  8.27033533
##                                Pr(>|t|)
## (Intercept)                1.251766e-65
## word_freq_make             3.007324e-03
## word_freq_address          1.491800e-03
## word_freq_all              9.513144e-05
## word_freq_3d               5.793221e-04
## word_freq_our              2.227934e-28
## word_freq_over             1.224892e-10
## word_freq_remove           2.233620e-58
## word_freq_internet         9.049893e-14
## word_freq_order            1.342885e-04
## word_freq_mail             5.284186e-02
## word_freq_receive          3.025340e-02
## word_freq_will             2.412158e-06
## word_freq_people           4.754234e-01
## word_freq_report           7.415974e-01
## word_freq_addresses        3.990504e-01
## word_freq_free             4.591060e-35
## word_freq_business         1.335155e-05
## word_freq_email            1.479927e-08
## word_freq_you              5.390798e-06
## word_freq_credit           3.886214e-10
## word_freq_your             3.237116e-29
## word_freq_font             7.309433e-17
## word_freq_000              9.795535e-28
## word_freq_money            2.275476e-15
## word_freq_hp               2.519458e-10
## word_freq_hpl              1.331632e-03
## word_freq_george           7.520722e-16
## word_freq_650              7.530159e-01
## word_freq_lab              5.053564e-01
## word_freq_labs             1.259987e-03
## word_freq_telnet           2.297922e-01
## word_freq_857              9.700567e-01
## word_freq_data             2.134281e-06
## word_freq_415              7.580335e-01
## word_freq_85               1.072871e-02
## word_freq_technology       1.773122e-01
## word_freq_1999             9.026897e-03
## word_freq_parts            1.744908e-02
## word_freq_pm               9.185309e-02
## word_freq_direct           1.345882e-01
## word_freq_cs               5.609119e-01
## word_freq_meeting          1.232234e-06
## word_freq_original         7.304297e-03
## word_freq_project          3.892892e-05
## word_freq_re               1.341339e-12
## word_freq_edu              6.566721e-11
## word_freq_table            2.108269e-03
## word_freq_conference       6.050173e-04
```

```
## char_freq_semicolon        2.302109e-10
## char_freq_leftbrac         7.231816e-03
## char_freq_leftsquarebrac   1.882909e-01
## char_freq_exclaim          3.211481e-28
## char_freq_dollar           1.363955e-26
## char_freq_pound            1.720020e-02
## capital_run_length_average 2.039733e-01
## capital_run_length_longest 7.223964e-02
## capital_run_length_total   1.736491e-16
```

```
T<-predict(mod, email[,1:57])
min(T)
```

```
## [1] -0.6515491
```

```
max(T)
```

```
## [1] 2.270832
```

```
mean(T)
```

```
## [1] 0.3940448
```

```
sd(T)
```

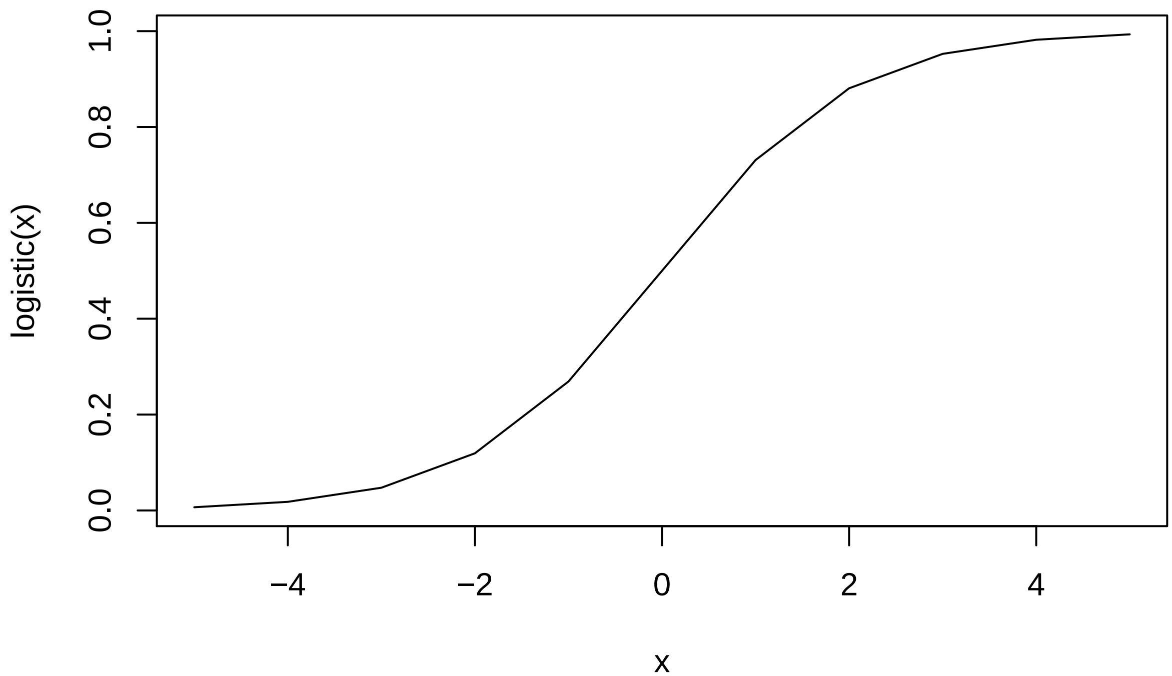
```
## [1] 0.3656871
```

Often in classification problems, to prevent having predictions larger than one or smaller than zero, we use logistic regression. In logistic regression, we assume that the probability is defined according to the function:

$$P(y_i = 1) = \frac{1}{1 + e^{-x_i\beta}} \tag{3}$$

This parametric form guarantees that we have probabilities between zero and one. Let us see how it looks:

```
x<-seq(-5,5,1)
logistic<-function(z){
  ans<-1/(1+exp(-z))
  return(ans)
}
plot(x,logistic(x),type="l")
```



Now, how do we fit a logistic regression model into our data? We want to predict probabilities for spam and non-spam emails. This means that, ideally, in our model, we want very large probabilities predicted for spam emails, and very low probabilities for non-spam emails. Let us see an intuition of how our cost function should look like.

We want $\frac{1}{1+e^{-x_i\beta}} \to 1$ if $y_i = 1$.

For those that are non-spam $y_i = 0$, we want $\frac{1}{1+e^{-x_i\beta}} \to 0$. Similarly, we want $1 - \frac{1}{1+e^{-x_i\beta}} \to 1$.

We can use the GLM function in R to obtain the estimates of logistic regression. We will compare a linear regressor and a logistic regression for a classification problem with the regular Machine Learning workflow.

The first step is split the data into training and testing. For such a purpose, we will use the packate 'caTools'.

```
if (!require("caTools")) install.packages("caTools")
```

```
## Loading required package: caTools
```

```
library(caTools)
set.seed(241567)
email$sample<-sample.split(email$spam,SplitRatio=0.8)
emailtrain=subset(email, email$sample==TRUE)
emailtest=subset(email, email$sample==FALSE)

#Removing the last column (sample)
email$sample<-NULL
emailtrain$sample<-NULL
emailtest$sample<-NULL
```

```
#Logistic regression
LogitModel <- glm(spam ~ ., data=emailtrain, family="binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
bLogit <- coef(LogitModel)
```

```r
#Linear regression
LinearRegressionModel <- lm(spam ~ ., data=emailtrain)
bLinearRegression <- coef(LinearRegressionModel)

#Predictions in logistic regression
PredictProbabilitiesLogistic <- predict(LogitModel, newdata=emailtest, type="response")
PredictProbabilitiesLinear <- predict(LinearRegressionModel, newdata=emailtest,type="response")

#Predicted probabilities in test data
PredictionsLogistic<-PredictProbabilitiesLogistic>0.5
PredictionsLogistic<-as.numeric(PredictionsLogistic)
TLogistic<-table(PredictionsLogistic,emailtest$spam)
TLogistic
```

```
##
## PredictionsLogistic   0   1
##                   0 524  42
##                   1  34 321
```

```r
colnames(TLogistic)<-c("Predicted non-spam","Predicted Spam")
rownames(TLogistic)<-c("Non-spam","Spam")
TLogistic
```

```
##
## PredictionsLogistic Predicted non-spam Predicted Spam
##            Non-spam                524             42
##            Spam                     34            321
```

```r
D<-dim(emailtest)
TLogistic<-100*(TLogistic/D[1])
TLogistic
```

```
##
## PredictionsLogistic Predicted non-spam Predicted Spam
##            Non-spam          56.894680       4.560261
##            Spam               3.691640      34.853420
```

```r
TLogistic[1,1]+TLogistic[2,2]
```

```
## [1] 91.7481
```

```r
TLogistic
```

```
##
## PredictionsLogistic Predicted non-spam Predicted Spam
##            Non-spam          56.894680       4.560261
##            Spam               3.691640      34.853420
```

```r
PredictProbabilitiesLinear<-PredictProbabilitiesLinear>0.5
PredictProbabilitiesLinear<-as.numeric(PredictProbabilitiesLinear)
TLinear<-table(PredictProbabilitiesLinear,emailtest$spam)
TLinear
```

```
##
## PredictProbabilitiesLinear   0   1
##                          0 525  85
##                          1  33 278
```

```r
colnames(TLinear)<-c("Predicted non-spam","Predicted Spam")
rownames(TLinear)<-c("Non-spam","Spam")
D<-dim(emailtest)
TLinear<-100*(TLinear/D[1])
TLinear
```

```
##
## PredictProbabilitiesLinear Predicted non-spam Predicted Spam
##                   Non-spam          57.003257       9.229099
##                   Spam               3.583062      30.184582
```

```r
TLinear[1,1]+TLinear[2,2]
```

```
## [1] 87.18784
```

```r
TLinear
```

```
##
## PredictProbabilitiesLinear Predicted non-spam Predicted Spam
##                   Non-spam          57.003257       9.229099
##                   Spam               3.583062      30.184582
```

We predict accurately 92% of emails in logistic regression and 87% in linear regression. In logistic regression 5% of predicted spam emails are non-spam (type I error) whereas 4% of spam emails are not identified correctly (type II error). Some measures commonly used in Machine Learning to evaluate the performance of an algorithm are precission, recall, and accuracy.

$$\text{Precission} = \frac{\text{True positive}}{\text{True Positive+False Positive}} \tag{4}$$

$$\text{Recall} = \frac{\text{True positive}}{\text{True Positive+False Negative}} \tag{5}$$

$$\text{Accuracy} = \frac{\text{True positive+true negative}}{\text{All}} \tag{6}$$

```r
#Precission:
TLogistic[2,2]/(TLogistic[2,2]+TLogistic[1,2])
```

```
## [1] 0.8842975
```

```r
#Recall
TLogistic[2,2]/(TLogistic[2,2]+TLogistic[2,1])
```

```
## [1] 0.9042254
```

```r
##Accuracy
TLogistic[2,2]+TLogistic[1,1]
```

```
## [1] 91.7481
```

Linear Regression: II

Now that we know the basics of R programming language and how to derive the coefficient estimates for OLS, we will code our own OLS function. Recall, we have information about log-wages (y_i) that we can store in a vector:

$$Y_{n\times 1} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{7}$$

And similarly, we can store the information about schooling and age in vectors$(X_1)_{n\times 1}$ for schooling, and $(X_2)_{n\times 1}$ for age.

We will skip the derivation of the parameters $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$[1] but it can be shown that:

$$\hat{\beta} = (X'X)^{-1}(X'Y) \tag{8}$$

where:

$$X_{n\times 3} = [X_0, X_1, X_2] \tag{9}$$

$$(X_0)_{n\times 1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{10}$$

$$\hat{\beta} = \left[\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2\right] \tag{11}$$

We can code this function in R as follows:

```
    n<-length(WageD$wage)
    X0<-rep(1,n)
    X1<-WageD$age
    X2<-WageD$schooling
    Y<-WageD$wage

    X<-cbind(X0,X1,X2)

    Beta<-(solve(t(X)%*%X))%*%(t(X)%*%Y)
    Beta0<-Beta[1]
    Beta1<-Beta[2]
    Beta2<-Beta[3]
```

Compare your results with the coefficients obtained directly from the *lm* function in R. We will not go through the derivation of the standard errors of the linear regression model.

Now we will code our own OLS finding the coefficients that minimize a function that we will create which is the sum of squared errors.

```
SSR<-function(Beta){
  Pred<-Beta[1]+Beta[2]*WageD$age+Beta[3]*WageD$schooling
  SR<-(WageD$wage-Pred)^2
  ans<-sum(SR)
  return(ans)
}

BetaInic<- c(1 ,1 ,1)
Parameters<- optim(BetaInic, SSR)
Parameters$par
```

[1] If you are interested in the derivation you can check various textbooks. The Wikipedia entry includes some derivations

```
## [1] 0.364708473 0.007459398 0.116422858
```

The last step might seem as an overkill. However, in some machine learning algorithms we will not be able to derive analytical solutions for our coefficients of interest. The approach of writing a coast function and finding the parameters that minimize it is often best suited.

Logistic Regression II

Note that we can write these two goals in one function. We want the following function to be as large as possible:

$$g(\beta) = \left(\frac{1}{1+e^{-x_i\beta}}\right)^{y_i} \times \left(1 - \frac{1}{1+e^{-x_i\beta}}\right)^{1-y_i} \tag{12}$$

This is a special case of the binomial probability mass function. If our email is a spam $y_i = 1$ our function $g()$ becomes:

$$g(\beta) = \left(\frac{1}{1+e^{-x_i\beta}}\right)^{1} \times \left(1 - \frac{1}{1+e^{-x_i\beta}}\right)^{1-1} = \tag{13}$$

$$\left(\frac{1}{1+e^{-x_i\beta}}\right)^{1} \times 1 = \tag{14}$$

$$\left(\frac{1}{1+e^{-x_i\beta}}\right) \tag{15}$$

If our email is non-spam, our $g()$ function becomes:

$$g(\beta) = \left(\frac{1}{1+e^{-x_i\beta}}\right)^{0} \times \left(1 - \frac{1}{1+e^{-x_i\beta}}\right)^{1-0} = \tag{16}$$

$$1 \times \left(1 - \frac{1}{1+e^{-x_i\beta}}\right)^{1-0} = \tag{17}$$

$$\left(1 - \frac{1}{1+e^{-x_i\beta}}\right) \tag{18}$$

In any case, we want our function $g()$ to be as large as possible. We will make a log-transformation. This log-transformation will be useful in the future for many properties that we will see in the future.

$$\ln(g(\beta)) = y_i \ln\left(\frac{1}{1+e^{-x_i\beta}}\right) + (1-y_i) \times \ln\left(1 - \frac{1}{1+e^{-x_i\beta}}\right) \tag{19}$$

Finally, we can write our cost the negative of all the individual $g()$ functions in the sample:

$$J(\beta) = -\sum_{i=1}^{n} y_i \ln\left(\frac{1}{1+e^{-x_i\beta}}\right) + (1-y_i) \times \ln\left(1 - \frac{1}{1+e^{-x_i\beta}}\right) \tag{20}$$

This is the cross-entropy cost, which is the negative of the log-likelihood function. The likelihood function is the probability mass function (probability density function if continuous random variable) where the parameters of the model are used as the arguments of the function.

We will now code our own logistic regression and obtain our estimates of the coefficients based on minimizing the cross-entropy function (maximizing the log-likelihood)

We first start with defining our own sigmoid function:

```
sigmoid<-function(z){
  return(1/(1+exp(-z)))
}
```

We will simply run a model with two predicting variables: frequency of the word 'make' and frequency of the word 'address'

We need to form our X vector taking into account that the first element should be all be vectors of ones as we have a constant β_0

```
x<-cbind(rep(1,4601),email$word_freq_make,email$word_freq_address)

y<-email$spam
```

Now we can define our cross-entropy function:

```
CrossEntropy <- function(beta){
  linearTransformation<-x%*%beta
  input <- sigmoid(linearTransformation)
  Cost <- -sum((y*log(input)) + ((1-y)*log(1-input)))
  n<-nrow(x)
  return(Cost/n)
}
```

We can test it with a given vetor:

```
beta<-c(0,0,0)
CrossEntropy(beta)
```

```
## [1] 0.6931472
```

And now we obtain our own estimates of the logistic regression by minimizing the CrossEntropy function:

```
betaOwn1 <- optim(par=beta,fn=CrossEntropy)
```

We can compare it with the estimates we would obtain with the glm function:

```
spammy <- glm(spam ~ word_freq_make +word_freq_address, data=email, family="binomial")
b <- coef(spammy)
b
```

```
##       (Intercept)    word_freq_make word_freq_address
##       -0.51653118        0.91963804       -0.05000053
```

```
betaOwn1$par
```

```
## [1] -0.51695107  0.91903511 -0.05012397
```

We can also define our cross-entropy function with a for loop:

```
CrossEntropy2<-function(beta){
  cost<-0
  for(i in 1:nrow(x)){
```

```
    linearTransformation<-x[i,]%*%beta
    input<-sigmoid(linearTransformation)
    costI<-y[i]*log(input) + (1-y[i])*log(1-input)
    cost<-cost+costI
  }
  return(-cost)
}
```

And we can attempt to minimize our function CrossEntropy with loops:

```
if (!require("tictoc")) install.packages("tictoc")
```

```
## Loading required package: tictoc
```

```
library(tictoc)
tic("Start Cross Entropy with loops")
betaOwnLoop <- optim(par=beta,fn=CrossEntropy2)
print("done with looped-version")
```

```
## [1] "done with looped-version"
```

```
toc()
```

```
## Start Cross Entropy with loops: 2.091 sec elapsed
```

```
tic("Start cross entropy vectorized")
betaOwn1 <- optim(par=beta,fn=CrossEntropy)
toc()
```

```
## Start cross entropy vectorized: 0.022 sec elapsed
```

```
betaOwn1
```

```
## $par
## [1] -0.51695107  0.91903511 -0.05012397
##
## $value
## [1] 0.6619381
##
## $counts
## function gradient
##       88       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
betaOwnLoop
```

```
## $par
## [1] -0.51695107  0.91903511 -0.05012397
##
## $value
## [1] 3045.577
##
## $counts
## function gradient
```

```
##       88       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

You can see that the vectorized function is much faster. In general, in high-level programing languages (R, Python, Matlab, Julia) you prefer to vectorize than to run for loops even though sometimes it might be easier to code using for loops.

5. Bayesian classification

Let's consider the problem of a spam classifier. We will use a Bayesian classifier to construct a spam classifier using real emails from the "Spambase Data set" available in the Machine Learning Repository from the Center for Machine Learning and Intelligent Systems at the University of California Irvine.

We downlowad the dataset following this command:

```
SpamData<-read.csv("SpamData2.csv")
```

The dataset contains 59 columns. The first 58 columns indicate the occurrence of a word or a character. Let's check some of this data.

```
head(SpamData)
```

```
##   make address all X3d our over remove internet order mail receive will
## 1    0       1   1   0   1    0      0        0     0    0       0    1
## 2    1       1   1   0   1    1      1        1     0    1       1    1
## 3    1       0   1   0   1    1      1        1     1    1       1    1
## 4    0       0   0   0   1    0      1        1     1    1       1    1
## 5    0       0   0   0   1    0      1        1     1    1       1    1
## 6    0       0   0   0   1    0      0        1     0    0       0    0
##   people report addresses free business email you. credit your font X000
## 1      0      0         0    1        0     1    1      0    1    0    0
## 2      1      1         1    1        1     1    1      0    1    0    1
## 3      1      0         1    1        1     1    1      1    1    0    1
## 4      1      0         0    1        0     0    1      0    1    0    0
## 5      1      0         0    1        0     0    1      0    1    0    0
## 6      0      0         0    0        0     0    0      0    0    0    0
##   money hp hpl george X650 lab labs telnet X857 data X415 X85 technology
## 1     0  0   0      0    0   0    0      0    0    0    0   0          0
## 2     1  0   0      0    0   0    0      0    0    0    0   0          0
## 3     1  0   0      0    0   0    0      0    0    0    0   0          0
## 4     0  0   0      0    0   0    0      0    0    0    0   0          0
## 5     0  0   0      0    0   0    0      0    0    0    0   0          0
## 6     0  0   0      0    0   0    0      0    0    0    0   0          0
##   X1999 parts pm direct cs meeting original project re edu table
## 1     0     0  0      0  0       0        0       0  0   0     0
## 2     1     0  0      0  0       0        0       0  0   0     0
## 3     0     0  0      1  0       0        1       0  1   1     0
## 4     0     0  0      0  0       0        0       0  0   0     0
## 5     0     0  0      0  0       0        0       0  0   0     0
## 6     0     0  0      0  0       0        0       0  0   0     0
##   conference semicolon parenthesis squared_brackets exclamation_point
```

```
## 1          0          0            0                0                 1
## 2          0          0            1                0                 1
## 3          0          1            1                0                 1
## 4          0          0            1                0                 1
## 5          0          0            1                0                 1
## 6          0          0            1                0                 0
##   usd_symbol number_sign Spam
## 1          0           0    1
## 2          1           1    1
## 3          1           1    1
## 4          0           0    1
## 5          0           0    1
## 6          0           0    1
```

```r
mean(SpamData$Spam)
```

```
## [1] 0.3940448
```

39% of emails in the dataset are spam.

```r
SpamEmails<-subset(SpamData,Spam==1)
mean(SpamEmails$exclamation_point==1)
```

```
## [1] 0.8334253
```

83.3% of spam emails contain at least one exclamation point.

Now, let's consider the problem of predicitng wether or not an email is spam based on the observed characteristics. Let y_i be a discrete variable taking the value of zero if email i is spam, and zero otherwise.

$$y_i = \begin{cases} 0 \text{ if email } i \text{ is no spam} \\ 1 \text{if email } i \text{ is spam} \end{cases} \tag{21}$$

For each email i we observe the occurrence of various words and characters. We call these features $x_i = [x_i^1, x_i^2, ...x_i^p]$ where

$$x_i^p = \begin{cases} 0 \text{ if email } i \text{ does not containt feature } p \\ 1 \text{if email } i \text{ contains feature } p \end{cases} \tag{22}$$

We are interested in predicting the probability of an email being spam or not based on the occurrence of several features: $ f(y_i|x_i)$

Recall that:

$$\underbrace{f(y_i|x_i)}_{posterior} = \frac{\underbrace{f(x_i|y_i)}_{likelihood}\underbrace{f(y_i)}_{prior}}{\underbrace{f(x_i)}_{evidence}} \tag{23}$$

The "Naive Bayes classifier" assumes conditional indpendence between features given y. Note that if x_1 and x_2 are conditionally independent given y:

$$f(x_i^1, x_i^2|y) = f(x_i^1|x_i^2, y) \times f(x_i^2|y) = f(x_i^1|y) \times f(x_i^2|y) \tag{24}$$

Repeating a similar procedure for x, we can see that the likelihood can be expressed as:

$$f(x_1, x_2, ...x_p|y) = f(x_1|y) \times f(x_2|y) \times .... \times f(x_p|y) = \prod_{j=1}^{p} f(x_i|p) \tag{25}$$

With this in mind, the posterior can be expressed as:

$$f(y_i|x_i) = \frac{f(y_i) \times \prod_{j=1}^{p} f(x_i|p)}{f(x_i)} \propto f(y_i) \times \prod_{j=1}^{p} f(x_i|y) \tag{26}$$

The $\propto$ symbol denotes proportionallity and is used commonly in Bayesian statistics as the term $f(x_i|p)$ is not something we need to worry when estimating our parameters of interest. The goal of the Bayesian classifier is to obtain a classifier by maximizing the posterior probability. That is:

$$\hat{y}_i = h(x_i) = \arg\max_{y} f(y_i) \times \prod_{j=1}^{p} f(x_i^j|y) \tag{27}$$

Note that if the conditionally independence assumption holds, the estimated function $h(x_i)$ corresponds to the posterior distribution. We need to estimate two elements: the likelihood function $\prod_{j=1}^{p} f(x_i^j|y)$ and the prior $f(y_i)$. In our example, we observe that 39% of emails are spam, then we estimate that the probability of an e-mail being spam is 0.39 and the probability of an email not being spam is 0.61. We let y follow a Bernoulli distribution: $f(y) = 0.39^y \times (1 - 0.39)^{(1-y)}$

Now we need estimates of our likelihood function. In the wages example we showed how to estiamte $f(x_i^j|p)$ through maximum likelihood if we assume a normal distribution.

$$f(x_i^j|y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\left(x_i^j - \mu_{j,y}\right)^2}{\sigma_y^2}} \tag{28}$$

In this case, however, the normal assumption does not fit. We have zero, one variables depending on the occurence. We are going to assume a Bernoulli distribution. Before going into the specifics of this pdf, let's define:

$$\theta_{j,y}: \text{ the probability that in an email of type } y\text{, the feature } j \text{ is observed.} \tag{29}$$

For instance, if $\theta_{1,0} = 0.4$ it means that in non-spam emails ($y = 0$) the probability of observing the feature 1, which corresponds to the word "make" is 0.4. Then, the likelihood of a given feature, for one observation, can be expressed as:

$$f(x_i^j|y) = \theta_{j,y}^{x_i^j} \times (1 - \theta_{j,y})^{(1-x_i^j)} \tag{30}$$

Under the assumption of independence across observations, we know that:

$$f(x^j|y) = f(x_1^j, x_2^j, ...x_n^j|y) = \prod_{i=1}^{n} f(x_i^j|y) = \theta_{j,y}^{x_i^j} \times (1 - \theta_{j,y})^{(1-x_i^j)} \tag{31}$$

The maximum likelihood estimator of $\theta_{j,y}^{x_i^j}$ corresponds to the proportion of emails in the category y such that the feature j is observed. For instance, 83% of spam emails contain the exclamation point, which corresponds to feature "53". This means that $\theta_{53,1} = 0.83$.

Challenge: prove that the maximum likelihood estiamte of $\theta_{j,y}$ corresponds to the proportion of emails in category y that contain the feature j.

Let's now run our Bayesian classifier. We will a package already implementing a Bayesian classifier in R that will repeat what we just did for all the features.

```
#install.packages("e1071")
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.2
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     sigmoid
```

We will also split our sample into training and testing data:

```
#install.packages("caTools")
library(caTools)
SpamData$sample<-sample.split(SpamData$Spam,SplitRatio=0.8)
train=subset(SpamData, SpamData$sample==TRUE)
test=subset(SpamData, SpamData$sample==FALSE)

train<-train[,1:55]
test<-test[,1:55]
```

We call our naiveBayes function with to estimate the model and we evaluate its performance on the testing dataset

```
nB_model <- naiveBayes(as.factor(Spam) ~.,data=train)
T<-table(predict(nB_model, test[,1:54]),as.factor(test[,55]))
colnames(T)<-c("predicted non-spam","predicted spam")
rownames(T)<-c("true non-spam", "true spam")
T<-T/sum(T)
T
```

```
##
##                 predicted non-spam predicted spam
##   true non-spam         0.48968512     0.02605863
##   true spam             0.11617807     0.36807818
```

We predict accurately 82% of emails. However, 2% of predicted spam emails are non-spam (type I error) whereas 15% of spam emails are not identified correctly (type II error). Some measures commonly used in Machine Learning to evaluate the performance of an algorithm are precission, recall, and accuracy.

$$\text{Precission} = \frac{\text{True positive}}{\text{True Positive+False Positive}} \tag{32}$$

$$\text{Recall} = \frac{\text{True positive}}{\text{True Positive+False Negative}} \tag{33}$$

$$\text{Accuracy} = \frac{\text{True positive+true negative}}{\text{All}} \tag{34}$$

```
#Precission:
T[2,2]/(T[2,2]+T[1,2])
```

```
## [1] 0.9338843
```

```
#Recall
T[2,2]/(T[2,2]+T[2,1])
```

```
## [1] 0.7600897
```

```
##Accuracy
T[2,2]+T[1,1]
```

```
## [1] 0.8577633
```

Challenge: The "Arrhythmia" dataset available in this link: https://archive.ics.uci.edu/ml/datasets/Arrhythmia contains information about an individuals and their ECG results, together with labels to identify if there is arrhytmia or not. Construct a Bayesian classifier to identify heart arrhythmia based on these conditions. There are 279 different attributes.

6. Maximum Likelihood Estimation

Before going into the specifics of maximum likelihood estimation, let's review some basic concepts from statistics:

Let y and x be continuous random variables with a joint probability. Let $f(y, x)$ denote the joint probability density function. Then:

$$1.f(y|x) = \frac{f(y,x)}{f(x)} \text{ Conditional pdf} \tag{35}$$

$$2.f(y) = \int_{-\infty}^{\infty} f(y,x)dx \text{ marginal pdf} \tag{36}$$

$$3.f(y) \geq 0 \forall y \tag{37}$$

$$4. \int_{-\infty}^{\infty} f(y)dy = 1 \tag{38}$$

$$5. \int_{-\infty}^{\bar{y}} f(y)dy = F(\bar{y}) = P(y \leq \bar{y}) \text{ cumulative distribution function} \tag{39}$$

$$6.\text{Do not interpred } f(y) \text{ as a probability!} \tag{40}$$

$$7.y \text{ and } x \text{ are independent iif } f(y,x) = f(y) \times f(x) \tag{41}$$

Bayes rule

$$f(y|x) = \frac{f(y,x)}{f(x)} \tag{42}$$

$$\underbrace{f(y|x)}_{posterior} = \frac{\underbrace{f(x|y)}_{likelihood}\underbrace{f(y)}_{prior}}{\underbrace{f(x)}_{evidence}} \tag{43}$$

Suppose the data $\{Data\}_{i=1}^{n} = \{Wage, age, schooling\}_{i=1}^{n}$ has a density $f(Data;\Theta)$. The density $f(.)$ is known up to a parameter Θ. The joint density is given by $\bar{f}$:

$$\bar{f}(Data_1, Data_2, ..., Data_n; \Theta) \tag{44}$$

If the data is i.i.d, the joint density can be written as the product of independent pdf's:

$$\bar{f}(Data_1, Data_2, ..., Data_n; \Theta) = f(Data_1;\Theta) \times f(Data_2;\Theta), ..., f(Data_n;\Theta) \tag{45}$$

The likelihood function is the joint pdf but it is considered a function of the parameter Θ conditional on the data:

$$\mathcal{L}(\Theta, Data) = \prod_{i=1}^{n} f(Data_i;\Theta) \tag{46}$$

Caution: you cannot interpret the likelihood as the probability of observing the dataset conditional on a parameter. In continuous random variables the pdf is not the probability.

Let's assume that log-wages are determined according to the following model:

$$\text{wage}_i = \beta_0 + \beta_1 \text{age}_i + \beta_2 \text{yrschool}_i + \varepsilon_i \tag{47}$$

And consider the case where

$$\varepsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{48}$$

Recall, the pdf of random variable x following a normal distribution with mean μ and variance σ^2 is:

$$f(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{\sigma^2}(x-\mu)^2} \tag{49}$$

From Equation 47 we see that the pdf of wages follows the distribution described in Equation 50:

$$f(\text{wage}_i; \text{age,yrschool,}\sigma^2) = \mathcal{N}\left(\beta_0 + \beta_1 \text{age}_i + \beta_2 \text{yrschool}_i, \sigma^2\right) \tag{50}$$

Then the joint likelihood becomes:

$$\mathcal{L}(\Theta; Data) = \prod_{i=1}^{n} (f(\text{wage})_i; \beta, \sigma^2) \tag{51}$$

Likelihood functions are usually very close to zero. For computational precission, we usually work with the log-likelihood function.

$$l(\Theta; Data) = \ln\left(\mathcal{L}(\Theta; Data)\right)$$

$$= \sum_{i=1}^{n} ln(f(\text{wage})_i; \beta, \sigma^2) \tag{52}$$

where all the parameters $\Theta = \left[\beta_0, \beta_1, \beta_2, \sigma^2\right]$

The maximum likelihood estimator: $\Theta_{\text{MLE}} = \left[\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \hat{\sigma}^2\right]$ is:

$$\Theta_{\text{MLE}} = \arg max_{\Theta} l(\Theta; Data) \tag{53}$$

MLE in R

In this section we will compute the Maximum Likelihood estimators of the model described in Equations 47 - 53. First, we define the likelihood function of each individual observation. This is an intuitive, but computationally slow, way of defining the likelihood function

```
Likelihood<-function(Theta){
  bbeta0<-Theta[1]
  bbeta1<-Theta[2]
  bbeta2<-Theta[3]
  ssigma<-exp(Theta[4])

  loglik=0;
  for(i in 1:n){
    predwage<-bbeta0+bbeta1*WageD$age[i]+bbeta2*WageD$schooling[i]
    yobserved<-WageD$wage[i]
    error<-yobserved-predwage
    loglikelihood=log(dnorm(error,mean=0,sd=(ssigma)))
    loglik=loglik+loglikelihood
  }
  loglik=-loglik
  return(loglik)

}
```

Once defined the (negative) of the likelihood function, we find the parameters that maximize (minimize the negative):

```
Theta<-c(1,1,1,1)
Likelihood(Theta)
```

```
## [1] 124240
```

```
Parameters<-optim(Theta,Likelihood)
Parameters
```

```
## $par
## [1]  0.343513669  0.007879194  0.116957103 -0.481395875
##
## $value
## [1] 750.7314
##
## $counts
## function gradient
##      351       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Challenge homework 1: Prove analytically that the maximum likelihood estimator is the same as the OLS estimator.

Parallelism dominates several applications of machine learning and is, in many cases, the common way to estimate machine learning models that are computationally burdensome. However, what is parallelism? [This, in principle, would go in the ppt, not necessarily in the markdown]

Challenge: Estimate the likelihood function in section 3 via GPU-parallelization. The code is available in this link

Resources:

https://www.infoworld.com/article/3299703/deep-learning/what-is-cuda-parallel-programming-for-gpus.html

https://developer.nvidia.com/how-to-cuda-python

https://developer.nvidia.com/machine-learning

7. Gradient descent

As you have seen, most problems in machine learning involve minimizing a cost function. So far we have implemented one function in R called 'optim' to perform the minimization of a function. This package implements various algorithms to minimize a function. We will se one algorithm commonly used in Machine learning called 'Gradient descent'.

5.1. Basics of gradient descent.

Let's assume we want to minimize the function $f(x) = x^2 + 2x + 2$. Although we can analytically find the value x that minimizes this function, we will often encounter problems without analytic solution. In this case, how do we proceed?

We can start with a guess, say $x_0 = 3$. However, how do we proceed afterwards? What would be our next guess x_1? What gradient descent does is, it evaluates the derivative of the function at that point and updates the new value accordingly.

Since we know that $\frac{\partial f(x)}{\partial x} = 2x + 2$ we know $f'(3) = 8$. We update our guess of x_1 with the following rule:

$$x_1 = x_0 - \alpha \times f'(x_0) \tag{54}$$

where α is called the 'learning rate'. The derivative is a measure of infinitesimale change in the function. This implies that if the function is very steep at a point, the update will be far from the previous guess and the other way around. Gradient descent performs this update iteratively until a stopping criterion is reached. This stoping criterion can be something such as $|x_i - x_{i-1}| < \varepsilon$ for ε close to zero or when a maximum number of iterations is reached.

Formally, this is the gradient descent algorithm:

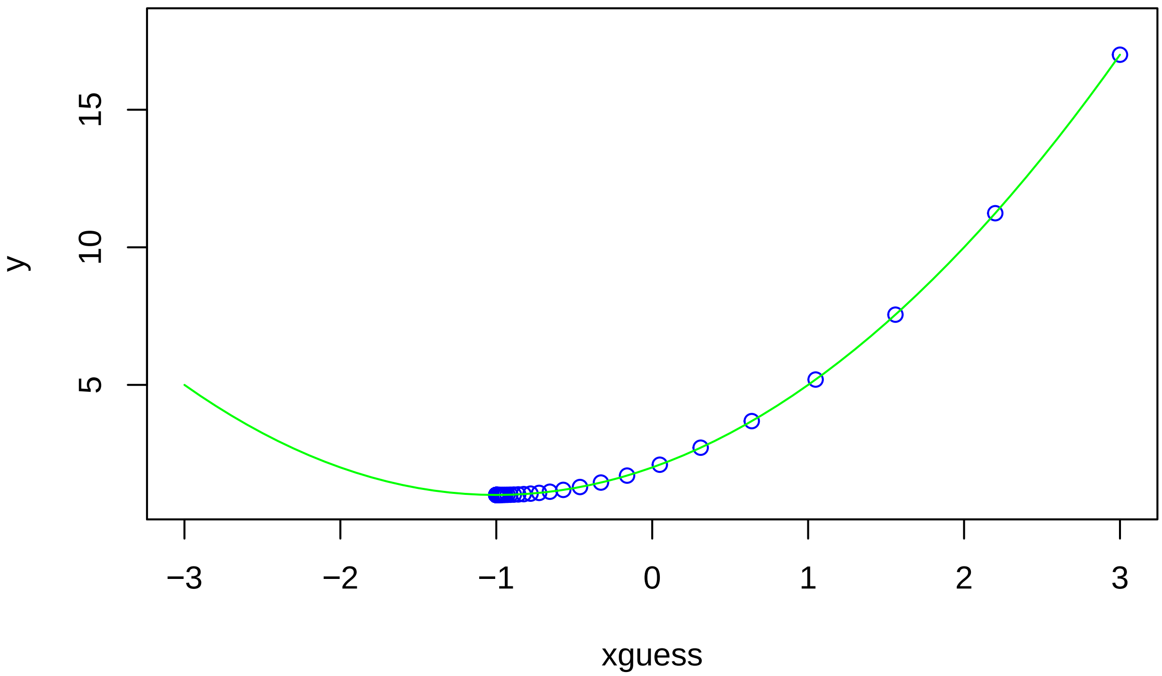
1. Take initial guess x_0
2. For i=1,…L 2.1 $x_i = x_{i-1} - \alpha f'(x_{i-1})$

In the following code you are going to try gradient descent. You can modify the learning rate to see what happens with the optimizer.

```r
x<- seq(-3,3,0.1)
L<-length(x)
xguess <- rep(NA, L)
y <- rep(NA, L)
fx=x^2+2*x+2
fxe<-function(x){x^2+2*x+2}
df<-function(x){2*x+2}
x0=3
alpha=0.1


for (i in 1:L) {
  xguess[i]=x0
  y[i]=fxe(x0)
  x0=x0-alpha*df(x0)
}


plot(xguess,y, ylim = c(0.8, 18),xlim=c(-3,3),col="blue")
lines(x,fx,col="green")
```



Or you can check the animation:

```
if (!require("animation")) install.packages("animation")
```

```
## Loading required package: animation
```

```
library(animation)
saveHTML({
  x<- seq(-3,3,0.1)
  xguess=x*10000
  y<-10000*x
  fx=x^2+2*x+2
  df<-function(x){2*x+2}
  x0=3
  fxe<-function(x){x^2+2*x+2}
  fxeVector<-fx*0
  y[1]=x0
  for (i in 1:20) {
    plot(xguess,y, ylim = c(0.8, 18),xlim=c(-3,3),col="blue")
    lines(x,fx,col="green")
    x0=x0-0.1*df(x0)
    xguess[i+1]=x0
    y[i+1]=fxe(x0)
    ani.pause()
  }
}, img.name = "Grad_Descent", imgdir = "Grad_descent", htmlfile = "Grad_descent.html",
autobrowse = FALSE, title = "Demo of 20 grad descent")
```

```
## HTML file created at: Grad_descent.html
```

Chosing the right learning rate can be a challenge. A small learning rate can slow significantly your code whereas a large one can make the algorithm to diverge. Chosing the initial guess can also be a challenge. Some functions have a set of local minima and the election of your initial guess might determine where you end up.