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.

- Use the Auto data set in the ISLR2 library and the training algorithm from lectures.
- Put 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.

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

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

z <- seq(-10, 10, 0.0001)
plot(z, sigmoid(z), xlab="z")
grpline <- 20, lwd = 36, col= "black", lty = "dotted",
    lwd = 3pt"left", mar=c(bm + 7mm)
```

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

```
# Function to predict output
Prediction <- function(w, x)
  # w is weights
  p <- sigmoid(dot(w, x))
  # cat("p", p)
  if (is.nan(p, TRUE))
    if (p[1] >= 0.5)
      return 1
    else
      # cat("p0")
      return 0
  return (f.nan)
}
```

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

```
# Use gradient descent iteratly the final intercept and coefficient values
# d is attributes
# n is rows
# X is training rows
# y is training rows
# w is number of steps
# lr is learning rate
# nd is the weight
# ll is log-likelihood
GD <- function(X, y, lr, n=NULL){

  ll <- c(rep(0,n), 0.5)
  w0 <- as.matrix(c(rep(0,n), -7.5 ))

  ll <- ll * 2/n * 1/n %*% t(sigmoid(X %*% w0) - y)

  if (is.nulll) {
    print("attributes!!! n)
    ll <-
    w0 <- w0 -
    ll <- ll * 2/n * 1/n %*% t(sigmoid(X %*% w0) - y)
  }
  # grad <- Prob(X=1|w)(Z_i-Y)Z_i(Z_i)
  # w <- w - lr*(1/n)*(Z_i-Y)Z_i
  return(list(w=w,ll=ll,nd=nd))
}
```

2. Task 2.

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

$$\text{high} = \begin{cases} 1 & \text{if mpg >= 25 otherwise 0} \end{cases}$$

- 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.)
- Report the final intercept and coefficients, 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 the variable high
high <- ifelse(mpg >= 25,1,0)
Auto <- data.frame(Auto,high)
summary(Auto)
```

```
##       mpg          cylinders      displacement    horsepower       weight
##  Min.   :9.00   Min.   :3.000   Min.   :68.0   Min.   :46.0   Min.   :1613
```

2.3. Selecting only 5 attributes.

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

```
##  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))
Normalized_Auto = scale(subset(Auto,select=c(-origin,-high)))
# Combining the data set
Auto = data.frame(Normalized_Auto,Dummy_Auto)
library(boot)
```

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

```
boot::cv.glm(Auto,GD(d,0.1,0.0001,d))$delta[1]
```

horsepower	weight	year	origin	high
0.6633561	0.6237403	-1.625241	1	0
1.5725648	0.8422577	-1.625241	1	0
1.1829849	0.5390651	-1.625241	1	0
1.1829849	0.5361002	-1.625241	1	0
0.9329850	0.3544969	-1.625241	1	0
2.4290426	1.0951498	-1.625241	1	0
3.0014843	1.4204517	-1.625241	1	0
2.0733663	1.5710552	-1.625241	1	0
3.1313843	1.7040398	-1.625241	1	0
2.1329944	1.6370909	-1.625241	1	0
1.7329947	0.8650599	-1.625241	1	0
1.4429948	0.7432646	-1.625241	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.

```
# Random
set.seed(0405)
# Spliting the data into training and test data
new_data = sample(1:nrow(Auto), nrow(Auto))
# The training and test data set that will be used throughout this assignment
train_data = 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(mpg,1,dim(train))
auto_test <- cbind(mpg,1,dim(test))
# Making the response
colnames(auto_train)[1] = "response"
colnames(auto_test)[1] = "response"
# Removing the name
auto_train$name = NULL
auto_test$name = NULL
```

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

```
train_X <- as.matrix(auto_train[-response,train])
train_Y <- as.matrix(auto_train[response,train])
test_X <- as.matrix(auto_test[-response,
  test_Y <- as.matrix(auto_test[mpg,
```

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.

```
rm rnnbyme
```

$$ER(f, \{\mathbf{x}_n\}) = |\{\mathbf{x}_n : f(\mathbf{x}_n) \neq y_n\}|^{-1} \sum_{\{\mathbf{x}_n : f(\mathbf{x}_n) \neq y_n\}} |y - f|$$

- Try different values of the learning rate.

$$10^{-2} < \eta < 10^{-1}$$

- and of the number of training steps, T, so let your stopping rule be to stop after a given number of steps.
- Give a small table of and and training EER for each.

Computing the coefficients and the intercept of the training data set. And, comparing logistic regression model outputs built from first principles with the built-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 test data set using glm function:

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

```
## (Intercept)  horsepower      weight        year      origin
##  -0.7408667   0.1121566   0.3425823   0.1126007   0.4962773
```

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

```
# Model training
Train.GD = GD(train.X, train.Y, 0.01, d)
print(Train.GD$w[dim(train)])
```

```
##  [1]  horsepower  weight  year  origin
## [1,] 1.1.9999999 -1.111196 -1.0199999 -1.1199999 0.0000779
```

Training the model on test data set using glm function:

```
# Model training
Train.GD = GD(train.X, train.Y, 0.01, d)
print(Train.GD$w[dim(train)])
```

```
## [1]  horsepower  weight  year  origin
## [1,] 1.1.9999999 -1.111196 -1.0199999 -1.1199999 0.0000779
```

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

```
Test.GD = GD(test.X, Test.Y, 0.01, d)
print(Test.GD$w[dim(Test)])
```

```
##  [1]  horsepower  weight  year  origin
## [1,] 1.1.9999999 -1.111196 -1.0199999 -1.1199999 0.0000779
```

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

Calculating the error rate on the test set:

```
# GD.Error Rate of the trained model on the test set.
# Test.GD$w[dim(Test)]
test.coef = c(Test.GD$w[dim(Test)],0)
test.predicted <- Prob(X=1|test.X, test.coef)
# Error rate on the training set
cat("Test ER : ", Test.er)
```

```
## Test ER :  0.1121566
```

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

```
## [1] 0.1121566
```

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

```
# Trying different values of learning rate
Train.GD=c()
Test.GD=c()
learning.rate=c(10^-2,0.0001,0.1,0.001)
for(i in 1:4){
  train.gd <- GD(train.X, train.Y, lr, 1e-3)
  Train.GD[i] <- train.gd$ll[length(train.Y)]
  test.gd = c(Test.GD, Test.gd$ll)
  Train.GD = c(Train.GD, train.gd$ll)
  learning.rate[i] <- c(learning.rate, lr)
}
```

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

```
# Table of test and training EER
# GD.Error Rate of the trained model on the test set
kable(rbind(Train.GD,Test.GD),
      col.names=c(learning.rate))
```

Learning Rate	Train.EER	Test.EER
0.000001	0.6023409	0.3025449
0.000001	0.6429571	0.5357143
0.000001	0.3061020	0.7955934
0.000001	0.4367755	0.4561037
0.000001	0.4358736	0.4153563
0.000001	0.7764842	0.4367755
0.000001	0.2376610	0.3344860
0.000001	0.5324561	0.4561037
0.000001	0.3162265	0.1478582
0.000001	0.2552420	0.3032004
0.000001	0.4123563	0.2264157
0.000001	0.3552420	0.2280818

5. Task 5.

Try different stopping rules, such as:

- stop when the size of the objective function does not change.
- By more than 3% of the initial value over the last 10 training steps.

```
# 5.4
# Stopping rule 1
while (TRUE) {
  value <- c()
  for (i in 1:10){
    GD_1 = GD(train.X, train.Y, 0.01, d)
    Train.ER <- c(Train.ER, GD.ER$ll)
    value <- c(value,GD_1 - (Train.ER[i-1]*100))
    }
  for (m in 1:length(value, 10)) {
    if(abs(m) >= 3)
    print("3 = 3%")
    }
  if (num >= 10){
    break
  }
  if(i >= 10){
    break
  }
}
```

```
head(Train.GD)
```

```
## [1] 0.2324963 0.2324963 0.2324963 0.2324963 0.2323963 0.2323963 0.2324963
## [8] 2.0.2324963 0.2323963 0.2324963 0.2324963 0.2324963 0.2327963
```

```
print(num)
```

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

```
print(GD_B)
```

```
## [1] 27.94002
```

6. Task 6.

Run your logistic regression program for:

- a fixed value of *d* and for a fixed stopping rule 100 times.
- Report the average and standard deviation of the error rate.
- 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 EER
- Report the average and standard deviation.

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

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

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

```
# Box-plot
boxplot(test.ER.Fixed, xlab= c.GD, cex=0.5 = TRUE,
        lwd = 2, ylim=c(0,1),
        main="Boxplot of test EER",
        xlab="Test EER for 100 cases",
        ylab="EER")
```

```
# Box-plot
boxplot(test.ER.Fixed, xlab= c.GD, cex=0.5 = TRUE,
        lwd = 2, ylim=c(0,1),
        main="Boxplot of test EER",
        xlab="Test EER for 100 cases",
        ylab="EER")
```

The test EERs for 100 cases

7. Task 7.

Repeat the experiments in items 4-6

- Replacing the training procedure as follows:
- Train GD, D is a linear-logistic gradient descent with different values of the initial weights; and
- then choose the prediction rule with the best training objective.

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 coefficient and error rates will be computed.

```
# Model
GD.Four <- c()
coef1 = c()
coef2 = c()
coef3 = c()
coef4 = c()
for (i in 1:4){
  train.model = GD(train.X, train.Y, 0.01, 10, 1)
  GD.Four <- c(GD.Four, train.model$ll)
  if (i == 1){
    coef1 <- c(coef1, train.model$w[1])
  }else if (i == 2){
    coef2 <- c(coef2, train.model$w[2])
  }else if (i == 3){
    coef3 <- c(coef3, train.model$w[3])
  }else if (i == 4){
    coef4 <- c(coef4, train.model$w[4])
  }
}
```

7.2. Printing four error rates.

```
print(GD.Four)
```

```
## [1] 0.3387778 0.3239263 0.2273561 0.3143667
```

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

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

```
## [1] 0.3239263
```

Getting coefficients of the minimum error rate.

```
# Coefficient GD_Four
Best.coef = c(coef1, coef2, coef3, coef4)
print(Best.coef)
```

```
## [1] "numeric"
```

```
Test.w = Best.coef[-1]
Best.weight = c(Intercepts=Best.coef[1] Coefficients[-1])
```

Predicting by using coefficients that have lowest error rate:

```
Test.Prob = Prob(X=1|Test.X, Best.weight)
head(Test.Prob, 12)
```

```
## [1] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
```

The actual labels.

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

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

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=xdB4KiP-J4
4. https://towardsdatascience.com/gradient-descent-training-with-logistic-regression-c95185f44457
5. http://linasmakenas.net/2018/10/03/how-to-build-a-logistic-regression-model-from-scratch-in-r/