STAT GR5241 HW4_Q3_mjs2364

Mathieu Sauterey - UNI: MJS2364

April 2nd, 2018

Problem 3 - Boosting (25 points)

Implementation of the function *train* below for decision stumps:

```r
# Implementation of the "train" function using decision stumps
train <- function(X, w, y){

  # Inner function that calculates the misclassification rate
  misclass.rate <- function(theta, X, w, y){

    y_hat = ifelse(X > theta, 1, -1)
    indic = y != y_hat
    rate = sum(w %*% indic)/sum(w)
    return(rate)
  }

  # Inner function that finds the best theta and its error for a given j
  theta_optim <- function(X, w, y){

    all_theta <- rep(0,201)
    idx <- 0

    for (theta in seq(-1,1,0.01)){
      idx <- idx+1
      all_theta[idx] <- misclass.rate(theta, X, w, y)
    }

    lowest_theta <- seq(-1,1,0.01)[which.min(all_theta)]
    return(c(lowest_theta, min(all_theta)))
  }

  # 256x2 matrix of best theta and its error for all 256 j
  all_j_theta <- matrix(0,256,2)

  for (j in 1:256){
    all_j_theta[j,] <- theta_optim(X = X[,j], w=w, y=y)
  }

  # Returns pars (optimal stump j,theta with lowest misclassification rate)
  pars <- list(j=which.min(all_j_theta[,2]),theta=min(all_j_theta[,1]), m=1)
  return(pars)
}
```

Implementation of the function *classify* below for decision stumps:

```
# Implementation of "classify" function (inputs: training data and optimal stump)
classify <- function(X, pars){

  # Basic decision stump classification
  label <- rep(0,nrow(X))
  label <- ifelse(X[,pars$j] > pars$theta, 1, -1)

  # Returns a vector of predicted digit labels for a given pars (optimal stump)
  return(label)
}
```

Implementation of the *AdaBoost* training algorithm below:

```
# Implementation of AdaBoost algo (inputs: training data, labels, and # learners)
AdaBoost <- function(X_train,Y_train,B){

  # Sets initial weights equally and pre-allocates memory
  w <- rep(1/nrow(X_train), nrow(X_train))
  alpha <- rep(0,B)
  error <- rep(0,B)
  all.stumps <- rep(list(list()),B)

  # For each b, finds best pars, gets error, updates weights w and compute alpha
  for (b in 1:B){

    all.stumps[[b]] <- train(X_train, w, Y_train)
    try.label <- classify(X_train, all.stumps[[b]])
    label_match <- as.numeric(try.label != Y_train)

    error[b] <- sum(w %*% label_match/sum(w))
    alpha[b] <- log((1-error[b])/error[b])
    w <- w * exp(alpha[b]*label_match)
    }

  # Returns list containing a list of all pars(j,theta) and a vector of all alphas
  return(list(allPars=all.stumps,alpha=alpha))
}
```

Implementation of the *agg_class* function which is the aggregated weighted classifier:

```
# Implementation of "agg_class" (input: data to predict, all alpha, all pars)
agg_class <- function(X_pred, alpha, allPars){

  # matrix of prediction for each weak learner (each pred is in a column)
  matrix_agg <- matrix(NA, nrow(X_pred), length(allPars))

  # calculates preds and fills the matrix
  for (b in 1:length(allPars)){
    matrix_agg[,b] <- (classify(X_pred, allPars[[b]]))
  }

  # Calculates final prediction by assigning a weight to each learner's prediction
  c_hat <- sign(matrix_agg %*% alpha)

  # Returns the prediction from the weighted classifier
  return(c_hat)
}
```

NOTICE: The code below takes approximately 1h to run. This code trains the algorithm on handwritten digits from the USPS data set, cross-validates the model, assesses CV training and test error, and plots these errors vs the number of iterations:

```
set.seed(1)

# Stores training data and creates a training dataset and training label set
data_3 <- read.table("train_3.txt", as.is = TRUE, sep=",")
data_8 <- read.table("train_8.txt", as.is = TRUE, sep=",")
data_train <- as.matrix(rbind(data_3, data_8))
y_train <- as.matrix(rep(c(-1,1),c(nrow(data_3),nrow(data_8))))

# Stores test data and creates a test dataset and test label set
data_test <- as.matrix(read.table("zip_test.txt", as.is = TRUE))
data_test <- data_test[data_test[,1] %in% c("3","8"),]
y_test <- as.matrix(data_test[,1])
y_test <- ifelse(y_test == "8", 1, -1)
data_test <- data_test[,-1]


# Assigns number of folds and upper bound for the number of weak learners
k_fold <- 5
B_bound <- 20

# Randomly splits data into 5 folds
idx_k <- sample(rep(1:k_fold, each=nrow(data_train)/k_fold))

# Pre-allocates memory for the training and test error across iterations
error_b_tr <- c()
error_b_te <- c()
```

```r
# Loops on all weak learners
for (b in 1:B_bound){

  # Pre-allocates memory for the training and test error at each fold
  error_k_tr <- c()
  error_k_te <- c()

  # Loops performs cross validation on each fold
  for (k in 1:k_fold){
    allpars_alpha <- AdaBoost(data_train[idx_k != k,], y_train[idx_k != k,], b)
    alpha_k <- allpars_alpha$alpha
    allPars_k <- allpars_alpha$allPars
    error_k_tr[k] <- mean( y_train[idx_k != k,]!= agg_class(data_train[idx_k != k,],
        alpha_k, allPars_k))
    error_k_te[k] <- mean( y_train[idx_k == k,]!= agg_class(data_train[idx_k == k,],
        alpha_k, allPars_k))
  }

  # Calculates 5-fold cross-validated training and test error at each iteration
  error_b_tr[b] <- mean(error_k_tr)
  error_b_te[b] <- mean(error_k_te)
}

# Plots the Cv training and test misclassification error vs number of learners
plot(1:B_bound, error_b_tr, type="l", col="red",
     xlab="Number of weak learners",
     ylab="CV Misclassification error",
     main = "CV Misclassification error vs number of weak learners")
lines(1:B_bound, error_b_te, col="blue")
legend("topright", c("Training","Test"), cex=1, bty="n", fill=c("red","blue"))
```