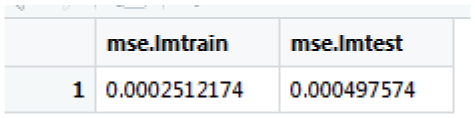
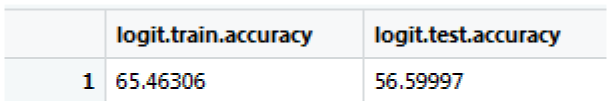
Report

I build two models logistic regression and linear regression. The mse for linear regression and the accuracy for the logistic regression show below:

Models Accuracy



	mse.lmtrain	mse.lmtest
1	0.0002512174	0.000497574

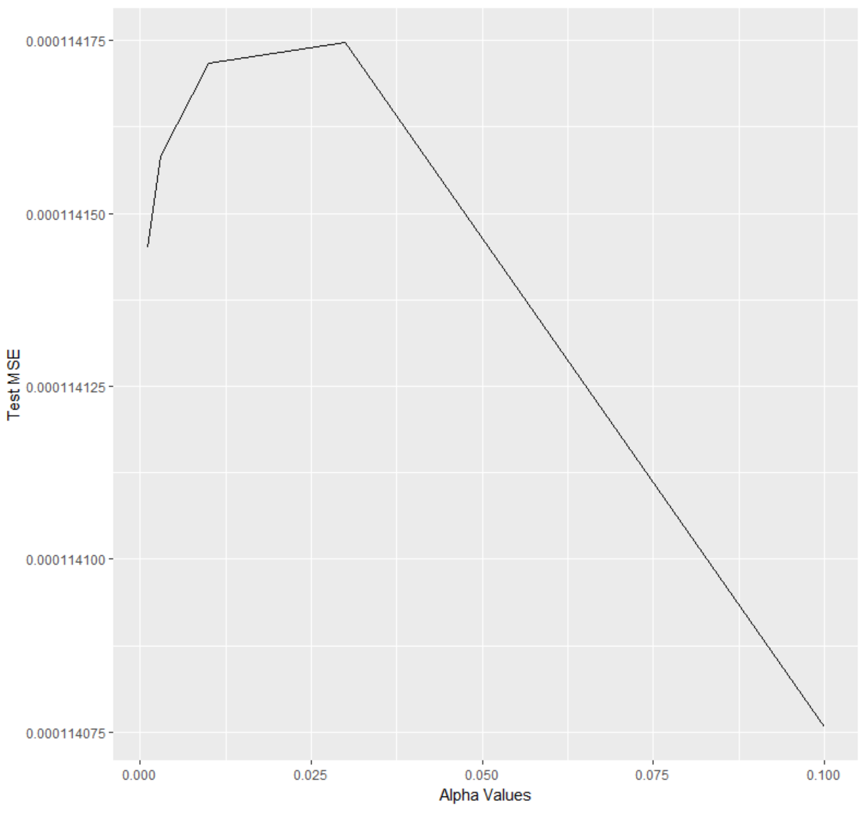


	logit.train.accuracy	logit.test.accuracy
1	65.46306	56.59997

I experiment changing the learning rate for the linear regression. The learning rate determines how big each step the gradient descent takes. If the learning rate is big enough, the gradient descent usually converges very fast. In this case, convergence threshold comes into play. Setting proper convergence threshold let us save time and resources without doing those iterations that decrease the cost value less. On the other hand, if the learning rate is too big, it will over shoot and can never find the global minimum. If the learning rate is too small, it takes long time for cost value converge. Therefore, setting proper learning rate is very important.

In my experiments (five pictures shown in the next two pages), with the first learning rate 0.001, the cost decreases the most- more than 0.00001. In the consecutive three pictures, with larger learning rate, the cost value decrease really less after certain point of time. This proves the importance of convergence threshold which can stop the interaction after cost value decrease less than really small value at each interaction. In fifth learning rate shows the situation where the cost value start to increase at some point as it has reach the global minimum and starts to go up the hill.

Mean Squared Error for Various learning rate

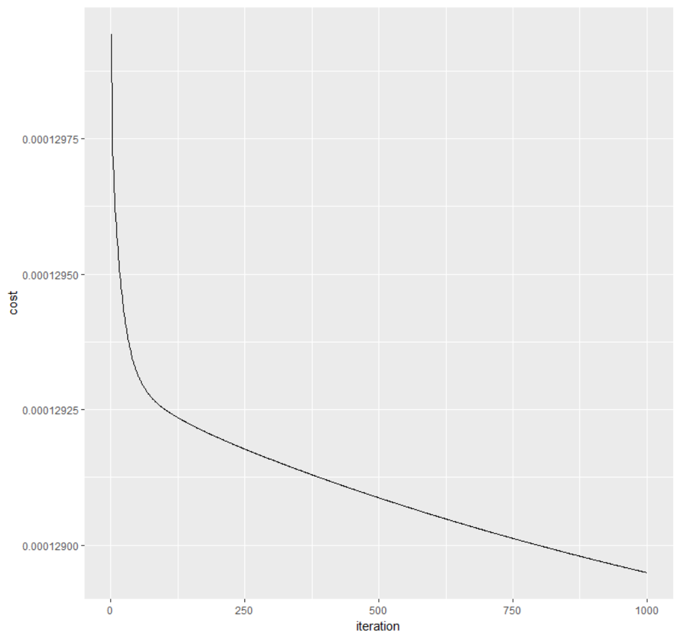


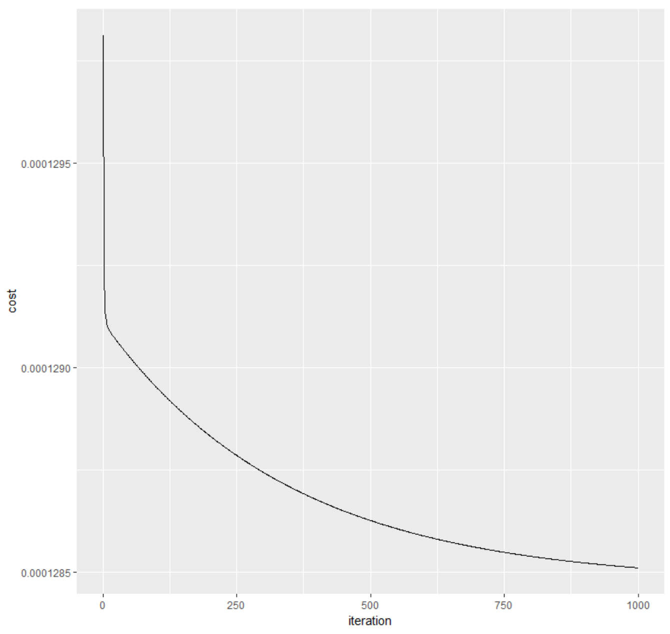
The plot above shows how the MSE varies while I using various learning rates for using all of the variables. While the learning rates become smaller, the error rate for training set is lowering as cost value keeps on decreasing, holding the iteration times and convergence threshold unchanged, and the coefficients are getting close to the optimized coefficients resulting lowest training error. However, while the models apply on the test set, the testing error rate will not keep decreasing while training error decreases. The test error goes up a little bit at the beginning and decrease after certain point when training errors are going down.

The best learning rate from my experiement is 0.1

Cost Value for Training set with Random 10 and Picked 10 Variables

(From left to right)





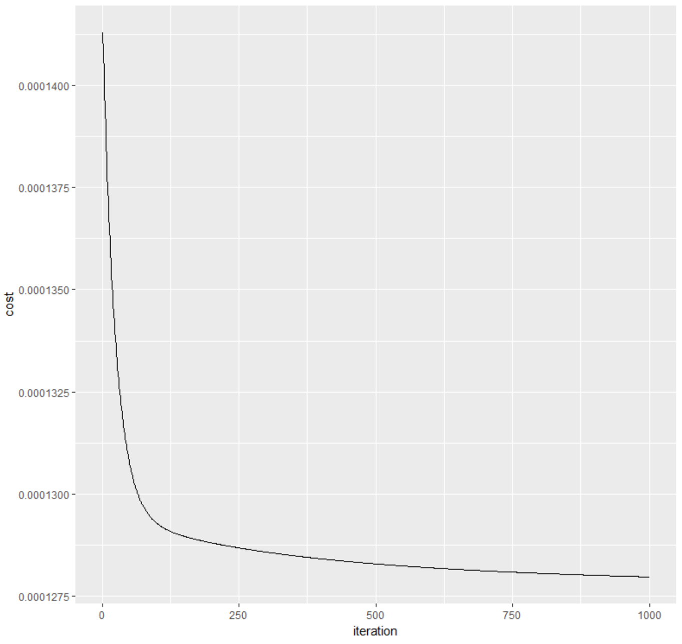
I randomly select 10 variables: avg_positive_polarity, num_self_hrefs, title_subjectivity, data_channel_is_world, n_non_stop_words, kw_max_max, LDA_02, self_reference_avg_sharess, max_negative_polarity, kw_min_avg.

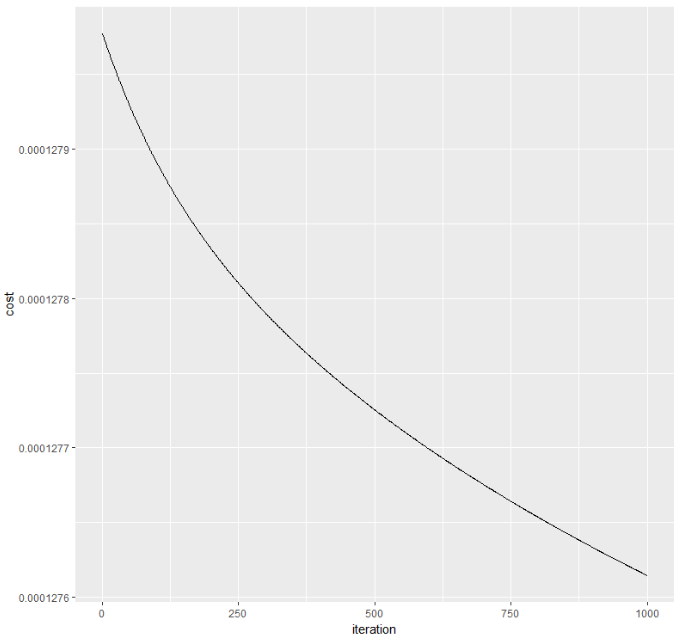
I choose these 10 variables: "num_imgs", "weekday_is_monday", "is_weekend", "abs_title_subjectivity", "abs_title_sentiment_polarity", "global_sentiment_polarity", "global_subjectivity", "max_positive_polarity", "max_negative_polarity", "average_token_length".

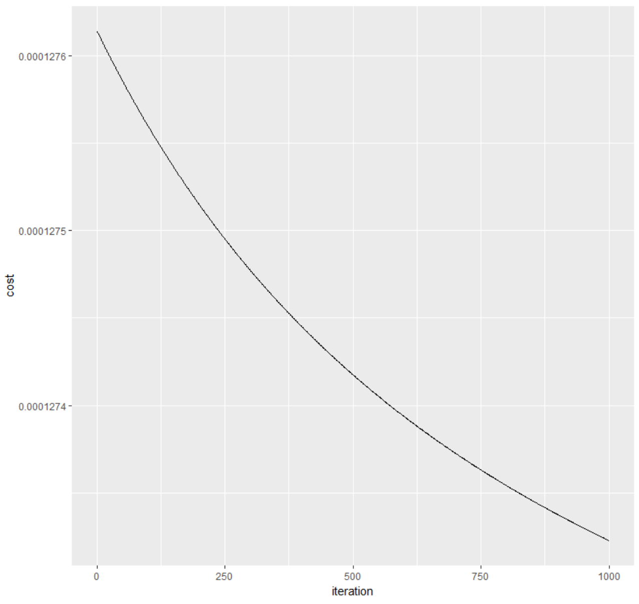
The cost value for training set with picked 10 variables has the lower value than the one with randomly selected 10 variables.

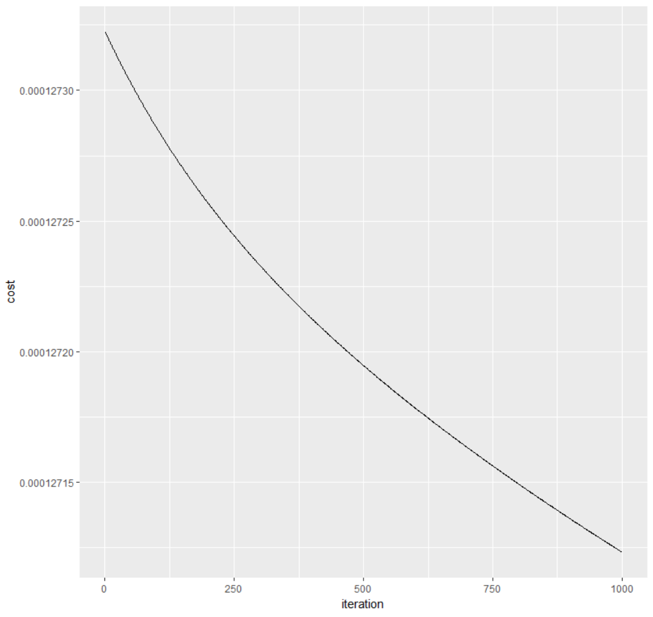
Alpha: 0.001, 0.003, 0.01, 0.03, 0.1

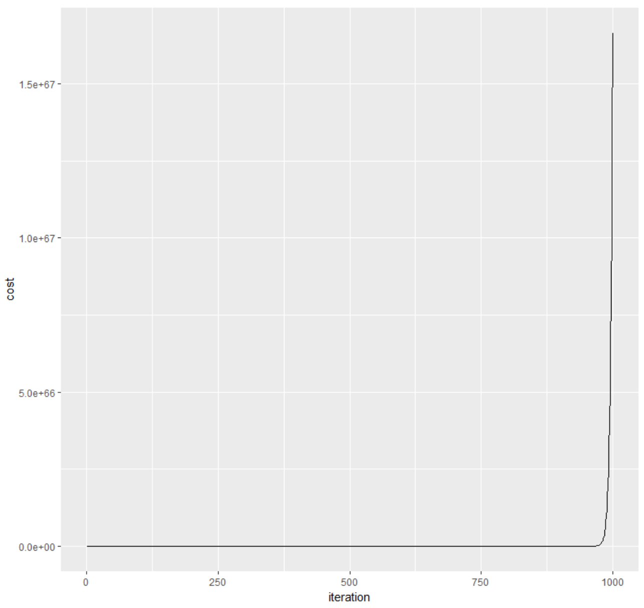
(From left to right)



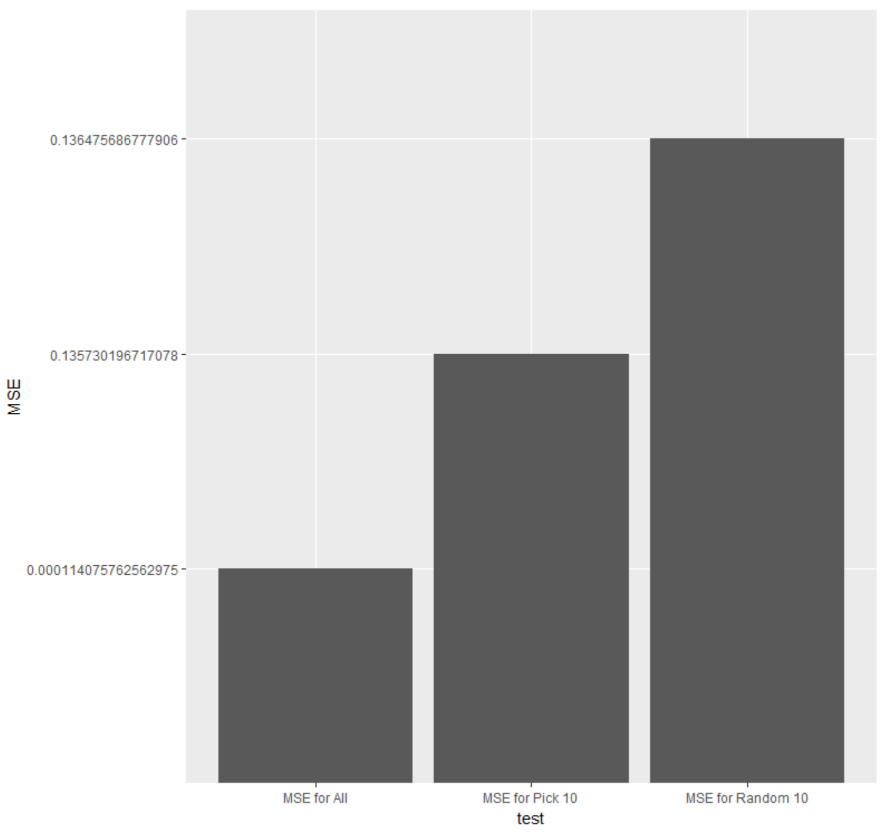








Summary for the MSE for Test sets with Different Variables

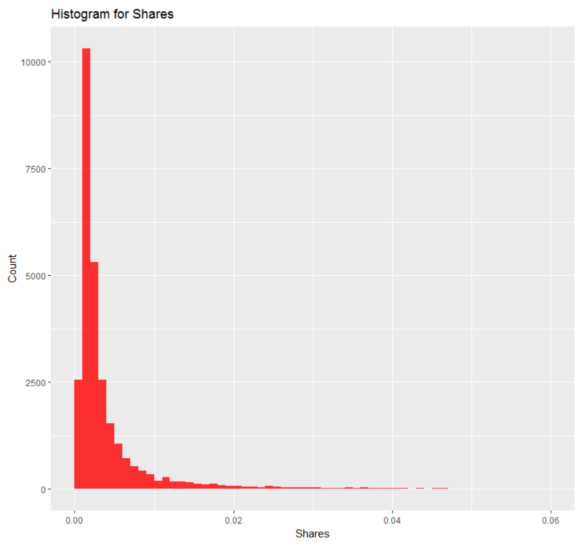


The MSE for the test set with all of the variables have lowest MSE which is 0.00011. The test set with random selected 10 variables have the highest MSE compared to the other two.

Logistic Regression

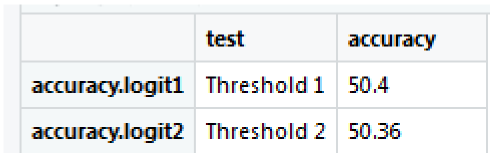
I constructed the threshold for the logistic regression as 0.5. I plot the distribution for the shares and find the median value of shares as the cutpoint for class 1 and class 0.

Distribution of Shares



This helps maintain the dataset as a balanced dataset. As occurrence of the class 0 and class 1 are fairly balanced, the threshold is reasonable to set as 0.5. If the predicted class is imbalanced, say, one class appears less 5% according to the whole dataset, it will result in really high false positive rate or high false negative rate. For these cases, we need to adjust the threshold so that the predicted value is more accurate.

Accuracy for the Logistic Regression with Different Thresholds

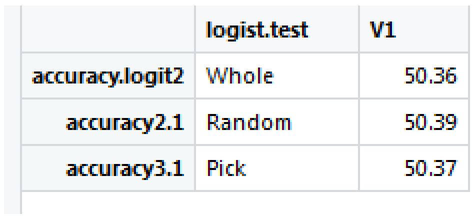


	test	accuracy
accuracy.logit1	Threshold 1	50.4
accuracy.logit2	Threshold 2	50.36

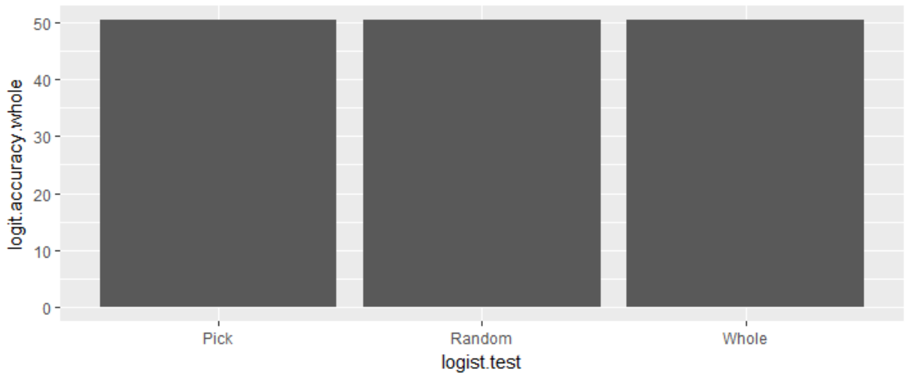
I experiment the parameter convergence threshold for logistic regression. Convergence threhold is a particular value set where the gradient descent stops at the point the cost value decrease less than this particular value. The first value for the convergence threhold is 1e-7 and the second one is 1e-3. According to the chart shown below, as the treshold increases, the accuracy for the linear regression is decreasing. One of the reasons is that the lower the threshold, the optimized the coefficients might be as cost value can keep on decreasing at more iterations than the one with value 1e-3.

The best convergence threhold is 1e-7 in this case.

Accuracy for Training set with Random 10 and Picked 10 Variables



	logist.test	V1
accuracy.logit2	Whole	50.36
accuracy2.1	Random	50.39
accuracy3.1	Pick	50.37



The accuracy for logistic regression for the test set with different variables vary less. The best accuracy is 50.39% for the test set with randomly selected 10 variables.

For constructing and returning best predict power of the models, it appears picking right parameters with learning rates, convergence threshold is very important. From the experiments, picking variables seem less important. However, in the real world, if you want a model can show its value, picking the logical variables and right variables that can create value for business is more important than training the best models- picking the right variables and giving good interpretation are more important.

Appendix

Code for the assignment:

```
pkg <-
c("ModelMetrics","gclus","devtools","reshape2","purrr","tidyr","ggplot2","caTools","corrplot","ROCR")

require("caTools")

require("ggplot2")

require("tidyr")

require("purrr")

require("reshape2")

require("devtools")

require("gclus")

require("ModelMetrics")

require("corrplot")

library("ROCR")



popularity <- read.csv("OnlineNewsPopularity.csv")

set.seed(100)


#Each column has correct data type

factorCols <- c("data_channel_is_lifestyle", "data_channel_is_bus", "data_channel_is_socmed",

            "data_channel_is_tech","data_channel_is_world",
"weekday_is_monday","weekday_is_tuesday",
"weekday_is_wednesday","weekday_is_thursday","weekday_is_friday","weekday_is_saturday",
        "weekday_is_sunday","is_weekend","data_channel_is_entertainment")
popularity[,factorCols] <- lapply(popularity[,factorCols], factor)



#Split data in 70% train set and 30% test set
popularity$url <- sample.split(popularity$url,0.7)
train <- subset(popularity, popularity$url == TRUE)
test <- subset(popularity, popularity$url == FALSE)
train <- train[,-c(1,2)]
test <- test[,-c(1,2)]



#Feature Scaling and Standardization
scaleCol <- c(1,2,6,7,8,10,11,18,19,20,21,22,23,24,25,26,27,28,29,59)
factorScale <-
c(1,2,6,7,8,10,11,18,19,20,21,22,23,24,25,26,27,28,29,59,12,13,14,15,16,17,30,31,32,33,34,35,36,37)
trainscale <- train
testscale <- test
trainscale[,scaleCol] <- apply(train[,scaleCol], MARGIN = 2, FUN = function(X) (X -
min(X))/diff(range(X)))
trainscale[,-factorScale] <- apply(train[,-factorScale], MARGIN = 2, FUN = function(X)(X-mean(X))/sd(X))
testscale[,scaleCol] <- apply(test[,scaleCol], MARGIN = 2, FUN = function(X) (X - min(X))/diff(range(X)))
testscale[,-factorScale] <- apply(test[,-factorScale], MARGIN = 2, FUN = function(X)(X-mean(X))/sd(X))
```

```
#Independent variables and dependent variables

x.train <- trainscale[,-59]

x.test <- testscale[,-59]

y.train <- trainscale[,59]

y.test <- testscale[,59]


#===================================

###Task1.Build linear regression

variable.correlation <- trainscale[,-c(12,13,14,15,16,17,30,31,32,33,34,35,36,37)]

variable.correlation <- cor(variable.correlation)

corrplot(variable.correlation, method="color", type="lower")

linearMod <- lm(shares ~ ., data=trainscale)



#Use MSE to evaluate the linear regression model

mse.lmtrain <- mse(y.train,linearMod$fitted.values)

y.test.pred <- predict(linearMod, x.test)

mse.lmtest <- mse(y.test,y.test.pred)

lm.mse <- as.data.frame(cbind(mse.lmtrain,mse.lmtest))



###Task2.Logistic regression

##Plot distribution of Share

ggplot(data=trainscale, aes(trainscale$shares)) +

  geom_histogram(breaks=seq(0, 0.06, by= 0.001),

                 fill="red",

                 alpha = .8) +
  labs(title="Histogram for Shares", x="Shares", y="Count")



#Generate class 1 and class 0 for shares
##1 for large number of shares, 0 for small number of shares
trainLogit <- trainscale
testLogit <- testscale
trainLogit$shares <- ifelse(trainscale$shares>median(trainLogit$shares),1,0)
testLogit$shares <- ifelse(testscale$shares>median(testLogit$shares),1,0)
y.trainlogit <- trainLogit[,59]
y.testlogit <- testLogit[,59]

#=============================
#Build logistic regression
glm.model <- glm(shares ~.,family=binomial(link='logit'),data = trainLogit)
summary(glm.model)
y.trainlogit.pred <- glm.model$fitted.values
y.trainlogit.pred <- ifelse(y.trainlogit.pred > 0.5,1,0)
train.misClasificError <- mean(y.trainlogit.pred != y.trainlogit)
logit.train.accuracy <- (1-train.misClasificError)*100
y.testlogit.predict <- predict(glm.model,x.test, type = 'response')
y.testlogit.pred <- ifelse(y.testlogit.predict > 0.5,1,0)
test.misClasificError <- mean(y.testlogit.pred != y.testlogit)
logit.test.accuracy <- (1-test.misClasificError)*100
glm.accuray <- as.data.frame(cbind(logit.train.accuracy,logit.test.accuracy))

#ROC Plot
```

```
pr <- prediction(y.testlogit.pred, y.testlogit)

prf <- performance(pr, measure = "tpr", x.measure = "fpr")

plot(prf)


auc <- performance(pr, measure = "auc")

auc <- auc@y.values[[1]]

auc



#========================

##Experiements

#Define Gradient Descent Algorithmn

# define the gradient function dJ/dtheata: 1/m * (h(x)-y))*x where h(x) = x*theta

# in matrix form this is as follows:



# initialize coefficients

num.features <- ncol(x.train)

theta <- matrix(rep(0, num.features+1), nrow=1)



# add a column of 1's for the intercept coefficient


X<- data.matrix(cbind(1, x.train))

y <- y.train


# learning rate and iteration limit
```

```
num_iters <- 1000

alpha.values <- c( 0.001,0.003,0.01,0.03,0.1,0.3)


# gradient descent

# keep history

cost_history <- double(num_iters)

theta_history <- list(num_iters)

cost.history <- double(length(alpha.values))

theta.history <- list(length(alpha.values))


#Experiment with various learning rate

for (j in 1:length(alpha.values)) {


  alpha = alpha.values[j]


  # squared error cost function

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

    sum( (x %*% t(theta) - y)^2 ) / (2*length(y))

  }


  #Gradient Descent

  for (i in 1:num_iters) {

    error <- (X %*% t(theta) - y)

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

    theta <- theta - alpha * t(delta)

    cost_history[i] <- cost(X, y, theta)

    theta_history[[i]] <- theta
  }
  cost.history <- cbind(cost.history,cost_history)
  theta.history[[j]] <- theta_history
}



#Find the iteration for best theta value with each alpha values
cost.history <- as.data.frame(cost.history)
MSE.Train.location <- double(length(alpha.values))

MSE.Location.Matrix <- for(i in 2:(ncol(cost.history))){
  MSE.Train.location[i] <- which.min(cost.history[,i])
}



#Apply model on the test set
MES.Len<- length(MSE.Train.location)
MSE.Train.location <- as.data.frame(MSE.Train.location[2:MES.Len])
MSE.Test.Matrix <- double(length(alpha.values))

MSE.Test <- function(test, test.pred) {
  for (i in 1:(nrow(MSE.Train.location))) {
    t <- MSE.Train.location[i,1]
    Traintheta <- theta.history[[i]][[t]]
    test <- data.matrix(test)
    MSE.Test.Matrix[i] <- sum( (test %*% t(Traintheta) - test.pred)^2 ) / (2*length(test.pred))
  }
  return(MSE.Test.Matrix)
}

X.test <- data.matrix(cbind(1, x.test))
MSE.Test.output1 <- MSE.Test(X.test,y.test)
best.Alpha1 <- alpha.values[which.min(MSE.Test.output1)]
alpha.MSE<- as.data.frame(cbind(alpha.values,MSE.Test.output1))
alpha.values.no.six <- alpha.values[-6]
MSE.Test.output1.no.six <- MSE.Test.output1[-6]
MSE.Test.whole <- min(MSE.Test.output1)
dev.off()
qplot(alpha.values.no.six,MSE.Test.output1.no.six,geom=c("line"), xlab="Alpha Values", ylab="Test
MSE")

#Plot the cost against number of iterations
cost.history <- as.data.frame(cost.history[,-1])
colnames(cost.history) <- c("cost1","cost2","cost3","cost4")
qplot(seq(1,nrow(cost.history),1), cost.history[,1], geom=c("line"), xlab="iteration", ylab="cost")
qplot(seq(1,nrow(cost.history),1), cost.history[,2], geom=c("line"), xlab="iteration", ylab="cost")
qplot(seq(1,nrow(cost.history),1), cost.history[,3], geom=c("line"), xlab="iteration", ylab="cost")
qplot(seq(1,nrow(cost.history),1), cost.history[,4], geom=c("line"), xlab="iteration", ylab="cost")
qplot(seq(1,nrow(cost.history),1), cost.history[,5], geom=c("line"), xlab="iteration", ylab="cost")

#=============================================================
#Randomly choose 10 columns
```

```
ranCols <- sample(ncol(train[,-59]),10)

x.random  <- trainscale[,ranCols]

x.test.random <- testscale[,ranCols]



# initialize coefficients

num.features <- ncol(x.random)

theta <- matrix(rep(0, num.features+1), nrow=1)



# add a column of 1's for the intercept coefficient

X.random <- data.matrix(cbind(1,x.random))

X <- X.random


alpha.values <- 0.1

num_iters <- 1000


# gradient descent to retrain the model

# keep history

cost_history.random <- double(num_iters)

theta_history.random <- list(num_iters)


for (i in 1:num_iters) {

  error <- (X %*% t(theta) - y)

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

  theta <- theta - alpha * t(delta)

  cost_history.random[i] <- cost(X, y, theta)
```

```
    theta_history.random[[i]] <- theta
  }

#Apply model on the test set
X.test.random <- data.matrix(cbind(1, x.test.random))
best.iteration <- which.min(cost_history.random)
MSE.Test.random <- sum((X.test.random %*% t(theta_history.random[[1000]]) - y.test)^2 ) /
(2*length(x.test.random))



#Plot the cost against number of iterations
cost_random<- data.matrix(cost_history.random)
qplot(seq(1,nrow(cost_random),1), cost_random, geom=c("line"), xlab="iteration", ylab="cost")



#=============================================================
#Pick 10 columns
pickCols <- c("num_imgs", "weekday_is_monday", "is_weekend", "abs_title_subjectivity",
         "abs_title_sentiment_polarity", "global_sentiment_polarity", "global_subjectivity",
         "max_positive_polarity", "max_negative_polarity", "average_token_length")

x.pick<- trainscale[,pickCols]
x.test.pick <- testscale[,pickCols]



# initialize coefficients
num.features <- ncol(x.random)
```

```
theta <- matrix(rep(0, num.features+1), nrow=1)



# add a column of 1's for the intercept coefficient
X.pick <- data.matrix(cbind(1,x.pick))
X <- X.pick

# gradient descent to retrain the model
# keep history
cost_history.pick <- double(num_iters)
theta_history.pick <- list(num_iters)

for (i in 1:num_iters) {
  error <- (X %*% t(theta) - y)
  delta <- t(X) %*% error / length(y)
  theta <- theta - alpha * t(delta)
  cost_history.pick[i] <- cost(X, y, theta)
  theta_history.pick[[i]] <- theta
}

#Apply model on the test set
X.test.pick <- data.matrix(cbind(1, x.test.pick))
best.iteration <- which.min(cost_history.pick)
MSE.Test.pick <- sum((X.test.pick %*% t(theta_history.pick[[1000]]) - y.test)^2 ) /
(2*length(x.test.pick))
```

```
#Plot the cost against number of iterations
cost_pick<- data.matrix(cost_history.pick)
qplot(seq(1,nrow(cost_pick),1), cost_pick, geom=c("line"), xlab="iteration", ylab="cost")


#MSE Summary for Linear Regression
MSE<- rbind(MSE.Test.whole,MSE.Test.random,MSE.Test.pick)
test <- c("MSE for All", "MSE for Random 10","MSE for Pick 10")
combine.result.mse <- as.data.frame(cbind(test,MSE))
colnames(combine.result.mse) <- c("test","MSE")

ggplot() + geom_bar(aes(y = MSE, x = test),
                    data = combine.result.mse, stat="identity")


#Logistic Regression
#=================================================================

# Implement Sigmoid function
sigmoid <- function(z) {
  g <- 1/(1+exp(-z))
  return(g)
}

#Cost function for logistic regression
cost.glm <- function(x, y, theta1){
  m = nrow(x)
  hx = sigmoid(x %*% t(theta1))

  return (1/m) * (((-t(y) %*% log(hx)) - t(1-y) %*% log(1 - hx)))

}



# Gradient descent function

gradLog <- function(x, y, theta1) {

  gradient <- (1 / nrow(y)) * (t(x) %*% (1/(1 + exp(-x %*% t(theta1))) - y))

  return(t(gradient))

}



gradientLog.descent <- function(x, y, threshold,alpha=0.001, num.iterations=1000,output.path=FALSE)
{


  # Add x_0 = 1 as the first column

  m <- nrow(x)

  x <-  data.matrix(cbind(rep(1, m), x))


  num.features <- ncol(x)


  # Initialize the parameters

  theta1 <- matrix(runif(num.features), nrow=1)


  # Look at the values over each iteration

  theta.path <- theta1

  cost_history1 <- list()
  for (i in 1:num.iterations) {

    theta <- theta1 - alpha * gradLog(x, y, theta1)

    cost_history1[i] <- cost.glm(x, y, theta1)


    #print(cost.glm(x,y,theta))

    if(all(is.na(theta))) break

    theta.path <- rbind(theta.path, theta1)

    if(i > 2) if(all(abs(theta1 - theta.path[i-1,]) < threshold)) break

  }


  if(output.path) return(theta.path) else return(theta.path[nrow(theta.path),])

}


threhold <- c(1e-7,1e-3)


#test 1

y.trainlogit <- data.matrix(y.trainlogit)

theta.logit1 <- gradientLog.descent(x= x.train, y = y.trainlogit, alpha = 0.2,num.iterations=3000,
threshold = 1e-7,output.path=FALSE)

theta.logit.test1 <- data.matrix(theta.logit1)

X.test <- data.matrix(cbind(1, x.test))

y.testlogit.pred <-  sigmoid(X.test %*% theta.logit.test1)

table.threshold <- confusionMatrix(y.trainlogit,y.testlogit.pred,cutoff = 0.5)

accuracy.logit1 <- round(sum(table.threshold[1,1],table.threshold[2,2])/sum(table.threshold)*100,2)



#test 2
```

```
theta.logit2 <- gradientLog.descent(x=x.train, y = y.trainlogit, alpha = 0.2,num.iterations=3000,
threshold = 1e-3,output.path=FALSE)

theta.logit2 <- data.matrix(theta.logit2)

Xtest <- data.matrix(cbind(1, x.test))

y.testlogit.pred1 <-  sigmoid(Xtest %*% theta.logit2)

table.threshold <- confusionMatrix(y.trainlogit,y.testlogit.pred1,cutoff = 0.5)

accuracy.logit2 <- round(sum(table.threshold[1,1],table.threshold[2,2])/sum(table.threshold)*100,2)


tablecool <- rbind(accuracy.logit1,accuracy.logit2)

Testcool <- c("Threshold 1", "Threshold 2")

tablecool <- cbind(Testcool,tablecool)

colnames(tablecool) <- c("test","accuracy")

#===========================================================

#Randomly choose 10 columns


ranCols <- sample(ncol(train[,-59]),10)

x.random  <- trainscale[,ranCols]

x.test.random <- testscale[,ranCols]




theta.logit.random <- gradientLog.descent(x=x.random, y = y.trainlogit, alpha =
0.2,num.iterations=3000, threshold = 1e-7,output.path=FALSE)

theta.logit.random <- data.matrix(theta.logit.random)

Xtest <- data.matrix(cbind(1, x.test.random))

y.test.logit.pred2 <-  sigmoid(Xtest %*% theta.logit.random)

table.threshold1 <- confusionMatrix(y.trainlogit,y.test.logit.pred2,cutoff = 0.5)

accuracy2.1 <- round(sum(table.threshold1[1,1],table.threshold1[2,2])/sum(table.threshold1)*100,2)
```

```
#===========================================================
#Pick 10 columns


pickCols <- c("num_imgs", "weekday_is_monday", "is_weekend", "abs_title_subjectivity",
           "abs_title_sentiment_polarity", "global_sentiment_polarity", "global_subjectivity",
           "max_positive_polarity", "max_negative_polarity", "average_token_length")


x.pick<- trainscale[,pickCols]
x.test.pick <- testscale[,pickCols]


theta.logit.pick <- gradientLog.descent(x=x.random, y = y.trainlogit, alpha = 0.2,num.iterations=3000,
threshold = 1e-7,output.path=FALSE)
theta.logit.pick <- data.matrix(theta.logit.pick)
Xtest <- data.matrix(cbind(1, x.test.random))
y.test.logit.pred3 <-  sigmoid(Xtest %*% theta.logit.pick)
table.threshold2 <- confusionMatrix(y.trainlogit,y.test.logit.pred3,cutoff = 0.5)
accuracy3.1 <- round(sum(table.threshold2[1,1],table.threshold2[2,2])/sum(table.threshold2)*100,2)



#Accuracy Summary
logit.accuracy.whole <- as.data.frame(rbind(accuracy.logit2,accuracy2.1,accuracy3.1))
logist.test <- c("Whole","Random","Pick")
accuracy.logit.table <- as.data.frame(cbind(logist.test,logit.accuracy.whole))
ggplot() + geom_bar(aes(y = logit.accuracy.whole, x = logist.test),
               data = accuracy.logit.table, stat="identity")
```