CS5100: Assignment 1

1. Task 1.

Implement the logistic regression algorithm using gradient descent in R. And, the objective function, which the program should minimize, is the negative log-likelihood.

- Code both the logistic regression model and the training algorithm from first principles.
- Fix the number of attributes, d, so that you can use your algorithm on the following tasks.

1.1. Introduction

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

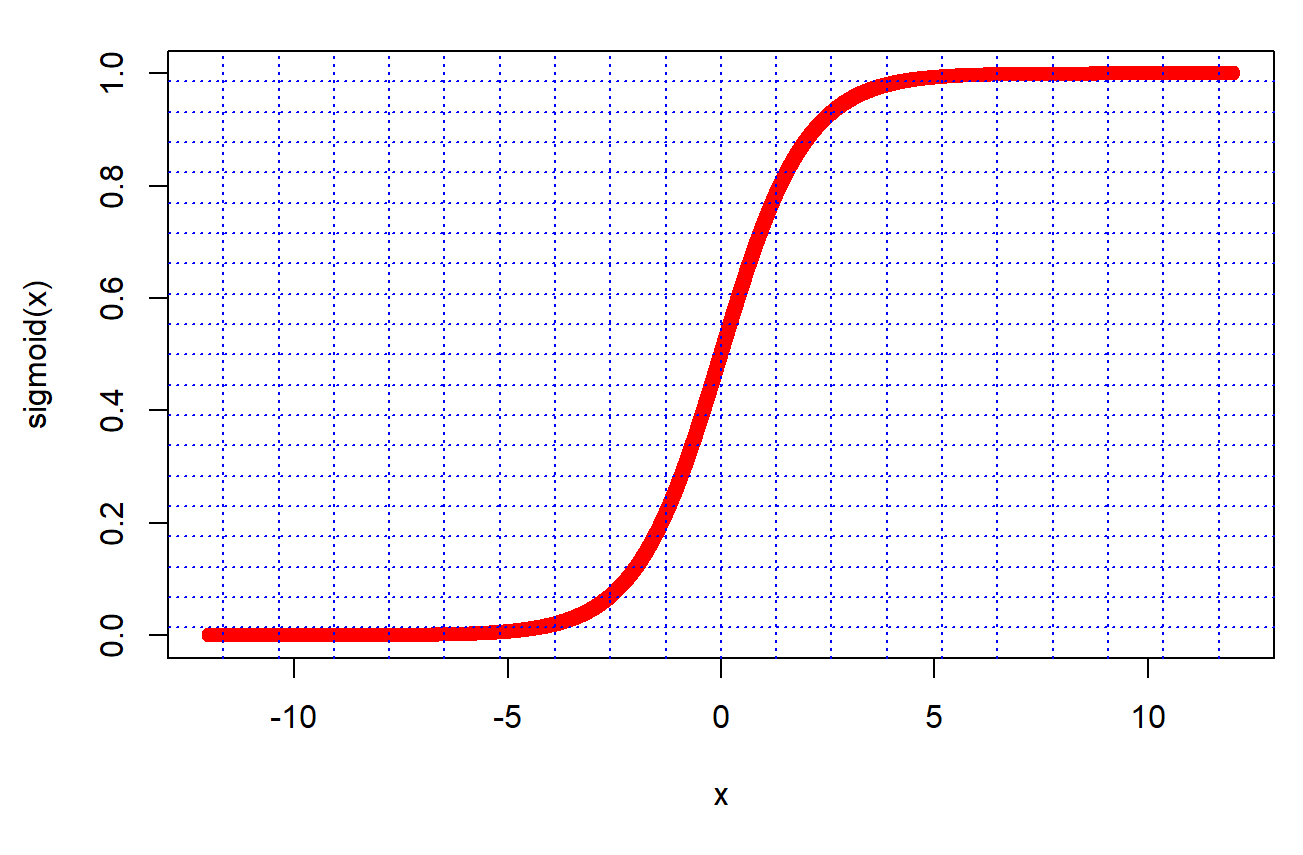
So, in this assignment the goal of gradient descent is to find the values of parameters (coefficients) of a function that minimizes the objective function, which is negative log-likelihood.

1.2. By using sigmoid function find the predicted probability of logistic regression.

$$sigmoid = \frac{1}{1+e^{-x}}$$

```
# Sigmoid function takes the value of linear model
sigmoid <- function(t){
  return (1/(1 + exp(-t)))
}
```

```
x <- seq(-12, 12, 0.0001)
plot(x, sigmoid (x), col="red")
grid(nx = 20, ny = 20, col = "Blue", lty = "dotted",
     lwd = par("lwd"), equilogs = TRUE)
```



1.3. If the sigmoid is $>= 0.5$ the prediction is 1 otherwise 0.

```
# Returns the prediction values
Prediction <- function(X, w1){
  model <- w1 %*% t(X)
  p <- sigmoid(model)
  f.hat <- c()
  for (j in 1:length(p[1,])){
    if (p[1,j] >= 0.5)
      f.hat[j] <- 1
    else
      f.hat[j] <- 0
  }
  return (f.hat)
}
```

1.4. Gradient descent returns the final parameters (coefficients) that minimizes negative log-likelihood function.

```
# The gradient descent returns the final intercept and coefficients values.
# X is Attributes.
# Y is Labels.
# lr is Learning rate.
# s is Stopping rule.
# n is number of steps.
# w0 is Initial weight.
# w1 is New weight.
# ER is Error rate.
GD <- function(X, Y, lr, s, n=NULL){

  w0 <- runif(ncol(X),-0.7,0.7)
  w0 <- as.matrix(runif(ncol(X), -0.7,0.7))

  w1 <- w0 - lr * t(X) %*% (sigmoid(X %*% w0) - Y)
  if (is.null(n)) {
    while(sum(w1==w0) != s)
    {
      w0 <- w1
      w1 <- w0 - lr * t(X) %*% (sigmoid(X %*% w0) - Y)
    }
  }
  y.pred <- Prediction(X[,2:5],t(w1)[,2:5])
  er <- sum((Y - y.pred)^2)/(nrow(X))
  return(list(coefficients=w1, ER=er))
}
```

2. Task 2.

Load the Auto data set and create a new variable **high** that takes values:

$$high = (\ 1\ if\ mpg >= 23\ \ otherwise\ 0\ )$$

- Apply your program to the Auto data set and new variable.
- To predict high given horsepower, weight, year, and origin.
- (In other words, high is the label and horsepower, weight, year, and origin are the attributes.)
- Since origin is a qualitative variable, you will have to create appropriate dummy variables.
- Normalize the attributes.

2.1. Loading the Auto data set.

```
# Loading the Auto data set
Auto <- read.table("Auto.data", header=T, na.strings="?")
dim(Auto)
```

```
## [1] 397   9
```

Deleting the rows that contain ?.

```
# Deleting the rows that contain ?
Auto <- na.omit(Auto)
dim(Auto)
```

```
## [1] 392   9
```

```
names(Auto)
```

```
## [1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
## [6] "acceleration" "year"         "origin"       "name"
```

2.2. Creating a new variable high and combine it with the Auto data set.

```
attach(Auto)
# creating a new variable high
high <- ifelse(mpg >= 23,1,0)
Auto <- data.frame(Auto,high)
names(Auto)
```

```
##  [1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
##  [6] "acceleration" "year"         "origin"       "name"         "high"
```

2.3. Selecting only 5 attributes.

```
# Selecting only 5 attributes.
Auto = subset(Auto, select = c(horsepower, weight, year, origin, high))
names(Auto)
```

```
## [1] "horsepower" "weight"     "year"       "origin"     "high"
```

2.4. Creating appropriate dummy variables and normalizing the selected attributes.

```
# creating appropriate dummy variables
# and Normalizing the selected attributes.
Dummy.Auto = subset(Auto, select = -c(origin,high))
Auto = cbind(scale(Dummy.Auto),subset(Auto,select=c(origin,high)))
# Removing the name of the column
rownames(Auto) <- c()
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 4.0.4
```

```
knitr::kable(Auto[1:12, ], align = "lccrr")
```

horsepower	weight	year	origin	high
0.6632851	0.6197483	-1.623241	1	0
1.5725848	0.8422577	-1.623241	1	0
1.1828849	0.5396921	-1.623241	1	0
1.1828849	0.5361602	-1.623241	1	0
0.9230850	0.5549969	-1.623241	1	0
2.4299245	1.6051468	-1.623241	1	0
3.0014843	1.6204517	-1.623241	1	0
2.8715843	1.5710052	-1.623241	1	0
3.1313843	1.7040399	-1.623241	1	0
2.2220846	1.0270935	-1.623241	1	0
1.7024847	0.6892089	-1.623241	1	0
1.4426848	0.7433646	-1.623241	1	0

3. Task 3.

Split the data set randomly into two equal parts, which will serve as the training set and the test set.

- Use your birthday (in the format MMDD) as the seed for the pseudo-random number generator.
- The same training and test sets should be used throughout this assignment.

3.1. Splitting the data set randomly into two equal parts.

```
# October, 06
set.seed(1006)
# Splitting the data set randomly into two equal parts
new.data <- sample(1:nrow(Auto), nrow(Auto)/2)
# the training set and the test set that will be used throughout this assignment.
auto.train <- Auto[new.data,]
auto.test <- Auto[-new.data,]
```

3.2. Adding 2 new columns for the intercepts and naming them.

```
# Adding 2 new columns for the Intercepts
auto.train <- cbind(rep(1,196),auto.train)
auto.test <- cbind(rep(1,196),auto.test)

# Naming the columns
names(auto.train)[1] <- "Intercept"
names(auto.test)[1] <- "Intercept"

# Removing the names
rownames(auto.train) <- c()
rownames(auto.test) <- c()
```

3.3. Separating the attributes and labels of the training set and the test set.

```
train.X <- as.matrix(auto.train[-ncol(auto.train)])
test.X <- as.matrix(auto.test[-ncol(auto.test)])
train.Y <- as.matrix(auto.train$high)
test.Y <- as.matrix(auto.test$high)
```

4. Task 4.

Train your algorithm on the training set.

- Using independent random numbers in the range [−0.7, 0.7] as the initial weights.
- Find the Error Rate of the trained model on the test set,

i.e. compute

$$ER(\hat{f}, D_{test}) = |D_{test}|^{-1}$$
$$\sum_{(x,y)\in D_{test}} (y - \hat{y})^2$$
$$\hat{y} = arg\ max\{1 - \hat{f}, \hat{f}\}$$

- Try different values of the learning rate,

$$10^{-10} < \eta <= 10^{-1}$$

- and of the number of training steps, T, i.e. let your stopping rule be to stop after a given number of steps.
- Give a small table of test and training ER in your report.

Computing the coefficients and the intercept of the training data set. And, comparing logistic regression model outputs built from first principles with the build-in function in R.

4.1. Training the algorithm on the training set using independent random numbers in the range [−0.7, 0.7] as the initial weights.

Training the model on train data set using glm function.

```
train.model<- glm(high ~ horsepower+ weight + year + origin, data=auto.train, family=binomial)
print(train.model$coefficients)
```

```
## (Intercept)  horsepower      weight        year      origin
##  -2.7088662  -2.1212646  -4.3629623   1.6240897   0.4550773
```

Training the model on train data set using implemented logistic regression using gradient descent.

```
# Model training
Train.GD <- GD(train.X, train.Y, 0.01, 4)
print(t(Train.GD$coefficients))
```

```
##      Intercept horsepower    weight    year    origin
## [1,] -2.708866  -2.121265 -4.362962 1.62409 0.4550773
```

Training the model on test data set using glm function.

```
# To check the outcomes, comparing gradient descent with the build-in function.
test.model<- glm(high ~ horsepower+ weight + year + origin, data=auto.test,family=binomial)
print(test.model$coefficients)
```

```
## (Intercept)  horsepower      weight        year      origin
##   -1.942954   -1.211135   -3.416030    2.163839    1.219584
```

Training the model on test data set using implemented logistic regression using gradient descent.

```
Test.GD <- GD(test.X, test.Y, 0.01, 4)
print(t(Test.GD$coefficients))
```

```
##      Intercept horsepower   weight     year   origin
## [1,] -1.942954  -1.211135 -3.41603 2.163839 1.219584
```

4.2. Finding the Error Rate of the trained model on the test set.

Calculating the error rate on the test set.

```
# ER (Error Rate) of the trained model on the test set.
test.x = test.X[,2:5]
test.coef = (t(Test.GD$coefficients)[,2:5])
test.predicted <- Prediction(test.x, test.coef)
test.er <- mean(test.predicted != test.Y)
cat("Test ER : ", test.er)
```

```
## Test ER :  0.1122449
```

```
test.er.model = Test.GD$ER
print(test.er.model)
```

```
## [1] 0.1122449
```

4.3. Trying different values of learning rate and number of training steps. And, letting the stopping rule to stop be fixed.

```
Test.ER<-c()
Train.ER<-c()
Learning.Rates<-c()
for (lr  in seq(0.000001,0.1,0.001)){
  for(i in c(1:2)){
    test.gd <- GD(test.X, test.Y, lr, 10, i)
    train.gd <- GD(train.X, train.Y, lr, 10, i)
    Test.ER <- c(Test.ER, test.gd$ER)
    Train.ER <- c(Train.ER, train.gd$ER)
    Learning.Rates <- c(Learning.Rates, lr)
  }
}
```

4.4. Table of test and training error rate (ER).

```
# Table of test and training ER
table <- cbind(Learning.Rates, Train.ER, Test.ER)
knitr::kable(table[1:12, ], align = "lccrr")
```

Learning.Rates	Train.ER	Test.ER
0.000001	0.6020408	0.3520408
0.000001	0.6428571	0.5357143
0.001001	0.5051020	0.7959184
0.001001	0.4387755	0.4591837
0.002001	0.4336735	0.4132653
0.002001	0.7704082	0.4387755
0.003001	0.1275510	0.2448980
0.003001	0.5153061	0.4591837
0.004001	0.3163265	0.1479592
0.004001	0.2551020	0.3010204
0.005001	0.4132653	0.2091837
0.005001	0.2551020	0.2295918

```
library(ggplot2)
```

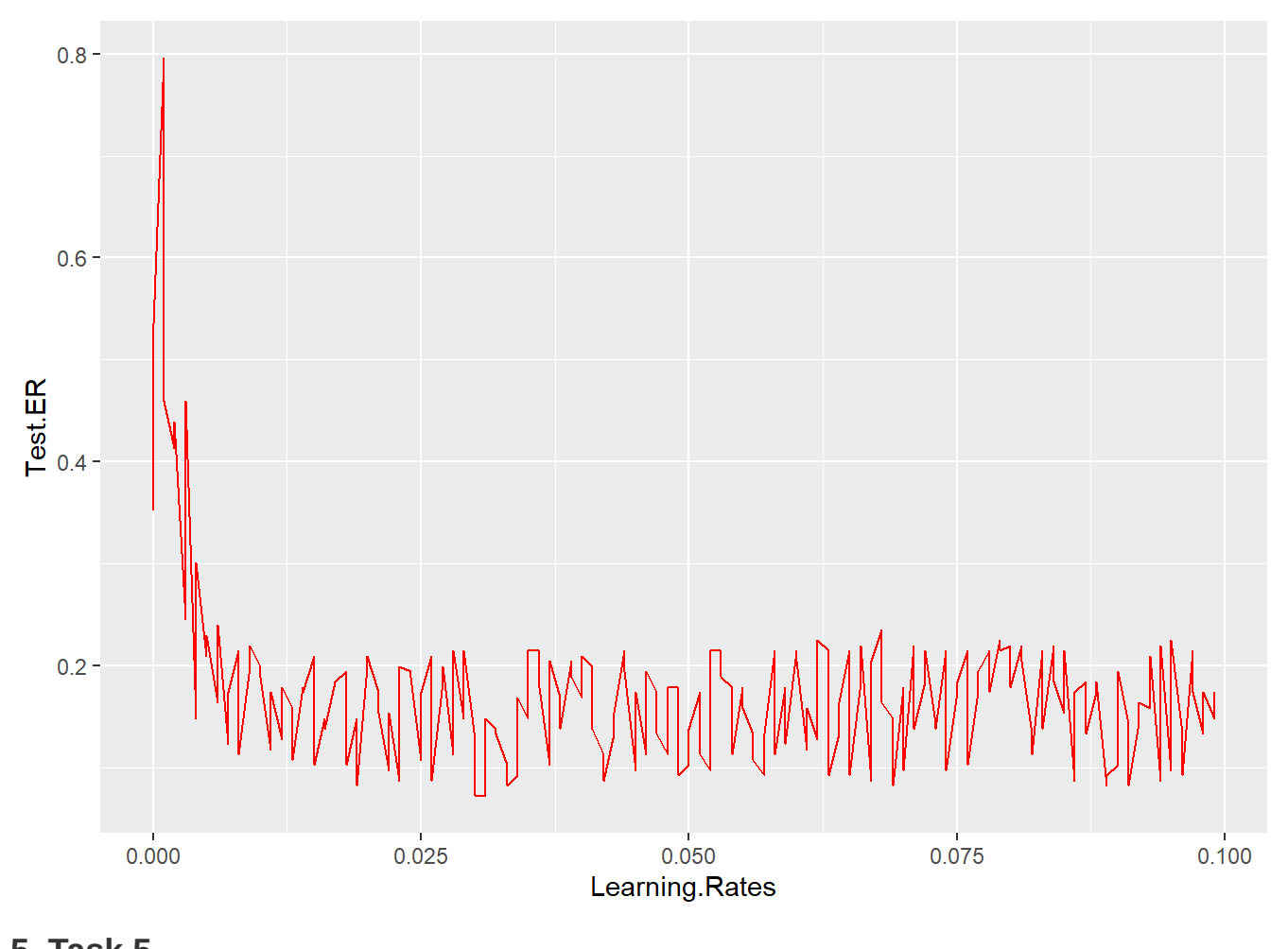
```
## Warning: package 'ggplot2' was built under R version 4.0.4
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'Auto':
##
##     mpg
```

```
table.data <- as.data.frame.matrix(table)
ggplot(data = table.data, aes(x = Learning.Rates, y =Train.ER)) + geom_line(color = "blue")
```


```
ggplot(data = table.data, aes(x = Learning.Rates, y =Test.ER)) + geom_line(color = "red")
```



5. Task 5.

Try different stopping rules, such as:

- stop when the value of the objective function does not change.
- by more than 1% of its initial value over the last 10 training steps.

```
j = 0
Total.ER <- c()
while (TRUE) {
  value <- c()
  j = j + 1
  ER.1 <- GD(train.X, train.Y, 0.01, 3, j)
  Total.ER <- c(Total.ER, ER.1$ER)
  ER.0 = Total.ER[1]*100

  num = 0
  if(j > 10){
    value <- c()
    for(k in c(j-10:j)){
      value = c(value,(ER.0 - (Total.ER[k]*100)))
    }
    for (n in tail(value, 10)) {
      if(abs(n) <= 1.00){
        num = num + 1
      }
    }
  if (num >= 10){
    break
  }
  }
  if(j > 100){
    break
  }
}
```

```
head(Total.ER, 12)
```

```
##  [1] 0.2704082 0.1530612 0.1683673 0.1734694 0.2602041 0.1632653 0.2346939
##  [8] 0.2193878 0.1989796 0.2448980 0.1836735 0.1377551
```

```
print(num)
```

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

```
print(ER.0)
```

```
## [1] 27.04082
```

6. Task 6.

Run your logistic regression program for:

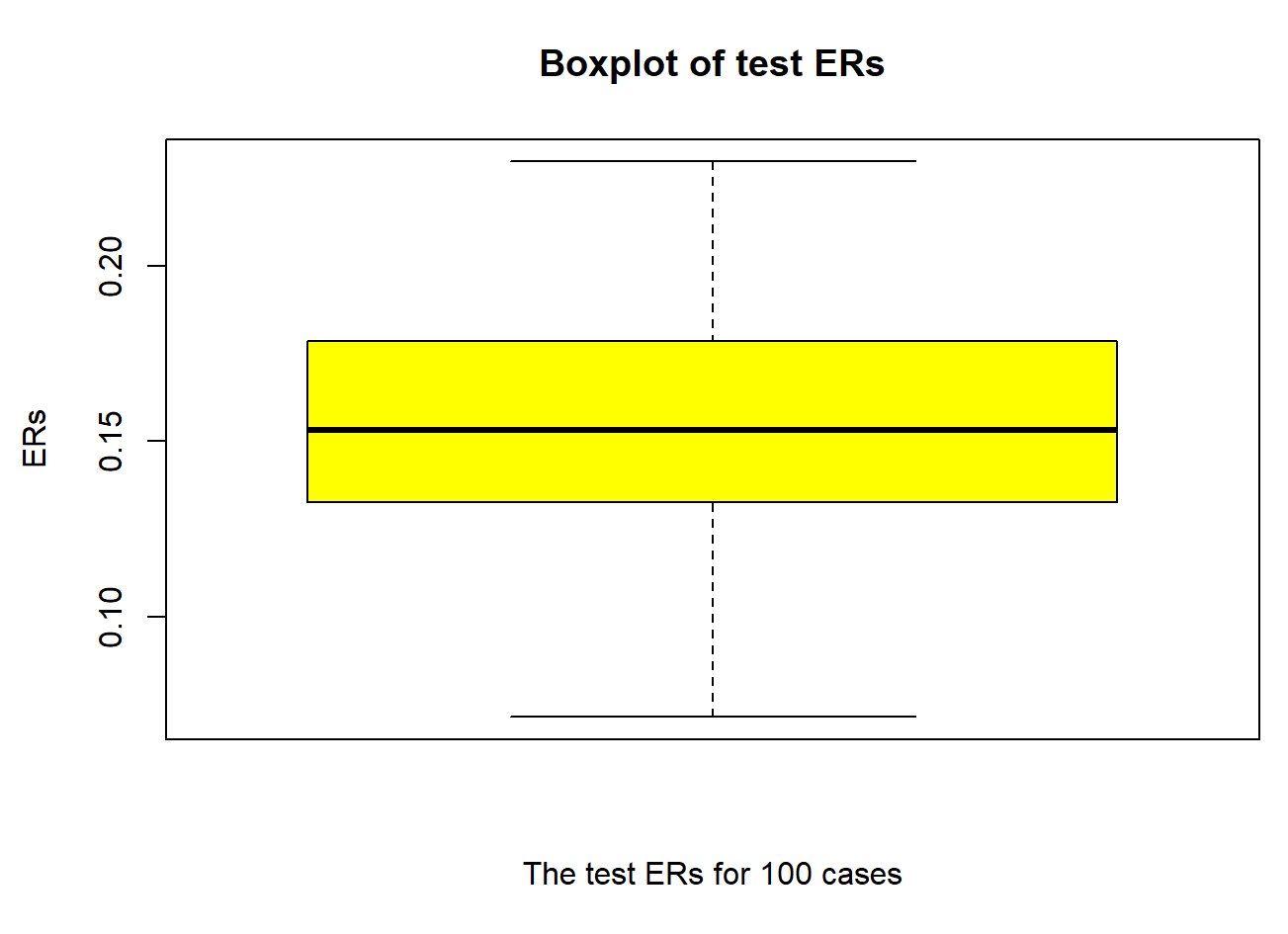
- a fixed value of η and for a fixed stopping rule 100 times
- (producing reasonable results in your experiments so far),
- for different values of the initial weights (produced as above, as independent random numbers in [−0.7, 0.7]).
- In each of the 100 cases compute the test ER
- and show it in your report as a box-plot.

6.1. Running logistic regression for fixed values of learning rates and stopping rules 100 times.

```
test.ER.fixed <- c()
for (l in c(1:100)){
  test.fixed = GD(test.X, test.Y, 0.01, 10, l)
  train.fixed = GD(train.X, train.Y, 0.01, 10, l)
  test.ER.fixed <- c(test.ER.fixed, test.fixed$ER)
}
```

6.2 Computing the test error rates for each case and plotting them as a box-plot.

```
# box plot
boxplot(test.ER.fixed, width = 10, varwidth = TRUE,
        col = 'yellow',
        main='Boxplot of test ERs',
        xlab='The test ERs for 100 cases',
        ylab='ERs')
```



7. Task 7.

Redo the experiments in items 4–5:

- modifying the training procedure as follows.
- Train F(X, λ) 4 times using gradient descent with different values of the initial weights and
- then choose the prediction rule with the best training objective.

7.1. Training the model 4 times with different values of the initial weights.

In this, the model will be trained four times by using a fixed learning rate and stopping rule, and different values of training steps. Finally, for each model coefficients and error rates will be computed.

```
ER.Four <- c()
coef1 <- c()
coef2 <- c()
coef3 <- c()
coef4 <- c()
for (j in c(1:4)) {
  train.model = GD(train.X, train.Y, 0.01, 10, j)
  if (j == 1){
    coef1 <- c(coef1, train.model)}
  if (j == 2){
    coef2 <- c(coef2, train.model)}
  if (j == 3){
    coef3 <- c(coef3, train.model)}
  if (j == 4){
    coef4 <- c(coef4, train.model)}
  ER.Four <- c(ER.Four, train.model$ER)
}
```

7.2. Printing four error rates.

```
print(ER.Four)
```

```
## [1] 0.1938776 0.1326531 0.1377551 0.2142857
```

7.3. Here, the minimum error rate will be printed out of four error rates.

```
Best.ER <- min(ER.Four)
print(Best.ER)
```

```
## [1] 0.1326531
```

Getting coefficients of the minimum error rate.

```
axi = match(Best.ER, ER.Four)
put = gsub(" ", "", paste('coef',axi))
print(put)
```

```
## [1] "coef2"
```

```
Test.X = test.X[,2:5]
Best.weight = (t(eval(parse(text = put))$coefficients)[,2:5])
```

Predicting by using coefficients that have lowest error rate.

```
Test.Pred <- Prediction(Test.X, Best.weight)
head(Test.Pred, 40)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0
## [39] 0 0
```

The actual labels.

```
head(test.Y, 12)
```

```
##       [,1]
##  [1,]    0
##  [2,]    0
##  [3,]    0
##  [4,]    0
##  [5,]    0
##  [6,]    0
##  [7,]    0
##  [8,]    0
##  [9,]    0
## [10,]    1
## [11,]    1
## [12,]    1
```

References

Implementing the logistic regression algorithm using gradient descent in R that the minimizes negative log likelihood function.

1. https://machinelearningmastery.com/gradient-descent-for-machine-learning/
2. https://machinelearningmastery.com/logistic-regression-for-machine-learning/
3. https://www.youtube.com/watch?v=xsAfedKOf-A
4. https://towardsdatascience.com/gradient-descent-training-with-logistic-regression-c5516f5344f7
5. http://theautomatic.net/2018/10/02/how-to-build-a-logistic-regression-model-from-scratch-in-r/