Hw4_p3

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 ar
e 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 class
ifier defined by an affine hyperplane.
#The plane is orthogonal to the jth axis, with which it intersects at xj = th
eta.
#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 dim
ensions.
  train = 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)
```

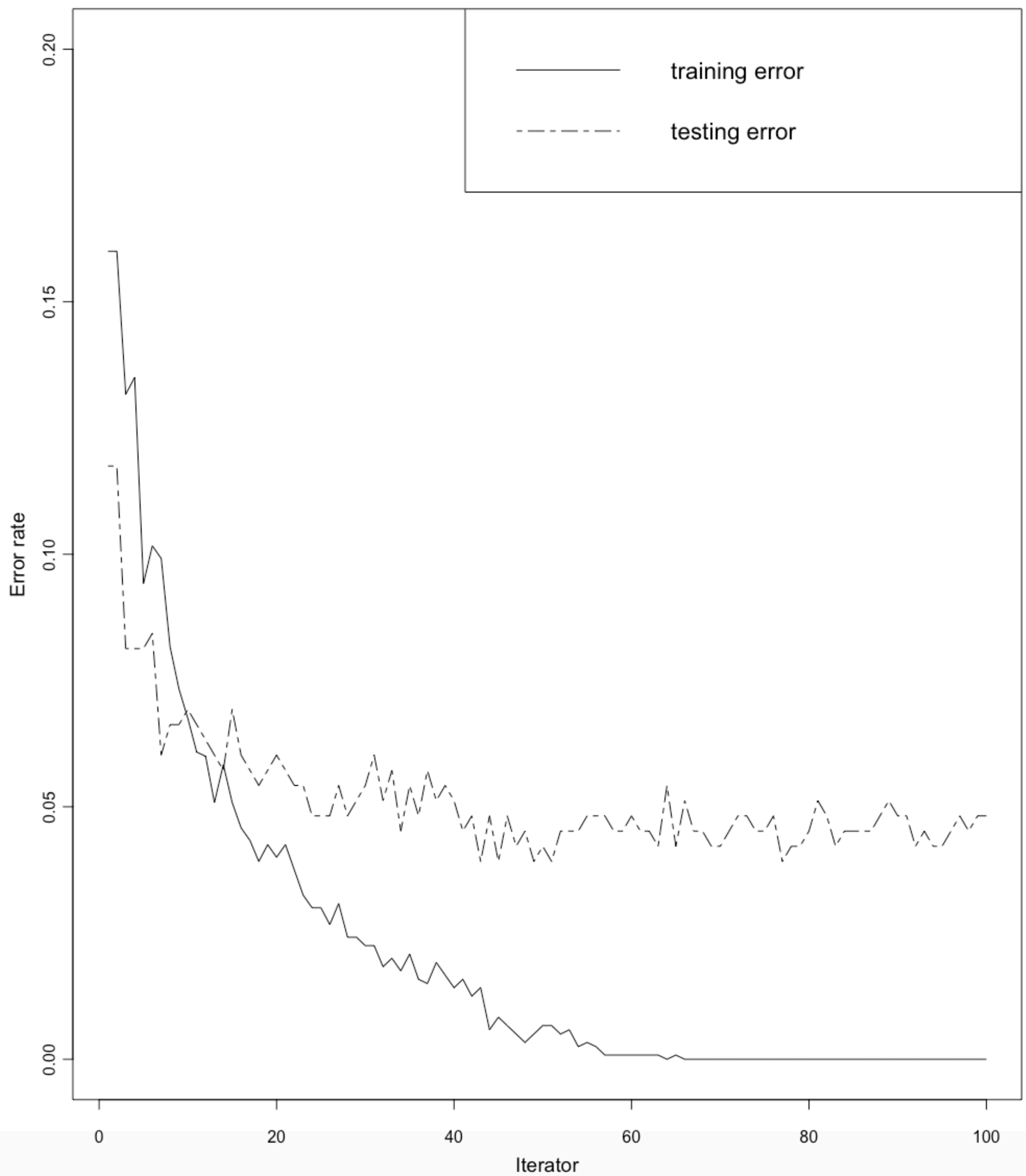
```
  }

  #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)
  }

    # 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, nu
mber=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.
Pars are the parameters of the <j,m,theta>


classify <- function(X, pars) {
  # classify X use the parameters in pars
  j <- pars[1]
```

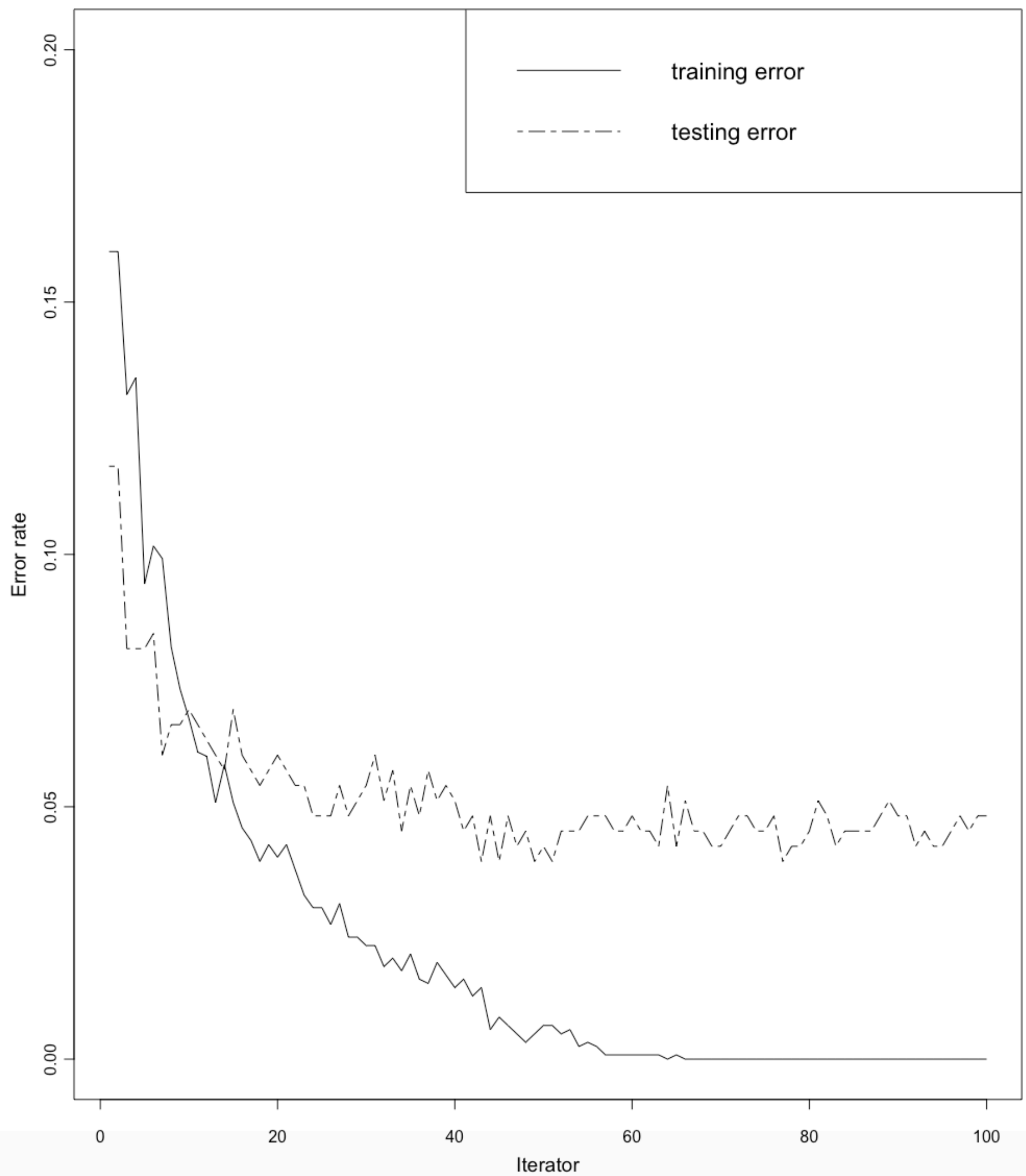
```r
  theta <- pars[2]
  m <- pars[3]
  n <- nrow(X)
  X_d <- X[,j]
  classified <- rep(-m, n)
  classified[X_d > theta] = m
  return(matrix(classified))
}

#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 contain
s 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),]

   # After cross validation, we can plog in the 3 parameters,m,j theta in tra
in 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), rbi
nd(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 through combined train function
and classify function, to find out the three parameters
  while (iterator < B) {
    iterator = iterator + 1
    cv_error <- comb_fun (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
}
  return(cbind(alpha, all_pars,error))
}
n <- nrow(xtrain)
w <- matrix(1/n, n)

result <- AdaBoost(100, xtrain, ytrain3, xtest, ytest)




#part 4: make a plot of the error

test_error = result[,7]

training_error = result[,6]

plot(training_error,type = "l",lty = 1,ylim=c(0, 0.2),xlab = "Iterator", ylab
 = "Error Rate"

lines(test_error, lty = 6)

legend("topright", c("training error” ,“testing error”), lty=c(1, 6),cex=0.7)
```

```
mtext(1,text="Iteration",cex=0.6,line = 1)
```




training error
testing error
Error rate
0.00
0.05
0.10
0.15
0.20
0
20
40
60
80
100
Iterator