hw32

Qiuying Li UNI ql2280

3/28/2017

setwd("~/Desktop/2017 spring/GR 5241/HW/HW4")  
library("freestats")  
# Part 1  
#step1  
#We need to calculate data points which are currently classified correctly are weighted down  
#First of all, we are giving the weights = 1/n  
#yi = {−1, +1} are class labels.  
#stump ignores all entries of x except xj, it is equivalent to a linear classifier defined by an affine hyperplane.   
#The plane is orthogonal to the jth axis, with which it intersects at xj = theta.   
#The orientation of the hyperplane is determined by m = {−1, +1}.   
  
#In order to have the parameters j, m and θ, we need a desicionStump function  
train\_parameters = function (X,w,Y){  
 parameters = decisionStump(X,w,Y)  
 j = parameters$j  
 theta = parameters$theta  
 m = parameters$m  
 return(j, theta, m)  
}  
  
# We want to generate a decisionStump funtion between -1 and 1  
# w is the weight which is 1/n  
# The goal of this function is to calculate optimal theta, m and j in all dimensions.  
training\_data = function (X,w,Y){  
 a = nrow(X)  
 b = ncol(X)  
 c\_min= rep(a,b)  
 theta\_min = rep(-2,b)  
 m\_min = rep(-1,p)  
 #In order to have the optimal theta, we need to build cost functions for -1 and 1  
 #when yi∈ { 1}  
 cost\_1 = function (theta,x,y,number,weights){  
 classifier = red(-1, number)  
 classifier[x > theta] = 1  
 result\_sum = sum(weights\*classify != y)  
 return(result\_sum)  
 }  
   
 #when yi∈ { 1}  
 cost\_neg1 = function (theta,x,y,number,weights){  
 classifier = red(1, number)  
 classifier[x < theta] = -1  
 result\_sum = sum(weights\*classify != y)  
 return(result\_sum)  
 }  
   
 # The more general case  
 classified = function (X,pars){  
 j = pars[1] #The plane is orthogonal to the jth axis  
 theta = pars[2]  
 level = pars[3]  
 n = nrow(X)  
 x\_class = X[,j]  
 classifier = rep(-m,n)  
 classifier[x\_class > theta] = m  
 result = matrix(classifier)  
 return(result)  
 }  
 # compute optimal theta for each dimension  
 for (dim in seq(b)) {  
 X\_dim <- X[,dim]  
 unique\_X\_dim <- unique(X\_dim)  
 unique\_X\_dim <- c(unique\_X\_dim, -2)  
 # costs for both m = 1 and -1  
 cost\_m\_1 <- apply(matrix(unique\_X\_dim), 1, cost\_1, x=X\_dim, y=Y, number=n, weights=w)  
 cost\_m\_neg1 <- apply(matrix(unique\_X\_dim), 1, cost\_neg1, x=X\_dim, y=Y, number=n, weights=w)  
   
 if (min(cost\_m\_neg1) < min(cost\_m\_1)) {  
 ind <- which.min(cost\_m\_neg1)  
 m\_min[dim] <- -1  
 c\_min[dim] <- c\_d\_n[ind]  
 } else {  
 ind <- which.min(cost\_m\_1)  
 c\_min[d] <- cost\_m\_1[ind]  
 }  
 theta\_min[d] <- unique\_X\_dim[ind]  
 }  
   
 # find out the dimision with the optimal theta and m   
 min\_dim <- which.min(c\_min)  
 theta\_min <- theta\_min[min\_d]  
 c\_min <- c\_min[min\_d]  
 m\_min <- m\_min[min\_d]  
 return(c(min\_dim, theta\_min, m\_min))  
}  
  
#Step 2  
#label <- classify(X, pars) for the classification routine,  
#which evaluates the weak learner on X using the parametrization pars.  
classify <- function(X, pars) {  
 # classify X use the parameters in pars  
 # pars is the triplet (j, theta, m)  
 j <- pars[1]  
 theta <- pars[2]  
 m <- pars[3]  
 n <- nrow(X)  
 X\_d <- X[,j]  
 classified <- rep(-m, n)  
 classified[X\_d > theta] = m  
 return(matrix(classified))  
}  
  
#step 3  
#A function c hat <- agg class(X, alpha, all\_pars)   
# It evaluates the boosting classifier (“ag- gregated classifier”) on X.   
#The argument alpha denotes the vector of voting weights and all\_pars contains the parameters of all weak learners.  
  
agg\_class <- function(X, alpha, all\_pars) {  
 a<- nrow(X)  
 b <- length(alpha)  
 agg\_labels <- matrix(0, a, 1)  
   
 # we need to be careful when there is only one row  
 # For this case, the function is no longer appliable.  
 if (b == 1) {  
 all\_pars <- rbind(all\_pars, matrix(0,1,3))  
 }  
   
 # otherwise, when we have multiple of rows  
 for (i in seq(b)) {  
 al <- alpha[i]  
 agg\_labels <- agg\_labels + al \* classify(X, all\_pars[i,])   
 }  
 classified <- matrix(-1, a, 1)  
 classified[agg\_labels >= 0] <- 1  
 return(classified)  
}  
  
# Part 2  
# Implement the functions train and classify for decision stumps.  
comb\_fun <- function(X, Y, all\_pars, alpha, iter) {  
 # In order to perform a cross validation, we choose k = 5  
 k = 5  
 fold\_size <- n / k  
 cv\_errors <- matrix(1,k,1)  
 for (cv in seq(k)) {  
 cv\_j <- X[-(((cv-1)\*fold\_size+1):(cv\*fold\_size)),]  
 cv\_m <- Y[-(((cv-1)\*fold\_size+1):(cv\*fold\_size)),]  
 cv\_theta <- w[-(((cv-1)\*fold\_size+1):(cv\*fold\_size)),]  
 cv\_set\_j <- X[((cv-1)\*fold\_size+1):(cv\*fold\_size),]  
 cv\_set\_m <- Y[((cv-1)\*fold\_size+1):(cv\*fold\_size),]  
  
 # By plog in the 3 parameters,m,j theta in train function and classify function  
 cv\_pars <- train(cv\_j, cv\_theta, cv\_m)   
 tr\_pred\_labels <- classify(cv\_j, cv\_pars)   
   
 # calculate the prediction error rate  
 tr\_error\_rate <- sum(cv\_theta\*(tr\_pred\_labels != tr\_set\_labels)) / sum(cv\_theta)  
   
 # Compute voting weights alpha by given formula  
 cv\_alpha <- log((1-tr\_error\_rate)/tr\_error\_rate)  
 cv\_labels <- agg\_class(cv\_set\_feats, c(alpha[0:(iter-1),], cv\_alpha), rbind(all\_pars[0:(iter-1),], cv\_pars))  
   
 # compute cross validation error rate  
 cv\_error <- sum(cv\_labels != cv\_set\_labels) / fold\_size  
 cv\_errors[cv,] <- cv\_error  
 }  
 cv\_avg\_error <- mean(cv\_errors)  
 return(cv\_avg\_error)  
}  
  
# part 3.  
#step 1 read USPS data  
train3 <- as.matrix(read.table("train\_3.txt", header=FALSE, sep=","))  
train8 <- as.matrix(read.table("train\_8.txt", header=FALSE, sep=","))  
xtrain <- rbind(train3, train8)   
ytrain3 <- rep(c(1,-1), c(nrow(train3), nrow(train8)))   
ytrain3 <- matrix(ytrain3)  
test <- as.matrix(read.table("zip\_test.txt",header = F))  
test <- test[test[,1]%in%c(3,8),]   
xtest <- test[,-1]  
ytest <- test[,1]  
ytest[ytest == 3] <- 1  
ytest[ytest == 8] <- -1  
ytest <- matrix(ytest)  
  
#step 2: apply AdaBoost function to USPS data  
# To calculate the test error, training error and cross validation error  
AdaBoost <- function(B, X, Y, testX, testY){  
 n <- nrow(X)  
 test\_size <- nrow(testX)  
 w <- matrix(1/n, n)   
 alpha <- matrix(0, B, 1)  
 all\_pars <- matrix(0, B, 3)  
 error <- matrix(0, B, 3)   
 iterator <- 0  
 # we need to perform a 5-cross vlaidation   
 while (iterator < B) {  
 iterator = iterator + 1  
 cv\_error <- ff\_cv(X, Y, all\_pars, alpha, iterator)  
 # calculate the three parameters m,j,theta  
 pars <- train(X, w, Y)   
 # use above parameters to perform classify function  
 labels <- classify(X, pars)  
 # calculate error of the training data set  
 error\_rate <- sum(w\*(labels != Y)) / sum(w)  
 # compute voting weights alpha  
 alpha <- log((1-error\_rate)/error\_rate)  
 # plug the calculated parameters and voting weights  
 alpha[iterator,] <- alpha  
 all\_pars[iterator,] <- pars  
 # recompute weights w  
 w <- w \* exp(alpha \* (labels != Y))   
 # calcualte error of the test data  
 lable\_test <- agg\_class(testX, alpha[1:iterator,], all\_pars[1:iterator,])  
 error\_test <- sum(lables\_test != testY) / test\_size  
 error[iterator,1] <- error\_rate  
 error[iterator,2] <- cv\_error  
 error[iterator,3] <- error\_test  
 print(paste0("iter ", iterator, ": j=", pars[1], ", theta=", pars[2], ", m=", pars[3], ", alpha=", round(alpha, digits=4), ", tr\_err=", round(error\_rate, digits=4), ", cv\_err=", round(cv\_error, digits=4), ", test\_err=", round(test\_error, digits=4)))  
 }  
 return(cbind(alpha, all\_pars,error))  
}  
n <- nrow(xtrain)  
w <- matrix(1/n, n)