hw4

Qiuying Li UNI ql2280

3/28/2017

```r
setwd("~/Desktop/2017 spring/GR 5241/HW/HW4")
library("freestats")

DScost <- function(theta, Xvec, Yvec, number, weights) {
  # calculate the cost of given theta
  # Xvec is only one dimension X data
  classified <- rep(-1, number)
  classified[Xvec > theta] = 1
  c_d <- sum(weights*(classified != Yvec))
  return(c_d)
}
DScostN <- function(theta, Xvec, Yvec, number, weights) {
  # calculate the cost of given theta
  # Xvec is only one dimension X data
  classified <- rep(1, number)
  classified[Xvec > theta] = -1
  c_d <- sum(weights*(classified != Yvec))
  return(c_d)
}
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))
}

train_pkg <- function(X, w, Y) {
  # use the decisionStump function with weights w
  # return [j, theta, m]
  dc.out <- decisionStump(X, w, Y)
  return(c(dc.out$j, dc.out$theta, dc.out$m))
}

train <- function(X, w, Y) {
  # produce a decision stump classifier on X, Y with weights w
  # m can be both 1 and -1
  # return [j, theta, m]
  n <- nrow(X)
  p <- ncol(X)
  min_cs = rep(n, p)
  min_thetas = rep(-2, p)
```

```r
  min_ms = rep(1, p)
  # compute optimal theta for each dimension
  for (d in seq(p)) {
    X_d <- X[,d]
    unique_X_d <- unique(X_d)
    # add -2 to the list to make sure every possible solution is touched
    unique_X_d <- c(unique_X_d, -2)
    # get costs for every possible theta
    # costs for both m = 1 and -1
    c_d <- apply(matrix(unique_X_d), 1, DScost, Xvec=X_d, Yvec=Y, number=n, weights=w)
    c_d_n <- apply(matrix(unique_X_d), 1, DScostN, Xvec=X_d, Yvec=Y, number=n, weights=w)
    # store the minimal costs with the respecting theta for this dimension
    if (min(c_d_n) < min(c_d)) {
      ind <- which.min(c_d_n)
      min_ms[d] <- -1
      min_cs[d] <- c_d_n[ind]
    } else {
      ind <- which.min(c_d)
      min_cs[d] <- c_d[ind]
    }
    min_thetas[d] <- unique_X_d[ind]
  }
  # return the minimal theta with the respecting dimension
  min_d <- which.min(min_cs)
  min_theta <- min_thetas[min_d]
  min_c <- min_cs[min_d]
  min_m <- min_ms[min_d]
  return(c(min_d, min_theta, min_m))
}

agg_class <- function(X, alpha, allPars) {
  # evaluates the boosting classifier on X
  # return the classified result with shape n x 1
  n <- nrow(X)
  B <- length(alpha)
  agg_labels <- matrix(0, n, 1)
  if (B == 1) {
    # deal with the case when there is one row, the matrix indexing is no longer applicable
    allPars <- rbind(allPars, matrix(0,1,3))
  }
  for (i in seq(B)) {
    a <- alpha[i]
    agg_labels <- agg_labels + a * classify(X, allPars[i,])
  }
  classified <- matrix(-1, n, 1)
  classified[agg_labels >= 0] <- 1
  return(classified)
}

ff_cv <- function(X, Y, allPars, alphas, iter) {
  fold_size <- n / 5
  cv_errors <- matrix(1,5,1)
  for (cv in seq(5)) {
```

```r
    tr_set_feats <- X[-(((cv-1)*fold_size+1):(cv*fold_size)),]
    tr_set_labels <- Y[-(((cv-1)*fold_size+1):(cv*fold_size)),]
    tr_w <- w[-(((cv-1)*fold_size+1):(cv*fold_size)),]
    cv_set_feats <- X[((cv-1)*fold_size+1):(cv*fold_size),]
    cv_set_labels <- Y[((cv-1)*fold_size+1):(cv*fold_size),]
    # train weak learners
    cv_pars <- train(tr_set_feats, tr_w, tr_set_labels)     # [j, theta, m]
    tr_pred_labels <- classify(tr_set_feats, cv_pars)
    tr_error_rate <- sum(tr_w*(tr_pred_labels != tr_set_labels)) / sum(tr_w)
    cv_alpha <- log((1-tr_error_rate)/tr_error_rate)
    # using the boosting classifier so far to classify the cv set
    # print(paste("cross validation alpha:", cv_alpha))
    cv_labels <- agg_class(cv_set_feats, c(alphas[0:(iter-1),], cv_alpha), rbind(allPars[0:(iter-1),],
    # compute cv 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)
}

####################################################
train.3 <- as.matrix(read.table("train_3.txt", header=FALSE, sep=","))
train.8 <-  as.matrix(read.table("train_8.txt", header=FALSE, sep=","))
xtrain <- rbind(train.3, train.8)       # 1200 x 256
ytrain.3 <- rep(c(1,-1), c(nrow(train.3), nrow(train.8)))   #1200
ytrain.3 <- matrix(ytrain.3)        # 1200 x 1
test <- as.matrix(read.table("zip_test.txt",header = F))
test <- test[test[,1]%in%c(3,8),]       # 332 x 257
xtest <- test[,-1]
ytest <- test[,1]
ytest[ytest == 3] <- 1
ytest[ytest == 8] <- -1
ytest <- matrix(ytest)

# perform AdaBoost
AdaBoost <- function(B, X, Y, testX, testY){
    # X is a nxp matrix
    # Y is a nx1 matrix
    # return [alphas, pars, tr_errors, cv_errors, test_errors]
    n <- nrow(X)
    test_size <- nrow(testX)
    w <- matrix(1/n, n)     # n x 1
    alphas <- matrix(0, B, 1)
    allPars <- matrix(0, B, 3)
    errors <- matrix(0, B, 3)   # training, cv and test errors
    itercount <- 0
    while (itercount < B) {
        itercount = itercount + 1
        # get 5 fold cross validation error rate
        cv_error <- ff_cv(X, Y, allPars, alphas, itercount)
        # train weak learners
        pars <- train(X, w, Y)      # [j, theta, m]
```

```r
        # use the trained weak learner to classify
        labels <- classify(X, pars)             # n x 1
        # compute training error rate
        error_rate <- sum(w*(labels != Y)) / sum(w)
        # compute voting weights
        alpha <- log((1-error_rate)/error_rate)
        # save the classifier
        alphas[itercount,] <- alpha
        allPars[itercount,] <- pars
        # recompute weights w
        w <- w * exp(alpha * (labels != Y))     # n x 1
        # calcualte test error
        test_labels <- agg_class(testX, alphas[1:itercount,], allPars[1:itercount,])
        test_error <- sum(test_labels != testY) / test_size
        errors[itercount,1] <- error_rate
        errors[itercount,2] <- cv_error
        errors[itercount,3] <- test_error
        print(paste0("iter ", itercount, ": j=", pars[1], ", theta=", pars[2], ", m=", pars[3], ", alpha
    }
    return(cbind(alphas, allPars, errors))
}
n <- nrow(xtrain)
w <- matrix(1/n, n)     # initial weights
#out <- AdaBoost(100, xtrain, ytrain.3, xtest, ytest)
```