Assignment 1

Rishabh Taneja

Northeastern University

ALY 6020: Predictive Analytics

FALL ‘18

Assignment Completion Date: 28th April 2019

Instructor Name: Justin Rodgers

**INTRODUCTION**

In this assignment I am going to perform various regression algorithms and other mathematical functions and try to interpret the results for the analysis of different datasets. A few regressions are linear regression, multiple regression, logistic regression and GLM or generalized linear modeling. I will build our own model and train it on the training dataset, and to check for the accuracy and the feasibility of my model, it is very important to test the model on a test data set. To perform the modeling and analysis, I will import some useful packages in the beginning and thus use it throughout.

1. **R-programming: user-defined functions**
   1. Create a user defined function in R taking two arguments to exponentiate arg1 to the arg2 power.

**exponentional\_function <- function(a,b){**

**return(a^b)**

**}**

A picture containing object

Description automatically generated

This part of the assignment tests our skills to perform mathematical calculations and to create our own function. Here I created my own function which takes two arguments a and b as parameters and return the exponential value of a^b. The above result shows the calculation of 2^3 which is 8.

1. **Applied Predictive Modeling: Linear regression review**
   1. Load the data <http://data.princeton.edu/wws509/datasets/salary.dat> into R

**salary\_data <- read.table("https://data.princeton.edu/wws509/datasets/salary.dat", header=TRUE)**

**head(salary\_data)**

**A close up of a keyboard

Description automatically generated**

Here I am going to use read.table() function since the data I am provided with is not properly divided into columns. For ease of use, this function will create columns and separate them according to their respective fields.

The head() function provides us with the first few rows of the data along with the names of the field.

Variables are:

Sx – sex

Rk – rank

Yr – year

Dg – degree

Yd – year since degree

Sl – salary

* 1. **Split data into train/test based on a 80/20 split (use seed: 123)**

**set.seed(123)**

**Splitting\_data <- sample(x=1:nrow(salary\_data), size=0.80\*nrow(salary\_data))**

**train\_data = salary\_data[Splitting\_data,]**

**test\_data = salary\_data[-Splitting\_data,]**

**view(test\_data)**

**A screen shot of a computer

Description automatically generated**

Here I have set our seed value at 123 so that our results always remain consistent. I have created a variable splitting\_data where I have used sample() function to split the data into 80% for training and 20% of the actual data into test data set.

The sample function takes in parameters such as the range of values, the size of the dataset to be created which is multiplied by the number of rows present in the salary\_data, which is our main source of data.

* 1. **Fit model with train data to predict salary based on sex, rank, year, degree, and years since degree was awarded**

**fitting <- lm(sl~factor(sx)+factor(rk)+yr+factor(dg) + yd, data=train\_data)**

**predict\_train <- data.frame(predict(fitting, newdata = train\_data))**

**A screen shot of a computer

Description automatically generated**

To fit a model, we will use lm() function where the predicted variable would be the salary(sl) variable and the rest of them would be the predictor variables. Since there are 3 factors in the dataset, we will explicitly let the linear model know they are factors.

We will use predict() function to predict the values of the salaries with the train\_data. Above results shows the predicted values of the salaries. Since the values are in the decreasing order, the numbers are a bit jumbled on the left side. We can notice that the predicted values are quite close to the actual values of the salaries present in the dataset.

* 1. **Interpret model results**

**A screenshot of a cell phone on a table

Description automatically generated**

We have the residual values which is basically the difference between the actual value of the observed response variable and the predicted values of the variable. Since our mean value is not 0, it means that the residuals are not symmetrical. This means that there are some values which fall far from the actual observed values. For better understanding, we have plotted residual vs fitted graph which tells us that there are some points which are away from the observed value.

The estimated coefficient tells us about the intercept. The intercept is the average value of the salary ($16601) required for all the males and females to have almost the same salary. The slope tells us that for every dollar increase in the number of males, the salary goes down by $532.87. The standard error tells us that the value of salaries can vary by $861. The t-test values tell us that the values are far away from 0. This indicates that there exists a relationship among these variables and thus reject the null hypothesis. The 3 stars represents that they have highly significant p values. A small p values allows us to reject the null hypothesis and verify that there is a relationship among them. The residual error signifies the average amount that salary would deviate from other predictor variables which is 2002. R squared value is 0.88 which means that 88% of the variance found in the salaries can be explained by the predictor variables.

* 1. **Does model meet linear regression assumptions?**

A screenshot of a cell phone

Description automatically generated

The key assumptions made are linear relationship, No auto-correlation, multivariate normality, Homoscedasticity and no multicollinearity. Considering these assumptions, No, the model does meet all the assumptions, but follow some of them. For example, take the Q-Q plot, the points do not follow the line. Thus, they are deviating from the mean. However, the residual vs fitted graph shows a straight red line thus they follow linear relationship.

* 1. **Use trained linear regression model to make predictions with test data**

**predicted\_value <- data.frame(predict(fitting, newdata = test\_data))**

I have created a data frame using data.frame function and stored in into a variable names predicted\_value. I have used the predefined function, predict() to predict the values of the salaries based on the previously trained model and on the test dataset.

A screenshot of a cell phone

Description automatically generated

* 1. **Assess prediction error/accuracy with appropriate metrics comparing predicted vs. observed values. Interpret accordingly**

**predicted\_value <- data.frame(predict(fitting, newdata = test\_data))**

**observed\_value <- data.frame(test\_data$sl)**

**SSE <- sum((observed\_value - predicted\_value) ^ 2)**

**SST <- sum((observed\_value - mean(observed\_value)) ^ 2)**

**R2 <- 1 - SSE/SST**

**RMSE <- sqrt(sum((predicted\_value - observed\_value)^2)/length(observed\_value))**

**MAE <- mean(abs(observed\_value - predicted\_value))**

**RSS <- sum((predicted\_value - observed\_value)^2)**

Screen of a cell phone

Description automatically generated

The RMSE stands for root means squared value. These are calculated in order to check the performance of the model. The range is between 0-infinity. Lower the value of RMSE, better the model. Here, we can note that we have got the value of RMSE as 3508 which indicates the value is somewhat average and does not justify that the model is a great fit. However, this is quite the case when the numbers are quite huge. This denotes that the predicted values are 3508 root mean sq distant from the regression line. To lower this value, we can try tuning the values.

1. **Applied Predictive Analytics: Logistic Regression.** 
   1. **Load binary dataset using the following code**:

sales <- read.csv("http://ucanalytics.com/blogs/wp-content/uploads/2017/09/Data-L-Reg-Gradient-Descent.csv")

sales$X1plusX2 <- NULL

var\_names <- c("expenditure", "income", "purchase")

names(sales) <- var\_names

#Standardize predictors (X1, X2)

sales$expenditure <- scale(sales$expenditure, scale=TRUE, center = TRUE)

sales$income <- scale(sales$income, scale=TRUE, center = TRUE)

Context: this data derives from a marketing campaign. Marketers wanted to know the relationship between previous product expenditures (X1), consumer income (X2), and probability of purchasing their product (Y).

**sales <- read.csv("http://ucanalytics.com/blogs/wp-content/uploads/2017/09/Data-L-Reg-Gradient-Descent.csv")**

**sales$X1plusX2 <- NULL**

**var\_names <- c("expenditure", "income", "purchase")**

**names(sales) <- var\_names**

**#Standardize predictors (X1, X2)**

**sales$expenditure <- scale(sales$expenditure, scale=TRUE, center = TRUE)**

**sales$income <- scale(sales$income, scale=TRUE, center = TRUE)**

* 1. **Separate sales data into train/test sets (75/25) using seed(5689).**

**set.seed(5689)**

**sales\_split <- sample(x=1:nrow(sales), size=0.75\*nrow(sales))**

**train\_dataset = sales[sales\_split,]**

**test\_dataset = sales[-sales\_split,]**

Here I have set our seed value at 5689 so that our results always remain consistent. I have created a variable sales\_split where I have used sample() function to split the data into 75% for training and 25% of the actual data into test data set.

The sample function takes in parameters such as the range of values, the size of the dataset to be created which is multiplied by the number of rows present in the sales dataset, which is our main source of data.

* 1. **Train logistic regression GLM model to predict Y from X1-X2**

**gen\_linear\_model <- glm(purchase~expenditure+income, family=binomial(link='logit') , data=train\_dataset)**

**summary(gen\_linear\_model)**

**par(mfrow=c(3,2))**

**plot(gen\_linear\_model)**

**A screenshot of a cell phone on a table

Description automatically generated**

A close up of a map

Description automatically generated

The deviance residual is the measure of the goodness or badness of the model. The higher the values, worst the fit. We have the null deviance as 415 on 299 dof and residual deviance as 277 on 297 dof. This means that the total residual deviance as 138 with loss of 2 degrees of freedom. Fisher scoring is the measure of maximum likelihood. The mean is close to 0 (0.17) which means that the predicted or fit model has values closer to mean.

* 1. **Test trained logistic regression model using test data (predict Ys using trained model and convert probabilities to 0s/1s).**

**pred\_value <- predict(gen\_linear\_model,train\_dataset,type = "response")**

**train\_value <- ifelse(pred\_value > 0.5, 1,0)**

**train\_value\_table <- table(predicted=pred\_value, actual = train\_dataset$purchase)**

**train\_value\_table**

A screen shot of a smart phone

Description automatically generated

The total cross value turns out to be 63. These indicates that total 63 values were wrongly predicted. This can be fixed using tuning method. By training the model again and again would make these numbers smaller. Since the salary values are quite high, these are bound to happen.

**test\_value <- predict(gen\_linear\_model,test\_dataset,type = "response")**

**test\_y <- ifelse(test\_value > 0.5, 1,0)**

**test\_y <- data.frame(test\_value)**

**test\_y <- test\_y[1:100,]**

**test\_table <- table(predicted=test\_y, actual = test\_dataset$purchase)**

**test\_table**

**A close up of a screen

Description automatically generated**

The total cross value turns out to be 20. The values are a bit far from each other which does not indicate a good sign. This indicated that 20 values were wrongly predicted.

Here, we used the predict () function to predict the values of the variable set forth in the generalized linear model built. The type we have set to response. This would allow us to convert it into 0 and 1 compared to the predicted value.

The accuracy is found using the true positive by the total value.

By using these values, we can calculate sensitivity and specificity. This can help us determine how many values were not correctly predicted out of the total and can work on it to improve the accuracy of the model.

* 1. **Evaluate by comparing mismatch in observed/predicted 0s/1s. How did you model perform?**

**error\_calculation(actual = train\_dataset$purchase, predicted = train\_value)**

**error\_calculation(actual = test\_dataset$purchase, predicted = test\_y)**

**A screenshot of a cell phone

Description automatically generated**

The error calculation for train dataset turned out to be 0.21 which is very less and it is an indication of a good model. The error calculation for the test dataset is quite less and almost the same which is a great thing for a model.

**predicting\_values <- prediction(test\_y,test\_dataset$purchase)**

**performance\_value <- performance(predicting\_values,measure = "tpr", x.measure = "fpr")**

**plot(performance\_value)**

**auc(test\_dataset$purchase,test\_y)**

**A picture containing object

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

From the above results, We see that the graph is increasing and then getting steadied. This indicates that there might be a lot of false positive values for same true positive value.

AUC turned out to be 0.89 is which higher so model is good.

In the train table, we predicted 237 values out of 300 possible values which indicates that the model is a great fit and is some what accurate. On the other side, for test\_table, we have 80 correct outcomes out of 100, which indicates an accuracy of 80%.

1. **Predictive Analytics Theory & Applied: Gradient Descent and Generalized Linear Modeling via Maximum Likelihood Estimation**
   1. **Load data. Use entire sales dataset from Question 3 above**

**sales <- read.csv("http://ucanalytics.com/blogs/wp-content/uploads/2017/09/Data-L-Reg-Gradient-Descent.csv")**

**sales$X1plusX2 <- NULL**

**var\_names <- c("expenditure", "income", "purchase")**

**names(sales) <- var\_names**

**#Standardize predictors (X1, X2)**

**sales$expenditure <- scale(sales$expenditure, scale=TRUE, center = TRUE)**

**sales$income <- scale(sales$income, scale=TRUE, center = TRUE)**

* 1. **Logistic regression via user-defined maximum likelihood function. Create user-defined maximum likelihood function to estimate the parameters that maximize the log likelihood of the logistic density function. Use your logistic log likelihood function and the R optim() function to estimate the parameters (for B0, B1, and B2) that maximize the log likelihood given the observed sales data. You may use the “Logistic Regression Maximum Likelihood Code Outline.R” as a starting place. Just fill in the empty pieces denoted by brackets []. For a review of maximum likelihood estimation specific to logistic regression see:** [**https://www.statlect.com/fundamentals-of-statistics/logistic-model-maximum-likelihood**](https://www.statlect.com/fundamentals-of-statistics/logistic-model-maximum-likelihood)**. The log likelihood equation should look something like:**

**loglik <- sum(-y\*log(1 + exp(-(x%\*%beta))) - (1-y)\*log(1 + exp(x%\*%beta)))**

**logl <- function(theta,x,y){**

**y <- y**

**x <- as.matrix(x)**

**beta <- theta[1:ncol(x)]**

**loglik <- sum(-y\*log(1 + exp(-(x%\*%beta))) - (1-y)\*log(1 + exp(x%\*%beta)))**

**return(-loglik)**

**}**

**theta.start = rep(0,3)**

**names(theta.start) = colnames(X)**

**mle = optim(theta.start,logl,x=X,y=Y,hessian=T)**

**out = list(beta=mle$par,se=diag(sqrt(solve(mle$hessian))),ll=2\*mle$value)**

**#Get parameter values from out object and beta attribute**

**#[call "out" object and "beta" attribute, separated by $, as in object$attribute]**

**betaVal <- out$beta**

**print(out)**

A screenshot of a cell phone

Description automatically generated

* 1. **Logistc regression via user-defined gradient descent function. Create a user-defined gradient descent function to estimate beta parameters (B0, B1, B2) using the sales data. You may use the “Logistic Regression Gradient Descent Code Outline” as a starting place. For a review of gradient descent specific to logistic regression see:**

[**http://ucanalytics.com/blogs/gradient-descent-logistic-regression-simplified-step-step-visual-guide/**](http://ucanalytics.com/blogs/gradient-descent-logistic-regression-simplified-step-step-visual-guide/)**. The gradient descent cost function should look something like:**

**(1/m)\*sum((-Y\*log(g)) - ((1-Y)\*log(1-g))). The gradient descent delta formula**

**should look something like: (t(X) %\*% error) / length(Y).**

**logistic\_gradient\_descent <- function(alpha, iterations, beta, x, y){**

**for (i in 1:iterations) {**

**error <- (X %\*% beta - y)**

**delta <- (t(X) %\*% error) / length(Y)**

**beta <- beta - alpha \* delta**

**}**

**return(list(parameters=beta))**

**}**

**#Set initial parameters for gradient descent**

**# Define learning rate and iteration limit**

**initial\_alpha <- 0.01 #learning rate**

**num\_iterations <- 10000 #number of times we'll run the loop in the function**

**empty\_beta <- matrix(c(0,0,0), nrow=3) # initialized parameters (matrix of 0s)**

**#Run logistic regression gradient descent to find beta parameters**

**#[fill in all of the arguments for the user-defined logistic gradient descent function]**

**output <- logistic\_gradient\_descent(alpha = initial\_alpha, iterations = num\_iterations, beta = empty\_beta, x = X, y = Y)**

**#Get final estimated parameters from our output object:**

**#[call object "output" and stored attribute "parameters", separated by a $, as in object$attribute]**

**print(output)**

**A screenshot of a cell phone

Description automatically generated**

* 1. **Fit a generalized linear model logistic regression model using R’s built-in glm() function using the full sales dataset.**

**sales <- read.csv("http://ucanalytics.com/blogs/wp-content/uploads/2017/09/Data-L-Reg-Gradient-Descent.csv")**

**variable\_names <- c("expenditure", "income", "purchase")**

**names(sales) <- variable\_names**

**#Standardize predictors (X1, X2)**

**sales$expenditure <- scale(sales$expenditure, scale=TRUE, center = TRUE)**

**sales$income <- scale(sales$income, scale=TRUE, center = TRUE)**

**model <- glm(sales$purchase~sales$income+sales$expenditure,data=sales,family=binomial())**

**summary(model) # display results**

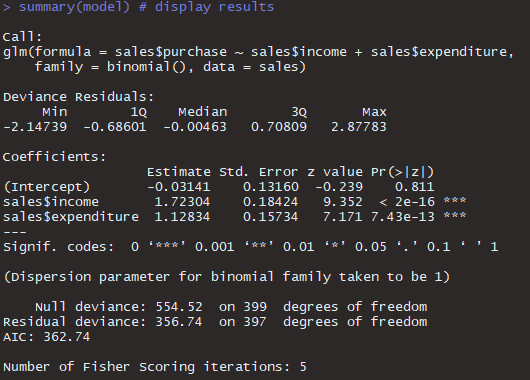
**confint(model) # 95% CI for the coefficients**

**exp(coef(model)) # exponentiated coefficients**

**exp(confint(model)) # 95% CI for exponentiated coefficients**

**predict(model, type="response") # predicted values**

**residuals(model, type="deviance") # residuals**

****

**A black sign with white text

Description automatically generated**

****

**A close up of a keyboard

Description automatically generated**

**A close up of text on a white background

Description automatically generated**

**A black sign with white text

Description automatically generated**

**par(mfrow=c(3,2))**

**plot(model)**

**A close up of a map

Description automatically generated**

* 1. **Summarize the parameters/weights (B0, B1, B2) from each of these three optimization approaches (parts b, c, and d). How do the results from these three approaches compare?**

We generally follow logistic regression when the variables we the dependent variables are in the binary form. Here, purchase variable is the dependent variable and is in binary form. We have some set rules from the results where 1 represents that a purchase has been made and 0 means otherwise. We have income and expenditure as response variables and purchase being the predicted variable.

For maximum likelihood, we follow probability distribution which takes in parameters expenditure and income. The final value of log likelihood turned out to be 356.73. The values of coefficient of Expenditure and income are 1.12 and 1.72 respectively which suggest that they are positively correlated to each other. The standard error turned out to be 0.15 and 0.18 for expenditure and income respectively which indicates the model is well fit.

For Gradient descent, it is used when we want to know the minimum and maximum values of a function using the optimization technique. The values of the coefficient are 0.16 and 0.26 for expenditure and income respectively which indicates that they are positively correlated to each other with minimal deviation among them. Such small values suggest that the model is a great fit.

The deviance residual is the measure of the goodness or badness of the model. The higher the values, worst the fit. We have the null deviance as 554 on 399 dof and residual deviance as 356 on 397 dof. This means that the total residual deviance as 198 with loss of 2 degrees of freedom. Fisher scoring is the measure of maximum likelihood. The mean is almost 0 which means that the predicted or fit model has values closer to mean.

The normal Q-Q plot shows the values follow the 45-degree line which means it follows linear relationship.

The exp coefficients of the income and expenditure tells us that for every 1 unit increase in the purchase, the income will increase by 5.6 units. We have the observed and predicted values with us. We found out the residual values which is basically the difference between the observed and the predicted values. Higher the residual values suggest that the model is bad whereas if the residual values are close to 0, this indicates that the model is a great fit. From the residual vs fitted graph, it indicates that the values almost/nearly follow logistic regression, or linear relationship in logarithmic form.

**CONCLUSION**

From the above problems, we can conclude a few things which are:

1. The summary of the linear model tells us whether the model is a good or a bad fit.
2. As close the mean value is to 0, best the model.
3. Greater the R sq. value, greater the accuracy and the model are a great fit.
4. Top 5 assumptions which a model must follow in order to be a great fit in linear regression are linear relationship, No auto-correlation, multivariate normality, Homoscedasticity and no multicollinearity.
5. The deviance residual is the measure of the goodness or badness of the model. The higher the values, worst the fit.
6. Gradient descent is used when we want to know the minimum and maximum values of a function using the optimization technique.
7. Follow logistic regression when the variables we the dependent variables are in the binary form.
8. Residual values are calculated by the difference between predicted and observed values. The less the value, best the model.
9. As far the values from 0 in the t-test, we can reject the null hypothesis stating there is a relationship among the variables.
10. Closer the values of the coefficient, better the positive linearity.

**CODE**

#1 Question

#a) funtion to perform arg1 exp arg2

exponentional\_function <- function(a,b){

return(a^b)

}

exponentional\_function(2,3)

#2 Question

#a) Load the dataset into R

#using read.table to separate all the data into individual columns

salary\_data <- read.table("https://data.princeton.edu/wws509/datasets/salary.dat", header=TRUE)

set.seed(123)

head(salary\_data)

Splitting\_data <- sample(x=1:nrow(salary\_data), size=0.80\*nrow(salary\_data))

train\_data = salary\_data[Splitting\_data,]

test\_data = salary\_data[-Splitting\_data,]

fitting <- lm(sl~factor(sx)+factor(rk)+yr+factor(dg) + yd, data=train\_data)

predict\_train <- data.frame(predict(fitting, newdata = train\_data))

summary(fitting)

par(mfrow=c(3,2))

plot(fitting)

predicted\_value <- predict(fitting, newdata = test\_data)

predicted\_value

observed\_value <- test\_data$sl

observed\_value

SSE <- sum((observed\_value - predicted\_value) ^ 2)

SSE

SST <- sum((observed\_value - mean(observed\_value)) ^ 2)

SST

R2 <- 1 - SSE/SST

R2

RMSE <- sqrt(mean((predicted\_value - observed\_value)\*\*2))

RMSE

MAE <- mean(abs(observed\_value - predicted\_value))

MAE

RSS <- sum((predicted\_value - observed\_value)^2)

RSS

#3

sales <- read.csv("http://ucanalytics.com/blogs/wp-content/uploads/2017/09/Data-L-Reg-Gradient-Descent.csv")

sales$X1plusX2 <- NULL

var\_names <- c("expenditure", "income", "purchase")

names(sales) <- var\_names

#Standardize predictors (X1, X2)

sales$expenditure <- scale(sales$expenditure, scale=TRUE, center = TRUE)

sales$income <- scale(sales$income, scale=TRUE, center = TRUE)

set.seed(5689)

sales\_split <- sample(x=1:nrow(sales), size=0.75\*nrow(sales))

train\_dataset = sales[sales\_split,]

test\_dataset = sales[-sales\_split,]

gen\_linear\_model <- glm(purchase~expenditure+income, family=binomial(link='logit') , data=train\_dataset)

summary(gen\_linear\_model)

par(mfrow=c(3,2))

plot(gen\_linear\_model)

pred\_value <- predict(gen\_linear\_model,train\_dataset,type = "response")

train\_value <- ifelse(pred\_value > 0.5, 1,0)

train\_value\_table <- table(predicted=train\_value, actual = train\_dataset$purchase)

train\_value\_table

test\_value <- predict(gen\_linear\_model,test\_dataset,type = "response")

test\_y <- ifelse(test\_value > 0.5, 1,0)

test\_y <- data.frame(test\_y)

test\_y <- test\_y[1:100,]

test\_table <- table(predicted=test\_y, actual = test\_dataset$purchase)

test\_table

error\_calculation <- function(actual,predicted){

mean(actual!= predicted)

}

error\_calculation(actual = train\_dataset$purchase, predicted = train\_value)

error\_calculation(actual = test\_dataset$purchase, predicted = test\_y)

library(ROCR)

predicting\_values <- prediction(test\_y,test\_dataset$purchase)

performance\_value <- performance(predicting\_values,measure = "tpr", x.measure = "fpr")

plot(performance\_value)

auc(test\_dataset$purchase,test\_y)

#4 question ##

sales <- read.csv("http://ucanalytics.com/blogs/wp-content/uploads/2017/09/Data-L-Reg-Gradient-Descent.csv")

sales$X1plusX2 <- NULL

var\_names <- c("expenditure", "income", "purchase")

names(sales) <- var\_names

sales$expenditure <- scale(sales$expenditure, scale=TRUE, center = TRUE)

sales$income <- scale(sales$income, scale=TRUE, center = TRUE)

#Predictor variables matrix

X <- as.matrix(sales[,c(1,2)])

#Add ones to Predictor variables matrix (for intercept, B0)

X <- cbind(rep(1,nrow(X)),X)

#Response variable matrix

Y <- as.matrix(sales$purchase)

logl <- function(theta,x,y){

y <- y

x <- as.matrix(x)

beta <- theta[1:ncol(x)]

loglik <- sum(-y\*log(1 + exp(-(x%\*%beta))) - (1-y)\*log(1 + exp(x%\*%beta)))

return(-loglik)

}

theta.start = rep(0,3)

names(theta.start) = colnames(X)

mle = optim(theta.start,logl,x=X,y=Y,hessian=T)

out = list(beta=mle$par,se=diag(sqrt(solve(mle$hessian))),ll=2\*mle$value)

#Get parameter values from out object and beta attribute

#[call "out" object and "beta" attribute, separated by $, as in object$attribute]

betaVal <- out$beta

print(out)

sigmoid <- function(z){

g <- 1/(1+exp(-z))

return(g)

}

#Cost Function

#Create user-defined Loss Function specific to logistic regression gradient descent

#[finish the equation for J - the logistic regression cost/loss function]

logistic\_cost\_function <- function(beta){

m <- nrow(X)

g <- sigmoid(X%\*%beta)

J <- (1/m)\*sum((-Y\*log(g)) - ((1-Y)\*log(1-g)))

return(J)

}

#Gradient Descent Function

#Create user-defined Gradient Descent Function specific to Logistic Regression

#[finish the equation for Beta - the updated weight (beta-alpha\*delta)]

logistic\_gradient\_descent <- function(alpha, iterations, beta, x, y){

for (i in 1:iterations) {

error <- (X %\*% beta - y)

delta <- (t(X) %\*% error) / length(Y)

beta <- beta - alpha \* delta

}

return(list(parameters=beta))

}

#Set initial parameters for gradient descent

# Define learning rate and iteration limit

initial\_alpha <- 0.01 #learning rate

num\_iterations <- 10000 #number of times we'll run the loop in the function

empty\_beta <- matrix(c(0,0,0), nrow=3) # initialized parameters (matrix of 0s)

#Run logistic regression gradient descent to find beta parameters

#[fill in all of the arguments for the user-defined logistic gradient descent function]

output <- logistic\_gradient\_descent(alpha = initial\_alpha, iterations = num\_iterations, beta = empty\_beta, x = X, y = Y)

#Get final estimated parameters from our output object:

#[call object "output" and stored attribute "parameters", separated by a $, as in object$attribute]

print(output)

#End of Logistic Regression via Gradient Descent Code Outline

#4d

sales <- read.csv("http://ucanalytics.com/blogs/wp-content/uploads/2017/09/Data-L-Reg-Gradient-Descent.csv")

variable\_names <- c("expenditure", "income", "purchase")

names(sales) <- variable\_names

#Standardize predictors (X1, X2)

sales$expenditure <- scale(sales$expenditure, scale=TRUE, center = TRUE)

sales$income <- scale(sales$income, scale=TRUE, center = TRUE)

model <- glm(sales$purchase~sales$income+sales$expenditure,data=sales,family=binomial())

summary(model) # display results

confint(model) # 95% CI for the coefficients

exp(coef(model)) # exponentiated coefficients

exp(confint(model)) # 95% CI for exponentiated coefficients

predict(model, type="response") # predicted values

residuals(model, type="deviance") # residuals

par(mfrow=c(3,2))

plot(model)

**REFERENCES**

1. Logistic classification model - Maximum likelihood estimation. (n.d.). Retrieved from <https://www.statlect.com/fundamentals-of-statistics/logistic-model-maximum-likelihood>
2. Roopam. (2018, September 16). Gradient Descent for Logistic Regression Simplified - Step by Step Visual Guide – YOU CANalytics-. Retrieved from <http://ucanalytics.com/blogs/gradient-descent-logistic-regression-simplified-step-step-visual-guide/>
3. Assumptions of Linear Regression. (n.d.). Retrieved from <https://www.statisticssolutions.com/assumptions-of-linear-regression/>
4. (n.d.). Retrieved from <https://feliperego.github.io/blog/2015/10/23/Interpreting-Model-Output-In-R>
5. Generalized Linear Models in R, Part 2: Understanding Model Fit in Logistic Regression Output. (2019, April 26). Retrieved from <https://www.theanalysisfactor.com/r-glm-model-fit/>
6. Humby, S. H. (n.d.). How to split data into training/testing sets using sample function. Retrieved from <https://stackoverflow.com/questions/17200114/how-to-split-data-into-training-testing-sets-using-sample-function>
7. HOME. (n.d.). Retrieved from <https://stats.idre.ucla.edu/r/dae/logit-regression/>