HW2.R

neeraj

Mon Feb 8 22:55:22 2016

##Neeraj Asthana (nasthan2)  
##CS 498 HW2 Problem 2.5  
  
options(warn=-1)  
  
#environment  
setwd('/home/neeraj/Documents/UIUC/CS 498/CS498MachineLearning/HW2')  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

## Note: the specification for S3 class "family" in package 'MatrixModels' seems equivalent to one from package 'lme4': not turning on duplicate class definitions for this class.

#parameters  
lambdas <- c(.001, .01, .1, 1)  
epochs <- 50  
steps <- 300  
percent\_training <- .8  
percent\_validation <- .1  
percent\_test <- .1  
num\_examples\_epoch\_test <- 50  
steps\_til\_eval <- 30  
steplength\_a <- .01  
steplength\_b <- 50  
  
#read files and create data set  
raw\_train\_data <- read.csv('adult.data', header=FALSE, na.strings = "?")  
raw\_test\_data <- read.csv('adult.test', header=FALSE, na.strings = "?")  
raw\_data <- rbind(raw\_train\_data, raw\_test\_data, make.row.names=FALSE)  
  
#Split continuous and label data  
#continuous variables: age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week  
x\_vector <- raw\_data[,c(1,3,5,11,12,13)]  
y\_labels <- raw\_data[,15]  
  
#add scaling  
for (i in 1:6){  
 x\_vector[i] <- scale(as.numeric(as.matrix(x\_vector[i])))  
}  
  
#Mentions of positive and negative examples for reference  
neg\_example <- y\_labels[1]  
neg\_example2 <- y\_labels[48842]  
pos\_example <- y\_labels[8]  
pos\_example2 <- y\_labels[48843]  
  
#split data into training, test, and validation sets  
datasplit <- createDataPartition(y=y\_labels, p=.8, list=FALSE)  
trainx <- x\_vector[datasplit,]  
trainy <- y\_labels[datasplit]  
otherx <- x\_vector[-datasplit,]  
othery <- y\_labels[-datasplit]  
datasplit2 <- createDataPartition(y=othery, p=.5, list=FALSE)  
testx <- otherx[datasplit2,]  
testy <- othery[datasplit2]  
valx <- otherx[-datasplit2,]  
valy <- othery[-datasplit2]  
  
#SVM Training  
#loss function for SVM:  
hinge\_loss <- function(predicted, actual){  
 return (max(0, 1 - (predicted \* actual) ))  
}  
  
#evaluation for specific example x (6 items in vector) with parameters a and b  
evaluate <- function(x, a, b){  
 new\_x <- as.numeric(as.matrix(x))  
 return (t(a) %\*% new\_x + b)   
}  
  
#Change y in dataset from <=50k and >50k to -1 and 1  
converty <- function(y){  
 if(y == neg\_example | y == neg\_example2){  
 return (-1)  
 }  
 else if(y == pos\_example | y == pos\_example2){  
 return (1)  
 }  
 else{  
 return(NA)  
 }  
}  
  
convertpred <- function(val){  
 if(val >= 0){  
 return(1)  
 }  
 else{  
 return(-1)  
 }  
}  
  
accuracy <- function(x,y,a,b){  
 correct <- 0  
 wrong <- 0  
 for (i in 1:length(y)){  
 pred <- evaluate(x[i,], a, b)  
 pred <- convertpred(pred)  
 actual <- converty(y[i])  
   
 if(pred == actual){  
 correct <- correct + 1   
 } else{  
 wrong <- wrong + 1  
 }  
 }  
 return(c( (correct/(correct+wrong)), correct, wrong) )  
}  
  
#array of validation accuracies  
val\_accuracies = c()  
test\_accuracies = c()  
  
for (lambda in lambdas){  
 #random initialization  
 #a <- runif(dim(x\_vector)[2], min=-.0001, max=.0001)  
 #b <- runif(1, min=0, max=.01)  
 a <- c(0,0,0,0,0,0)  
 b <- 0  
   
 accuracies <- c()  
 posup <- 0  
 negup <- 0  
   
 for (epoch in 1:epochs){  
   
 #set out 50 examples for testing after every 30 steps  
 ran\_vals <- sample(1:dim(trainx)[1], 50)  
 accuracy\_data <- trainx[ran\_vals, ]  
 accuracy\_labels <- trainy[ran\_vals]  
 train\_data <- trainx[-ran\_vals,]  
 train\_labels <- trainy[-ran\_vals]  
   
 #Keep track of the number of steps taken at each epoch for debugging purposes  
 num\_steps <- 0  
   
 for (step in 1:steps){  
   
 if(num\_steps %% steps\_til\_eval == 0){  
 calc <- accuracy(accuracy\_data, accuracy\_labels, a, b)  
 accuracies <- c(accuracies, calc[1])   
 #print(calc[1])  
 }  
   
 k <- sample(1:length(train\_labels), 1)  
 while(is.na( converty( train\_labels[k] ) )){  
 k <- sample(1:length(train\_labels), 1)  
 }  
 xex <- as.numeric(as.matrix( train\_data[k,] ))  
 yex <- converty( train\_labels[k] )  
   
 pred <- evaluate(xex, a, b)  
 steplength = 1 / ((steplength\_a \* epoch) + steplength\_b)  
   
   
 #gradient vectors  
 if(yex \* pred >= 1){  
 p1 <- lambda \* a  
 p2 <- 0  
 posup <- posup + 1  
 } else {  
 p1 <- (lambda \* a) - (yex \* xex)  
 p2 <- -(yex)  
 negup <- negup + 1  
 }  
   
 #update values for a and b by gradient descent  
 a <- a - (steplength \* p1)  
 b <- b - (steplength \* p2)  
   
 #update steps count  
 num\_steps <- num\_steps + 1  
 }  
 }  
   
 valeval <- accuracy (valx, valy, a, b)  
 val\_accuracies <- c(val\_accuracies, valeval[1])  
   
 testeval <- accuracy(testx, testy, a, b)  
 test\_accuracies <- c(test\_accuracies, testeval[1])  
   
 jpeg(file=paste(toString(lambda),".jpg") )  
 title <- paste("Lambda = ", toString(lambda), " Accuracies Graph")  
 plot(1:length(accuracies) , accuracies, type="o", col="blue", xlab ="Sample Time", ylab ="Accuracy", main = title)  
 dev.off()  
}  
  
##Problem 2.5a  
#A plot of the accuracy every 30 steps, for each value of the regularization constant  
#plotted from within the SVM function  
  
##Problem 2.5b  
#Your estimate of the best value of the regularization constant, together with a brief description of why you believe that is a good value.  
max\_index <- 1  
for(i in 1:length(val\_accuracies)){  
 if (val\_accuracies[i] >= val\_accuracies[max\_index]){  
 max\_index <- i  
 }  
}  
max\_lambda <- lambdas[max\_index]  
max\_lambda

## [1] 0.001

##Problem 2.5c  
#Your estimate of the accuracy of the best classifier on held out data  
test\_accuracies[max\_index]

## [1] 0.8108495