Main

Haosheng Ai, Yushi Pan, Chuyun Shu, Yuqi Xing, Serena Yuan

In your final repo, there should be an R markdown file that organizes **all computational steps** for evaluating your proposed Facial Expression Recognition framework.

This file is currently a template for running evaluation experiments. You should update it according to your codes but following precisely the same structure.

```
#install the packages needed
packages.used <- c("EBImage","R.matlab","readxl","dplyr","ggplot2","caret","glmnet",
                   "WeightedROC","gbm","AUC","MASS","randomForest","pROC", "e1071","xgboost")

packages.needed <- setdiff(packages.used, intersect(installed.packages()[,1], packages.used))

if(length(packages.needed) > 0){
   install.packages(packages.needed, dependencies = TRUE)
}
if(!require("xgboost")){
  install.packages("xgboost")
}


#load libraries
library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(glmnet)
library(WeightedROC)
library(gbm)
library(MASS)
library(AUC)
library(randomForest)
library(pROC)
library(e1071)
library(xgboost)
```

Step 0 set work directories

```
set.seed(2020)
# setwd("~/Project3-FacialEmotionRecognition/doc")
# here replace it with your own path or manually set it in RStudio to where this rmd file is located.
# use relative path for reproducibility
```

Provide directories for training images. Training images and Training fiducial points will be in different subfolders.

```r
train_dir <- "../data/train_set/" # This will be modified for different data sets.
train_image_dir <- paste(train_dir, "images/", sep="")
train_pt_dir <- paste(train_dir,  "points/", sep="")
train_label_path <- paste(train_dir, "label.csv", sep="")
```

Step 1: set up controls for evaluation experiments.

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) cross-validation on the training set
- (T/F) reweighting the samples for training set
- (number) K, the number of CV folds
- (T/F) process features for training set
- (T/F) run evaluation on an independent test set
- (T/F) process features for test set

```r
balance.data <- FALSE

sample.reweight <- TRUE # run sample reweighting in model training
K <- 5  # number of CV folds
run.feature.train <- TRUE # process features for training set
run.feature.test <- TRUE # process features for test set

#set up controls for baseline(gbm) model
run.cv.gbm <- FALSE # run cross-validation on the training set
run.train.gbm <- TRUE # run evaluation on a training set
run.test.gbm <- TRUE # run evaluation on an independent test set

#set up controls for xgboost model
run.cv.xgb <- FALSE
run.cv.xgb.plot <- TRUE
run.train.xgb <- FALSE
run.test.xgb <- TRUE

#set up controls for lda model
run.cv.pca <- FALSE
run.pca <- FALSE
run.train.lda <- FALSE
run.test.lda <- TRUE

#set up controls for unweighted random forest model
run.cv.randomForest <- FALSE
run.train.randomForest <- FALSE
run.test.randomForest <- TRUE


#set up controls for weighted random forest model
run.cv.randomForestWeight <- FALSE
run.train.randomForestWeight <- FALSE
run.test.randomForestWeight <- TRUE
```

```r
#set up controls for knn model
run.cv.knn <- FALSE
run.train.knn <- FALSE
run.test.knn <- TRUE

#set up controls for elastic net model
run.cv.enet <- FALSE
run.test.enet <- TRUE
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications.

```r
#create hyperparameter grid for gbm model (baseline)
hyper_grid_gbm <- expand.grid(
  shrinkage = c(0.01, 0.05, 0.1),
  n.trees = c(500, 1000, 1500)
)

#create hyperparameter list for xgboost / elastic net
lmbd <- c(1e-3, 5e-3, 1e-2, 5e-2, 1e-1)
model_labels = paste("xgboost with lambda =", lmbd)

#create pc list for lda model
pca.list = c(30, 50, 100, 200, 300, 400, 500, 600)

#create hyperparameter list for random forest model
ntree = c(50, 100, 150, 200, 250)
randomForest_model_labels = paste("Random Forest with number of trees =", ntree)
randomForestWeight_model_labels = paste("randomForestWithWeight with number of trees =", ntree)
```

Step 2: import data and train-test split

```r
#train-test split
info <- read.csv(train_label_path)
n <- nrow(info)
n_train <- round(n*(4/5), 0)
train_idx <- sample(info$Index, n_train, replace = F)
test_idx <- setdiff(info$Index, train_idx)
```

If you choose to extract features from images, such as using Gabor filter, R memory will exhaust all images are read together. The solution is to repeat reading a smaller batch(e.g 100) and process them.

```r
n_files <- length(list.files(train_image_dir))

image_list <- list()
for(i in 1:100){
   image_list[[i]] <- readImage(paste0(train_image_dir, sprintf("%04d", i), ".jpg"))
}
```

Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

```r
#function to read fiducial points
#input: index
#output: matrix of fiducial points corresponding to the index
readMat.matrix <- function(index){
    return(round(readMat(paste0(train_pt_dir, sprintf("%04d", index), ".mat"))[[1]],0))
}

#load fiducial points
fiducial_pt_list <- lapply(1:n_files, readMat.matrix)
save(fiducial_pt_list, file="../output/fiducial_pt_list.RData")
```

Step 3: construct features and responses

- The follow plots show how pairwise distance between fiducial points can work as feature for facial emotion recognition.
  - In the first column, 78 fiducials points of each emotion are marked in order.
  - In the second column distributions of vertical distance between right pupil(1) and right brow peak(21) are shown in histograms. For example, the distance of an angry face tends to be shorter than that of a surprised face.
  - The third column is the distributions of vertical distances between right mouth corner(50) and the midpoint of the upper lip(52). For example, the distance of an happy face tends to be shorter than that of a sad face.

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features and responses that are required by all the models you are going to evaluate later.

- `feature.R`
- Input: list of images or fiducial point
- Output: an RData file that contains extracted features and corresponding responses

```
source("../lib/feature.R")
tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(fiducial_pt_list, train_idx))
  save(dat_train, file="../output/feature_train.RData")
}else{
  load(file="../output/feature_train.RData")
}

tm_feature_test <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature(fiducial_pt_list, test_idx))
  save(dat_test, file="../output/feature_test.RData")
}else{
  load(file="../output/feature_test.RData")
}
```

Baseline Model: Gradient Boosted Trees

Step 4: Train a classification model with training features and responses

Call the train model and test model from library.

```
source("../lib/cross_validation_gbm.R")
source("../lib/train_gbm.R")
source("../lib/test_gbm.R")
```

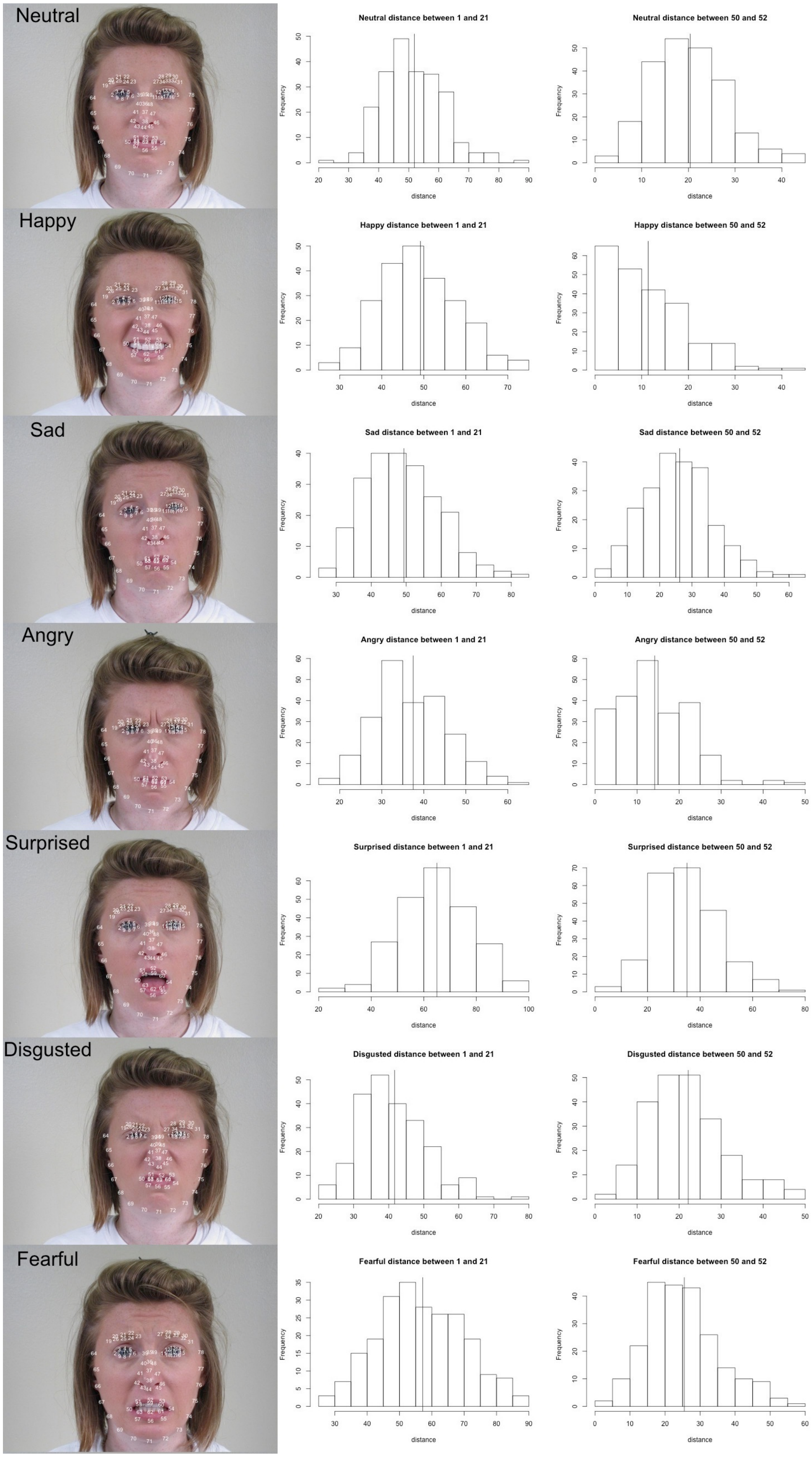


Figure 1: Figure1

Model selection with cross-validation

- Do model selection by performing a manual grid search and choosing among different values of training model parameters. We have constructed out grid of hyperparameter combinations. We're going to search across 18 models with varying shrinkage, n.trees, and interaction.depth, by looping through each hyperparameter combination and performing 5-fold CV.

```
feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)

if(run.cv.gbm){
  res_cv <- matrix(0, nrow = nrow(hyper_grid_gbm), ncol = 4)
  for(i in 1:nrow(hyper_grid_gbm)){
    cat("shrinkage = ", hyper_grid_gbm[i,1], "\n")
    cat("n.trees = ", hyper_grid_gbm[i,2], "\n")
    res_cv[i,] <- cv.function(features = feature_train, labels = label_train, K,
                              shrink = hyper_grid_gbm[i,1],
                              n.trees = hyper_grid_gbm[i,2],
                              depth = 1,
                              reweight = sample.reweight)
  save(res_cv, file="../output/res_cv_gbm.RData")
  }
}else{
  load("../output/res_cv_gbm.RData")
}
```

We output the first five sets of hyperparameters that has the highest mean AUC and the first five sets of hyperparameters that has the lowest mean error. We choose shrinkage = 0.05 and n.trees = 1000 in our final model because it yield the lowest mean error as well as the highest mean AUC.

```
res_cv_gbm <- data.frame(hyper_grid_gbm, res_cv)
colnames(res_cv_gbm) <- c("shrinkage", "n.trees",
                          "mean_error", "sd_error", "mean_AUC", "sd_AUC")
head(res_cv_gbm[order(-res_cv_gbm$mean_AUC),])
```

```
##   shrinkage n.trees mean_error   sd_error  mean_AUC     sd_AUC
## 5      0.05    1000  0.2650530 0.02024766 0.8107619 0.02404243
## 8      0.05    1500  0.2819252 0.01804849 0.8099137 0.02345825
## 9      0.10    1500  0.2977817 0.01358486 0.8066304 0.02041692
## 6      0.10    1000  0.2974643 0.01415036 0.8050799 0.02532261
## 2      0.05     500  0.2799460 0.02288277 0.8009663 0.02480173
## 3      0.10     500  0.2866214 0.01852532 0.7971020 0.01943246
```

```
head(res_cv_gbm[order(res_cv_gbm$mean_error),])
```

```
##   shrinkage n.trees mean_error   sd_error  mean_AUC     sd_AUC
## 5      0.05    1000  0.2650530 0.02024766 0.8107619 0.02404243
## 2      0.05     500  0.2799460 0.02288277 0.8009663 0.02480173
## 8      0.05    1500  0.2819252 0.01804849 0.8099137 0.02345825
## 7      0.01    1500  0.2827393 0.02621163 0.7912025 0.02496446
## 3      0.10     500  0.2866214 0.01852532 0.7971020 0.01943246
## 4      0.01    1000  0.2956279 0.02016686 0.7828075 0.02488460
```

- Choose the "best" parameter value

```
gbm_shrink <- 0.05
gbm_n.trees <- 1000
```

- Train the model with the entire training set using the selected model (model parameter) via cross-validation.

```
# training weights
weight_train <- rep(NA, length(label_train))

for (v in unique(label_train)){
  weight_train[label_train == v] = 0.5 * length(label_train) / length(label_train[label_train == v])
}

if(run.train.gbm) {
  tm_train_gbm <- system.time(fit_train_gbm <- train(feature_train, label_train,
                                                     w = weight_train,
                                                     n.trees = gbm_n.trees,
                                                     shrink = gbm_shrink,
                                                     depth = 1))
  save(fit_train_gbm, tm_train_gbm, file="../output/fit_train_gbm.RData")
} else {
  load(file = "../output/fit_train_gbm.RData")
}
```

Step 5: Run test on test images

```
tm_test_gbm = NA
feature_test <- as.matrix(dat_test[, -6007])

if(run.test.gbm){
  load(file="../output/fit_train_gbm.RData")
  tm_test_gbm <- system.time(
    {prob_pred <- test(fit_train_gbm, feature_test, pred.type = 'response');
                          label_pred <- ifelse(prob_pred > 0.5, 2, 1)})
}
```

```
## Using 1000 trees...
```

- evaluation

```
## reweight the test data to represent a balanced label distribution
label_test <- as.integer(dat_test$label)
weight_test <- rep(NA, length(label_test))

for (v in unique(label_test)){
  weight_test[label_test == v] = 0.5 * length(label_test) / length(label_test[label_test == v])
}

accu <- sum(weight_test * (label_pred == label_test)) / sum(weight_test)
tpr.fpr <- WeightedROC(prob_pred, label_test, weight_test)
auc <- WeightedAUC(tpr.fpr)

cat("The accuracy of the gbm model is ", accu*100, "%.\n")
```

```
## The accuracy of the gbm model is  72.42105 %.
```

```
cat("The AUC of the gbm model is", auc, ".\n")
```

```
## The AUC of the gbm model is 0.8067874 .
```

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

```r
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
```

```
## Time for constructing training features= 1.147 s
```

```r
cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
```

```
## Time for constructing testing features= 0.199 s
```

```r
cat("Time for training model=", tm_train_gbm[1], "s \n")
```

```
## Time for training model= 207.771 s
```

```r
cat("Time for testing model=", tm_test_gbm[1], "s \n")
```

```
## Time for testing model= 13.99 s
```

Advanced Model: xgboost

Since we are dealing with the imbalanced data and there are too few examples of the minority class, we first use SMOTE to get a balanced dataset.

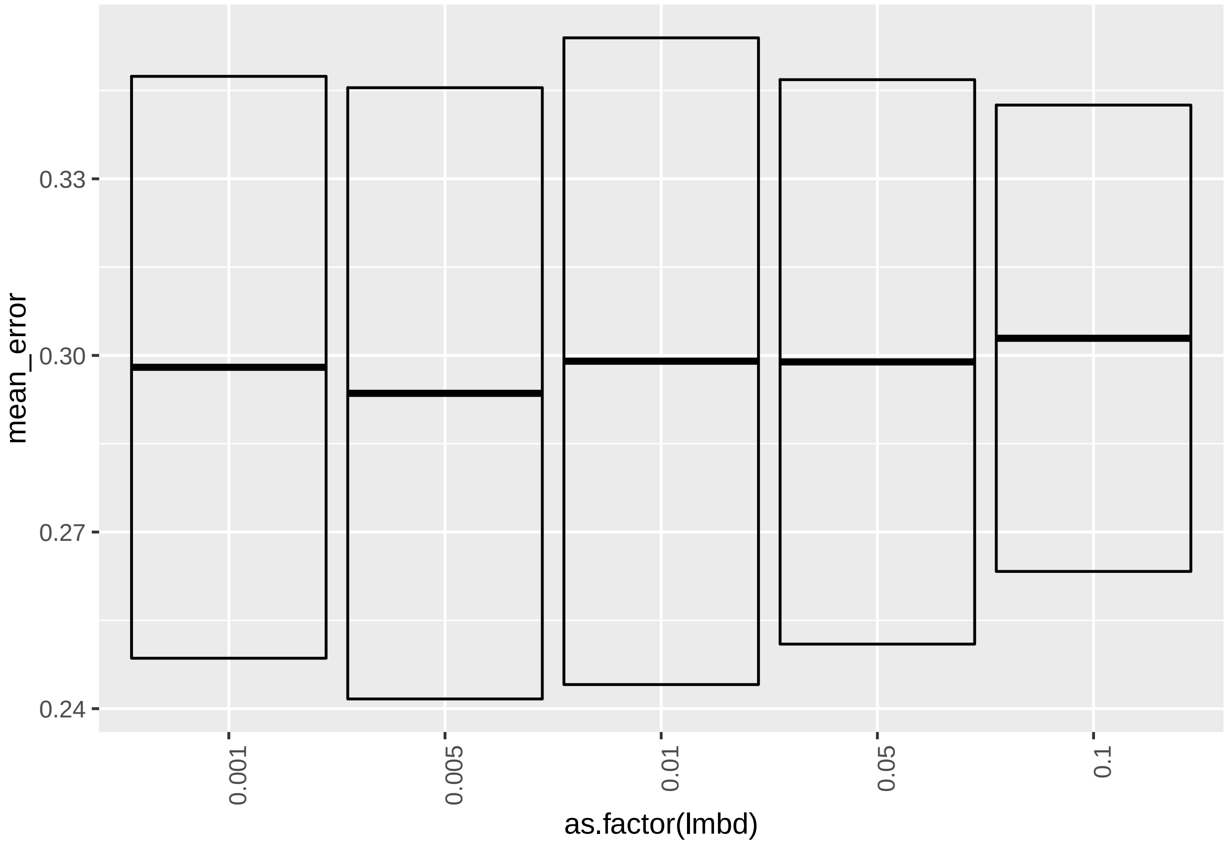
```r
feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)

# set balance.data = TRUE to generate the csv file
if (balance.data) {
  library(DMwR)
  dat_train_balanced <- SMOTE(label ~., data = dat_train,
                              perc.over=100, perc.under=200)
  save(dat_train_balanced, file="../output/dat_train_balanced.csv")
} else {
  load(file="../output/dat_train_balanced.csv")
}

label_train_balanced <- as.integer(dat_train_balanced$label)
feature_train_balanced <- as.matrix(dat_train_balanced[,-6007])
```

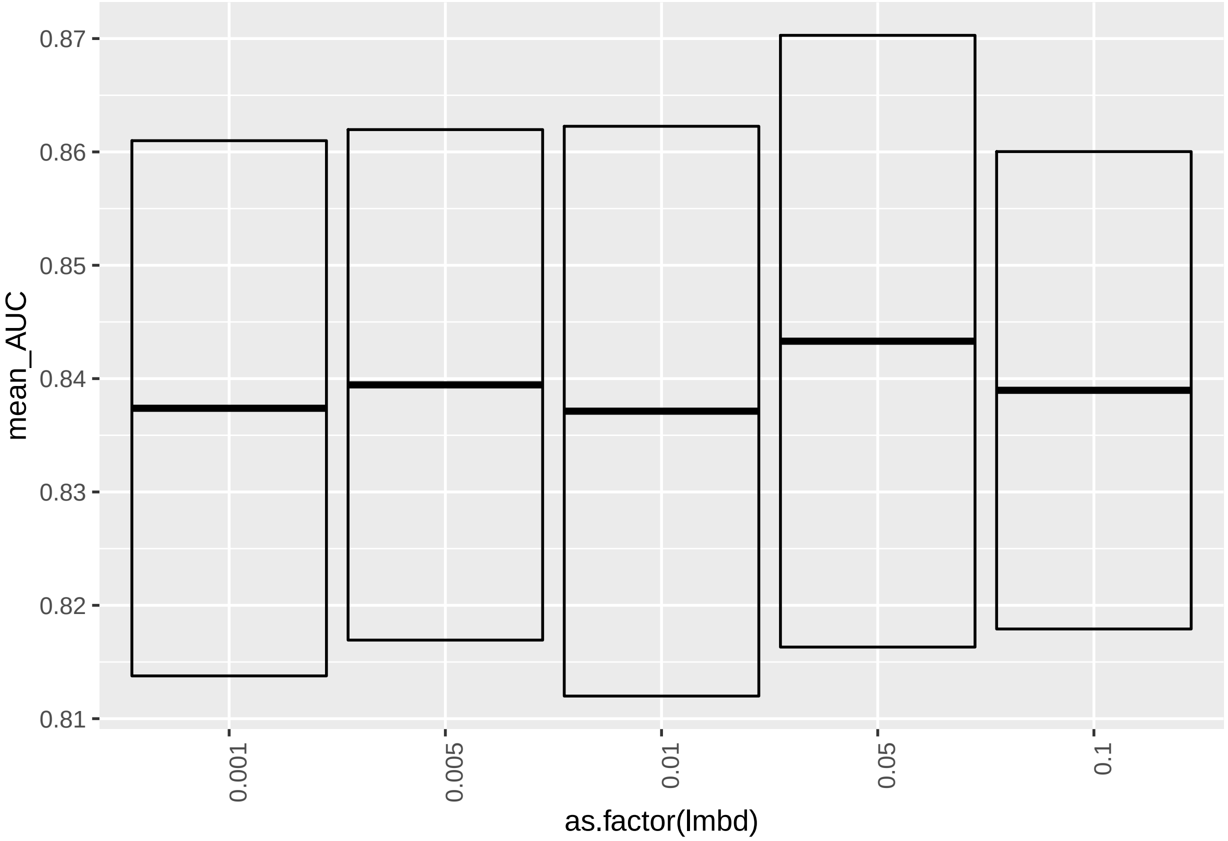
R setup and train

```r
source("../lib/train_XGBoost.R")
source("../lib/test_XGBoost.R")
source("../lib/cross_vallidation_XGBoost.R")

sample.reweight = FALSE

if(run.cv.xgb){
  xgb_res_cv <- matrix(0, nrow = length(lmbd), ncol = 4)
  for(i in 1:length(lmbd)){
    cat("lambda = ", lmbd[i], "\n")
    xgb_res_cv[i,] <- xgbcv.function(features = feature_train_balanced,
                                     labels = label_train_balanced, K,
                                     eta_val =0.08,
                                     lmd = lmbd[i],gamma = 0.4,md = 5,nr = 200,
                                     reweight = sample.reweight)
  save(xgb_res_cv, file="../output/xgb_res_cv.RData")
  }
}else{
  load("../output/xgb_res_cv.RData")
}
```

Visualize XGB cross-validation results.

```
xgb_res_cv <- as.data.frame(xgb_res_cv)
colnames(xgb_res_cv) <- c("mean_error", "sd_error", "mean_AUC", "sd_AUC")
xgb_res_cv$k = as.factor(lmbd)

if(run.cv.xgb.plot){
  p1 <- xgb_res_cv %>%
    ggplot(aes(x = as.factor(lmbd), y = mean_error,
               ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

  p2 <- xgb_res_cv %>%
    ggplot(aes(x = as.factor(lmbd), y = mean_AUC,
               ymin = mean_AUC - sd_AUC, ymax = mean_AUC + sd_AUC)) +
    geom_crossbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

  print(p1)
  print(p2)
}
```





Choose best model

```
par_best <- lmbd[which.max(xgb_res_cv$mean_AUC)] # lmbd[which.max(res_cv$mean_AUC)]

if (run.train.xgb){
  xgb_tm_train <- system.time(fit_train <- xgbtrain(feature_train_balanced, label_train_balanced, w = NU
  save(fit_train, xgb_tm_train, file="../output/xgb_fit_train.RData")
} else {
  load(file="../output/xgb_fit_train.RData")
}
```

Test on test set

```
xgb_tm_test = NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test.xgb){
  load(file="../output/xgb_fit_train.RData")
  xgb_tm_test <- system.time({prob_pred <- xgbtest(fit_train, feature_test);
                          label_pred <- ifelse(prob_pred >= 0.5, 1, 0)})
}

#evaluation
#accu_xgboost <- sum(weight_test * (label_pred == label_test)) / sum(weight_test)
accu_xgboost <- mean(label_pred == label_test)
tpr.fpr <- WeightedROC(prob_pred, label_test)
auc_xgboost <- WeightedAUC(tpr.fpr)

cat("The accuracy of model:", model_labels[which.max(xgb_res_cv$mean_AUC)], "is", accu_xgboost*100, "%.
```

```
## The accuracy of model: xgboost with lambda = 0.05 is 73 %.
```

```
cat("The AUC of model:", model_labels[which.max(xgb_res_cv$mean_AUC)], "is", auc_xgboost, ".\n")
```

```
## The AUC of model: xgboost with lambda = 0.05 is 0.04646737 .
```

```
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
```

```
## Time for constructing training features= 1.147 s
```

```
cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
```

```
## Time for constructing testing features= 0.199 s
```

```
cat("Time for training model=", xgb_tm_train[1], "s \n")
```

```
## Time for training model= 168.574 s
```

```
cat("Time for testing model=", xgb_tm_test[1], "s \n")
```

```
## Time for testing model= 0.15 s
```

Other model 1: PCA+LDA model

In this model, we first use principle component analysis (PCA) method to reduce dimension. Then we use LDA as our classification model.

step 4: find the best set of principle components

call the train model and test model from library.

```r
source("../lib/cross_validation_pca_lda.R")
source("../lib/train_lda.R")
source("../lib/test_lda.R")
```

model selection with cross-validation Do model selection by choosing among different numbers of principle components.

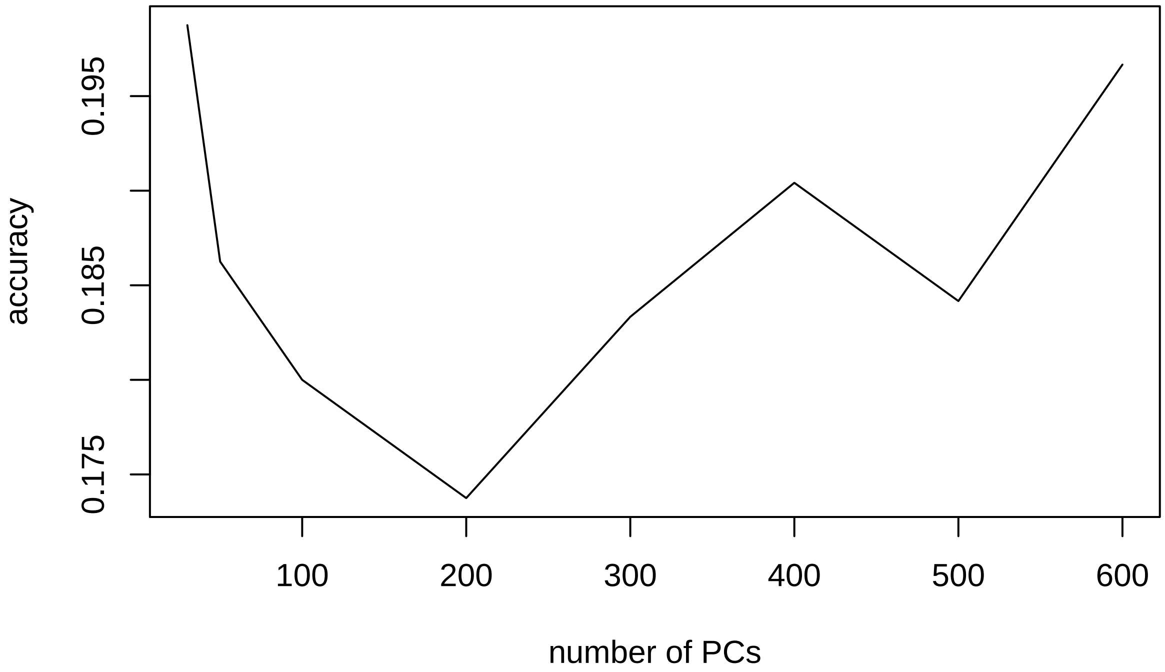
```r
if(run.cv.pca) {
  pc <- c()
  for(i in 1:length(pca.list)) {
    cat("number of principle components = ", pca.list[i], "\n")
    pc[i] <- pca_evaluation(features = feature_train, labels = label_train,
                            K = 5, num_pc = pca.list[i])
  }
  save(pc, file="../output/pc_lda.RData")
} else {
  load(file="../output/pc_lda.RData")
}
```

Visualize results and choose the best number of PCs: since we want to maximize the accuracy of our model (minimum mean_error), we choose 200 principle components.

```r
res_cv_pca <- data.frame(pca.list, pc)
colnames(res_cv_pca) <- c("number of pc", "mean_error")
res_cv_pca[order(res_cv_pca$mean_error),]
```

```
##   number of pc mean_error
## 4          200  0.1737500
## 3          100  0.1800000
## 5          300  0.1833333
## 7          500  0.1841667
## 2           50  0.1862500
## 6          400  0.1904167
## 8          600  0.1966667
## 1           30  0.1987500
```

```r
plot(pca.list, pc, type="l", xlab="number of PCs", ylab="accuracy")
```



```
par_best <- pca.list[which.min(pc)]
```

Train the model with the entire training set using the selected model (model parameter) via cross-validation.

```
if(run.pca) {
  #training dataset
  pca <- prcomp(feature_train)
  pca_result <- data.frame(pca$x[,1:200])
  #test dataset
  pca_result2 <- predict(pca, feature_test)
  test_pca <- data.frame(pca_result2[,1:200])
  save(pca, pca_result, test_pca, file="../output/pcaOutput.RData")
} else {
  load(file="../output/pcaOutput.RData")
}

dat_train_pca <- cbind(pca_result, label_train)
dat_test_pca <- cbind(test_pca, label_test)
```

```
if(run.train.lda) {
  tm_train_lda <- system.time(fit_train_lda <- train(dat_train_pca))
  save(fit_train_lda, tm_train_lda, file="../output/fit_train_lda.RData")
} else {
  load(file="../output/fit_train_lda.RData")
}
```

step 5: run test on test image

```
tm_test_lda <- NA

if(run.test.lda) {
  load(file="../output/fit_train_lda.RData")
  tm_test_lda <- system.time(pred <- test(fit_train_lda, dat_test_pca))
}
```

```
label_test <- as.integer(dat_test$label)
weight_test <- rep(NA, length(label_test))
for (v in unique(label_test)) {
  weight_test[label_test == v] <- 0.5*length(label_test) / length(label_test[label_test==v])
}

# accuracy of pca+lda model
accu_lda <- mean(dat_test_pca$label_test == pred$class)
cat("The accuracy of the lda model with pca is: ", accu_lda*100, "%.\n")
```

Evaluation

```
## The accuracy of the lda model with pca is:  73.5 %.
```

```
#ROC and AUC of model
#labels <- as.factor(ifelse(dat_test_pca$test_label == 2,1,0))
prob_pred <- pred$posterior[,2]
label_pred <- pred$class
label_pred <- factor(ifelse(label_pred == 2, 1, 0))
auc_lda <- auc(roc(predictor = prob_pred, response = factor(label_test)), min=0, max=1)
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
cat("The AUC of the lda model with pca is: ", auc_lda, ".\n")
```

```
## The AUC of the lda model with pca is:  0.5136 .
```

```
cat("Time for training lda model=", tm_train_lda[1], "s \n")
```

Summarize running time

```
## Time for training lda model= 0.503 s
```

```
cat("Time for testing lda model=", tm_test_lda[1], "s \n")
```

```
## Time for testing lda model= 0.021 s
```

Model 2: Random Forest model (unweighted)

Step 4: Train a classification model with training features and responses

```
source("../lib/randomForest.R")
source("../lib/randomforestTrain.R")
source("../lib/randomforestTest.R")
```

model selection with cross-validation Do model selection by choosing among different values of training model parameters.

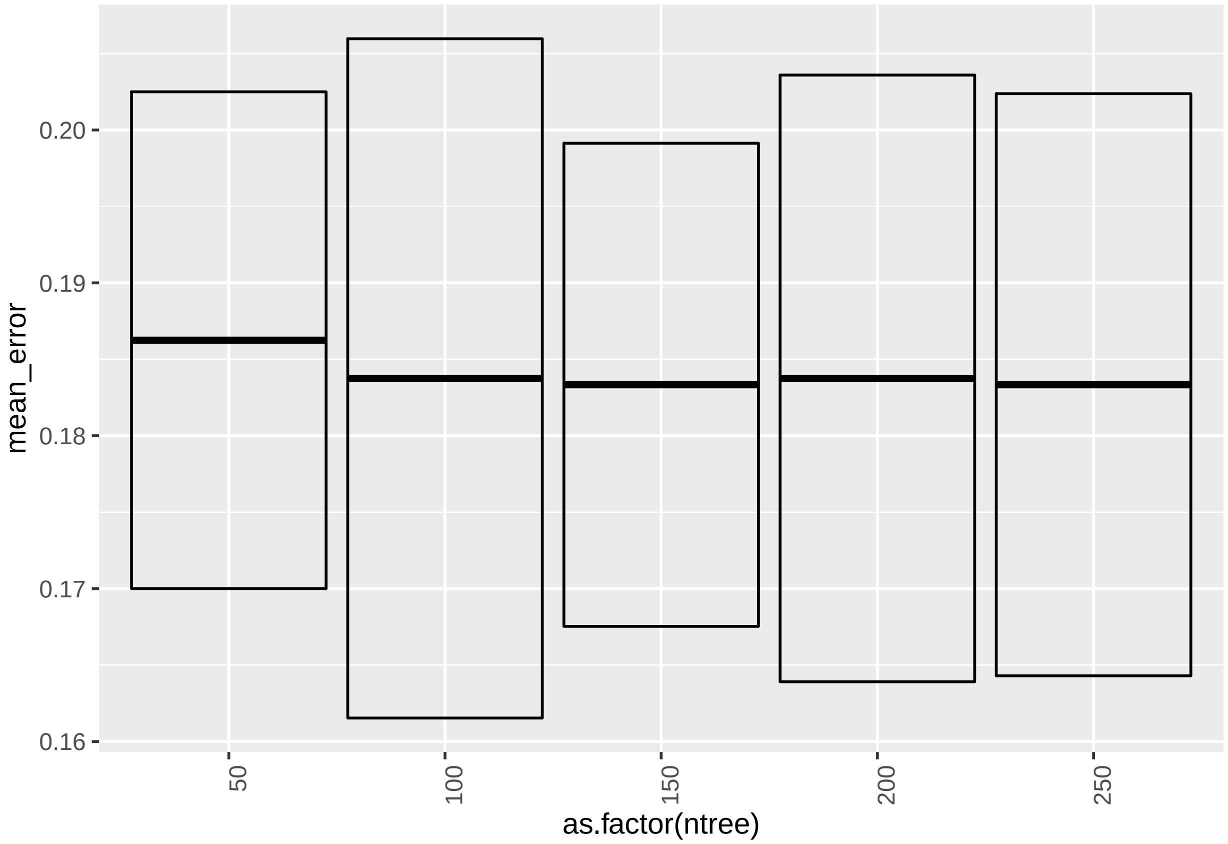
```
if(run.cv.randomForest){
  res_cv_randomForest <- matrix(0, nrow = length(ntree), ncol = 2)
  for (i in 1:length(ntree)){
cat("ntree =", ntree[i],"\n")
res_cv_randomForest[i,] <- cv.randomForest.function(features = feature_train, labels = label_train,K,ntr
}
  save(res_cv_randomForest, file="../output/res_cv_randomForest.RData")
}else{
  load("../output/res_cv_randomForest.RData")
}
```

Visualize cross-validation results.

```
res_cv_randomForest <- as.data.frame(res_cv_randomForest)
colnames(res_cv_randomForest) <- c("mean_error", "sd_error")
res_cv_randomForest$ntree = as.integer(ntree)
res_cv_randomForest
```

```
##   mean_error   sd_error ntree
## 1  0.1862500 0.01624466    50
## 2  0.1837500 0.02221463   100
## 3  0.1833333 0.01579766   150
## 4  0.1837500 0.01984095   200
## 5  0.1833333 0.01903715   250
```

```
plot_meanError_randomForest <- res_cv_randomForest %>%
ggplot(aes(x = as.factor(ntree), y = mean_error,
           ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
  geom_crossbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(plot_meanError_randomForest)
```



- Choose the best parameter value by choosing the one with the lowest mean error

```
ntree_best_randomForest <- res_cv_randomForest$ntree[which.min(res_cv_randomForest$mean_error)]

save(ntree_best_randomForest,file = "../output/ntree_best_randomForest.Rdata")
cat("ntree_best_randomForest=", ntree_best_randomForest)
```

```
## ntree_best_randomForest= 150
```

- Train the model with the entire training set using the selected model (model parameter) via cross validation.

```
feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)
tm_train=NA
tm_train <- system.time(fit_train_randomForest <- train_randomForest(feature_train, label_train, ntree=n
save(fit_train_randomForest, file="../output/fit_train_randomForest.RData")
```

Step 5: Run test on test images

```
tm_test=NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test.randomForest){
  load(file="../output/fit_train_randomForest.RData")
  tm_test <- system.time(label_pred_randomForest <- ifelse(test_randomForest(fit_train_randomForest, fea
}
```

- evaluation

```r
test_label <- dat_test$label
accurancy.randomForest <- mean(test_label == label_pred_randomForest)
tpr.fpr <- WeightedROC(as.numeric(label_pred_randomForest), test_label)
auc.randomForest <- WeightedAUC(tpr.fpr)

cat("The accuracy of random forest model:", randomForest_model_labels[which.min(res_cv_randomForest$mean
```

```
## The accuracy of random forest model: Random Forest with number of trees = 150 is 80.66667 %.
```

```r
cat("The AUC of random forest model:", randomForest_model_labels[which.min(res_cv_randomForest$mean_erro
```

```
## The AUC of random forest model: Random Forest with number of trees = 150 is 0.5389474 .
```

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

```r
cat("Time for training random forest model=", tm_train[1], "s \n")
```

```
## Time for training random forest model= 185.56 s
```

```r
cat("Time for testing random forest model=", tm_test[1], "s \n")
```

```
## Time for testing random forest model= 0.155 s
```

Model 3: Random Forest model with weight

Step 4: Train a classification model with training features and responses

Do model selection by choosing among different values of training model parameters.

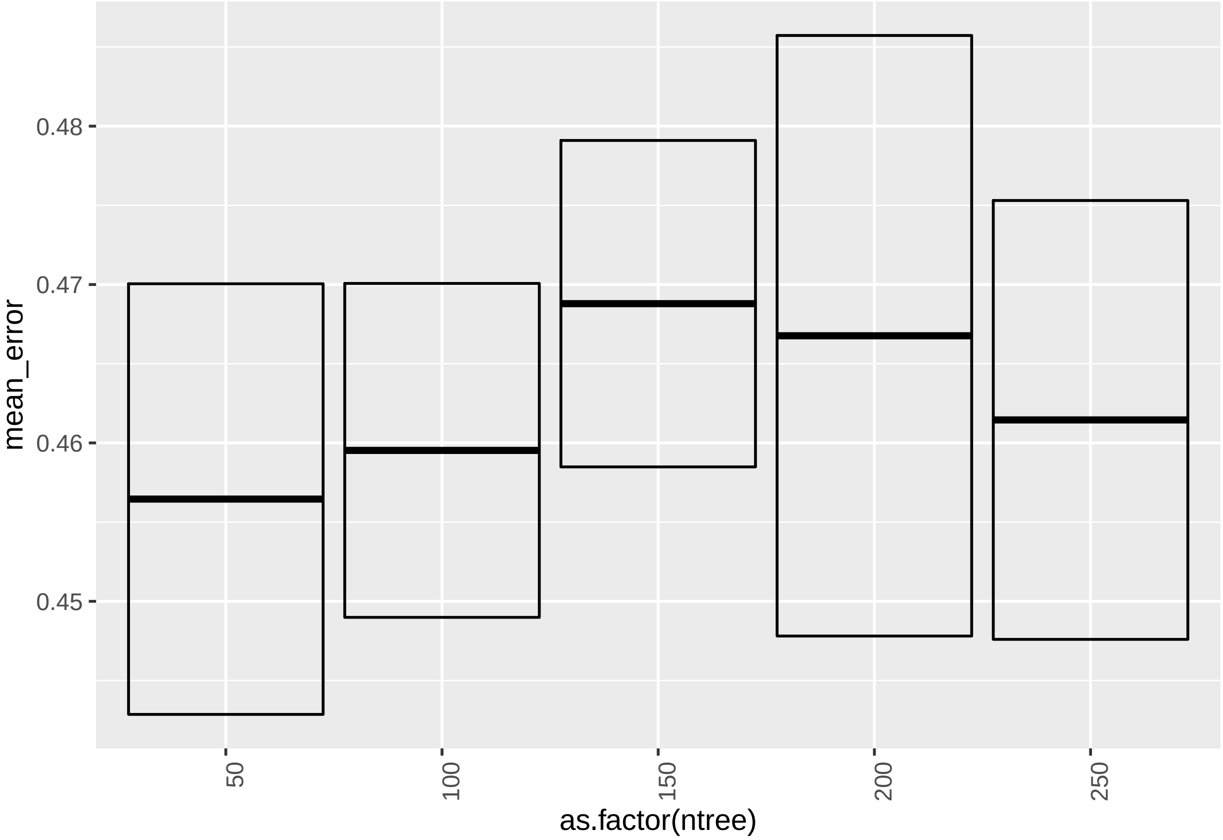
```r
source("../lib/randomForestWeight.R")
source("../lib/randomforestTrain.R")
source("../lib/randomforestTest.R")

feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)

if(run.cv.randomForestWeight){
  res_cv_randomForestWeight <- matrix(0, nrow = length(ntree), ncol = 2)
  for (i in 1:length(ntree)){
cat("ntree =", ntree[i],"\n")
res_cv_randomForestWeight[i,] <- cv.randomForestWeight.function(features = feature_train, labels = label
}
  save(res_cv_randomForestWeight, file="../output/res_cv_randomForestWeight.RData")
}else{
  load("../output/res_cv_randomForestWeight.RData")
}
```

Visualize cross-validation results.

```r
res_cv_randomForestWeight <- as.data.frame(res_cv_randomForestWeight)
colnames(res_cv_randomForestWeight) <- c("mean_error", "sd_error")
res_cv_randomForestWeight$ntree = as.integer(ntree)
res_cv_randomForestWeight
```

```
##   mean_error   sd_error ntree
## 1  0.4564540 0.01359338    50
## 2  0.4595283 0.01054060   100
## 3  0.4687925 0.01030525   150
## 4  0.4667663 0.01895810   200
## 5  0.4614523 0.01385381   250
```

```
plot_meanError_randomForestWeight <- res_cv_randomForestWeight %>%
ggplot(aes(x = as.factor(ntree), y = mean_error,
ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
geom_crossbar() +
theme(axis.text.x = element_text(angle = 90, hjust = 1))
print(plot_meanError_randomForestWeight)
```



- Choose the best parameter value by choosing the one with the lowest mean error

```
  ntree_best_randomForestWeight <- res_cv_randomForestWeight$ntree[which.min(res_cv_randomForestWeight$
save(ntree_best_randomForestWeight,file = "../output/ntree_best_randomForestWeight.Rdata")
cat("ntree_best_randomForestWeight=", ntree_best_randomForestWeight)
```

```
## ntree_best_randomForestWeight= 50
```

- Train the model with the entire training set using the selected model (model parameter) via cross validation

```
weight_train <- rep(NA, length(label_train))
for (v in unique(label_train)){
  weight_train[label_train == v] = 0.5 * length(label_train) / length(label_train[label_train == v])
```

```
}

tm_train=NA
tm_train <- system.time(fit_train_randomForestWeight <- train_randomForest(feature_train, label_train, r
save(fit_train_randomForestWeight, file="../output/fit_train_randomForestWeight.RData")
```

###Step 5: Run test on test images

```
tm_test=NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test.randomForestWeight){
  load(file="../output/fit_train_randomForestWeight.RData")
  tm_test <- system.time(label_pred_randomForestWeight <- ifelse(test_randomForest(fit_train_randomFores
}
```

- evaluation

```
test_label <- dat_test$label
label_test <- as.integer(dat_test$label)
## reweight the test data to represent a balanced label distribution
weight_test <- rep(NA, length(label_test))
for (v in unique(label_test)){
  weight_test[label_test == v] = 0.5 * length(label_test) /
    length(label_test[label_test == v])
}
accurancy.randomForestWeight <- sum(weight_test * (test_label == label_pred_randomForestWeight))/sum(wei
tpr.fpr <- WeightedROC(as.numeric(label_pred_randomForestWeight), test_label)
auc.randomForestWeight <- WeightedAUC(tpr.fpr)

cat("The accuracy of random forest with weight model:", randomForestWeight_model_labels[which.min(res_cv
```

```
## The accuracy of random forest with weight model: randomForestWithWeight with number of trees = 50 is
```

```
cat("The AUC of random forest with weight model:", randomForestWeight_model_labels[which.min(res_cv_rand
```

```
## The AUC of random forest with weight model: randomForestWithWeight with number of trees = 50 is 0.54
```

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

```
#cat("Time for constructing training features=", tm_feature_train[1], "s \n")
#cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
cat("Time for training random forest with weightmodel=", tm_train[1], "s \n")
```

```
## Time for training random forest with weightmodel= 63.951 s
```

```
cat("Time for testing random forest with weight model=", tm_test[1], "s \n")
```

```
## Time for testing random forest with weight model= 0.1 s
```

Model 4: svm (kernel = polynomial, linear, radial, sigmoid)

- Polynomial SVM:

```
run.svm.polynomial.cv.test <- FALSE
run.train.svm.polynomial <- FALSE
plot.svm.polynomial <- FALSE
```

Cross validation:

```
source("../lib/svm_polynomial_best_degree.R")
```

```
max.deg = 4
testset <- as.matrix(dat_test[, -6007])
if (run.svm.polynomial.cv.test) {
  acc.array <- svm_polynomial_best_degree(dat_train, dat_test, 5, testset) # up to degree 5
  max.deg <- which.max(acc.array)
} else {
  max.deg = 4
}
```

Train:

```
tm_svm_polynomial_train <- NA
if (run.train.svm.polynomial) {
  tm_svm_polynomial_train <- system.time( sclassifier.polynomial <- svm(formula=label~., data=dat_train
  save(tm_svm_polynomial_train, sclassifier.polynomial, file="../output/svmclassifier_polynomial.RData")
} else{
  load(file="../output/svmclassifier_polynomial.RData")
}
```

Test:

```
tm_svm_polynomial_test <- NA
testset <- as.matrix(dat_test[, -6007])
tm_svm_polynomial_test <- system.time( ypred <- predict(sclassifier.polynomial, testset))
```

Plot (feature 1 and feature 2) :

```
if (plot.svm.polynomial) {
  plot(sclassifier.polynomial, dat_train, formula=feature1~feature2, fill=T)
}
```

Evaluate:

```
tab_svm <- table(ypred, dat_test$label)
n1 <- tab_svm[1]
n2 <- tab_svm[2]
n3 <- tab_svm[3]
n4 <- tab_svm[4]
num <- n1+n4
den <- n1+n2+n3+n4
acc <- num/den
cat("The accuracy of svm polynomial model:", acc*100, "%.\n")
```

```
## The accuracy of svm polynomial model: 84.66667 %.
```

Running Times:

```
cat("Time for training model=", tm_svm_polynomial_train[1], "s \n")
```

```
## Time for training model= 108.506 s
```

```
cat("Time for testing model=", tm_svm_polynomial_test[1], "s \n")
```

```
## Time for testing model= 10.139 s
```

- Probabilistic Polynomial SVM:

```
run.svm.polynomial.prob.cv.test <- FALSE
run.svm.polynomial.prob.train <- FALSE
plot.svm.polynomial.prob <- FALSE
```

Cross validation:

```
source("../lib/svm_polynomial_prob_best_degree.R")
```

```
max.deg = 4
if (run.svm.polynomial.prob.cv.test) {
  acc.array <- svm_polynomial_prob_best_degree(dat_train, dat_test, 5, testset) # up to degree 5
  max.deg <- which.max(acc.array)
} else {
  max.deg = 4
}
```

Train:

```
tm_svm_polynomial_prob_train <- NA
if (run.train.svm.polynomial) {
  tm_svm_polynomial_prob_train <- system.time(sclassifier.polynomial.prob <- svm(formula=label~., data=
  save(tm_svm_polynomial_prob_train, sclassifier.polynomial.prob, file="../output/svmclassifier_polynom
} else{
  load(file="../output/svmclassifier_polynomial_prob.RData")
}
```

Plot (feature 1 and feature 2) :

```
if (plot.svm.polynomial.prob) {
  plot(sclassifier.polynomial.prob, dat_train, formula=feature1~feature2, fill=T)
  }
```

Test:

```
tm_svm_polynomial_prob_test <- NA
testset <- as.matrix(dat_test[, -6007])
tm_svm_polynomial_prob_test <- system.time(ypred_prob <- predict(sclassifier.polynomial.prob, testset,
```

Evaluate accuracy and auc:

```
tab_svm <- table(ypred_prob, dat_test$label)
n1 <- tab_svm[1]
n2 <- tab_svm[2]
n3 <- tab_svm[3]
n4 <- tab_svm[4]
num <- n1+n4
den <- n1+n2+n3+n4
acc <- num/den
cat("The accuracy of svm probabilistic polynomial model:", acc*100, "%.\n")
```

```
## The accuracy of svm probabilistic polynomial model: 83.33333 %.
```

```
# AUC
prob_vec_poly <- attr(ypred_prob, "probabilities")
```

```
prob_vec_poly_0 <- prob_vec_poly[,1]
prob_vec_poly_1 <- prob_vec_poly[,2]
r_prob <- roc(predictor= prob_vec_poly_1, response= factor(dat_test$label))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc_prob <- auc(r_prob, min=0, max=1)
cat("The auc of svm prob. polynomial model:", auc_prob)
```

```
## The auc of svm prob. polynomial model: 0.9297853
```

Running Times:

```
cat("Time for training model=", tm_svm_polynomial_prob_train[1], "s \n")
```

```
## Time for training model= 254.704 s
```

```
cat("Time for testing model=", tm_svm_polynomial_prob_test[1], "s \n")
```

```
## Time for testing model= 18.21 s
```

- Linear SVM:

```
run.train.svm.linear <- FALSE
plot.svm.linear <- FALSE
```

Train:

```
tm_svm_linear_train <- NA
if (run.train.svm.linear) {
  tm_svm_linear_train <- system.time(sclassifier.linear <- svm(formula=label~., data=dat_train, type="C
  save(tm_svm_linear_train, sclassifier.linear, file="../output/svmclassifier_linear.RData")
 } else{
  load(file="../output/svmclassifier_linear.RData")
 }
```

Test:

```
tm_svm_linear_test <- NA
testset <- as.matrix(dat_test[, -6007])
tm_svm_linear_test <- system.time(ypred <- predict(sclassifier.linear, testset))
```

Plot (feature 1 and feature 2) :

```
if (plot.svm.linear) {
  plot(sclassifier.linear, dat_train, formula=feature1~feature2, fill=T)
}
```

Evaluate:

```
tab_svm <- table(ypred, dat_test$label)
n1 <- tab_svm[1]
n2 <- tab_svm[2]
n3 <- tab_svm[3]
n4 <- tab_svm[4]
num <- n1+n4
den <- n1+n2+n3+n4
acc <- num/den
cat("The accuracy of svm linear model:", acc*100, "%.\n")
```

```
## The accuracy of svm linear model: 95.66667 %.
```

Running Times:

```
cat("Time for training model=", tm_svm_linear_train[1], "s \n")
```

```
## Time for training model= 101.09 s
```

```
cat("Time for testing model=", tm_svm_linear_test[1], "s \n")
```

```
## Time for testing model= 5.742 s
```

- Probabilistic Linear SVM:

```
run.train.svm.linear.prob <- FALSE
plot.svm.linear.prob <- FALSE
```

Train:

```
tm_svm_linear_prob_train <- NA
if (run.train.svm.linear.prob) {
  tm_svm_linear_prob_train <- system.time(sclassifier.linear.prob <- svm(formula=label~., data=dat_trai
  save(tm_svm_linear_prob_train, sclassifier.linear.prob, file="../output/svmclassifier_linear_prob.RDa
 } else{
  load(file="../output/svmclassifier_linear_prob.RData")
 }
```

Plot (feature 1 and feature 2) :

```
if (plot.svm.linear.prob) {
  plot(sclassifier.linear.prob, dat_train, formula=feature1~feature2, fill=T)
}
```

Test:

```
tm_svm_linear_prob_test <- NA
testset <- as.matrix(dat_test[, -6007])
tm_svm_linear_prob_test <- system.time(ypred_prob <- predict(sclassifier.linear.prob, testset, decision
```

Evaluate accuracy and auc:

```
tab_svm <- table(ypred_prob, dat_test$label)
n1 <- tab_svm[1]
n2 <- tab_svm[2]
n3 <- tab_svm[3]
n4 <- tab_svm[4]
num <- n1+n4
den <- n1+n2+n3+n4
acc <- num/den
cat("The accuracy of svm probabilistic linear model:", acc*100, "%.\n")
```

```
## The accuracy of svm probabilistic linear model: 80.66667 %.
```

```
# AUC
prob_vec <- attr(ypred_prob, "probabilities")
prob_vec_0 <- prob_vec[,1]
prob_vec_1 <- prob_vec[,2]
r_prob <- roc(predictor= prob_vec_1, response= factor(dat_test$label))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
auc_prob <- auc(r_prob, min=0, max=1)
cat("The auc of svm prob. linear model:", auc_prob)
```

```
## The auc of svm prob. linear model: 0.9177432
```

Running Times:

```r
cat("Time for training model=", tm_svm_linear_prob_train[1], "s \n")
```

```
## Time for training model= 261.033 s
```

```r
cat("Time for testing model=", tm_svm_linear_prob_test[1], "s \n")
```

```
## Time for testing model= 10.232 s
```

- Radial SVM:

```r
run.train.svm.radial <- TRUE
plot.svm.radial <- FALSE
```

Train:

```r
tm_svm_radial_train <- NA
if (run.train.svm.radial) {
  tm_svm_radial_train <- system.time(sclassifier.radial <- svm(formula=label~., data=dat_train, type="C
  save(tm_svm_radial_train, sclassifier.radial, file="../output/svmclassifier_radial.RData")
} else {
  load(file="../output/svmclassifier_radial.RData")
  }
```

Test:

```r
tm_svm_radial_test <- NA
testset <- as.matrix(dat_test[, -6007])
tm_svm_radial_test <- system.time(ypred <- predict(sclassifier.radial, testset))
```

Plot (feature 1 and feature 2) :

```r
if (plot.svm.radial) {
  plot(sclassifier.radial, dat_train, formula=feature1~feature2, fill=T)
}
```

Evaluate:

```r
tab_svm <- table(ypred, dat_test$label)
n1 <- tab_svm[1]
n2 <- tab_svm[2]
n3 <- tab_svm[3]
n4 <- tab_svm[4]
num <- n1+n4
den <- n1+n2+n3+n4
acc <- num/den
cat("The accuracy of svm radial model:", acc*100, "%.\n")
```

```
## The accuracy of svm radial model: 81.66667 %.
```

Running Times:

```r
cat("Time for training model=", tm_svm_radial_train[1], "s \n")
```

```
## Time for training model= 98.106 s
```

```
cat("Time for testing model=", tm_svm_radial_test[1], "s \n")
```

```
## Time for testing model= 8.959 s
```

- Probabilistic Radial SVM:

```
run.train.svm.radial.prob <- FALSE
plot.svm.radial.prob <- FALSE
```

Train:

```
tm_svm_radial_prob_train <- NA
if (run.train.svm.radial.prob) {
  tm_svm_radial_prob_train <- system.time(sclassifier.radial.prob <- svm(formula=label~., data=dat_trai
  save(tm_svm_radial_prob_train, sclassifier.radial.prob, file="../output/svmclassifier_radial_prob.RDa
} else {
  load(file="../output/svmclassifier_radial_prob.RData")
  }
```

Plot (feature 1 and feature 2) :

```
if (plot.svm.radial.prob) {
  plot(sclassifier.radial.prob, dat_train, formula=feature1~feature2, fill=T)
}
```

Test:

```
tm_svm_radial_prob_test <- NA
testset <- as.matrix(dat_test[, -6007])
tm_svm_radial_prob_test <- system.time(ypred_prob <- predict(sclassifier.radial.prob, testset, decision
```

Evaluate accuracy and auc:

```
tab_svm <- table(ypred_prob, dat_test$label)
n1 <- tab_svm[1]
n2 <- tab_svm[2]
n3 <- tab_svm[3]
n4 <- tab_svm[4]
num <- n1+n4
den <- n1+n2+n3+n4
acc <- num/den
cat("The accuracy of svm probabilistic radial model:", acc*100, "%.\n")
```

```
## The accuracy of svm probabilistic radial model: 90 %.
```

```
# AUC
prob_vec <- attr(ypred_prob, "probabilities")
prob_vec_0 <- prob_vec_poly[,1]
prob_vec_1 <- prob_vec_poly[,2]
r_prob <- roc(predictor= prob_vec_1, response= factor(dat_test$label))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc_prob <- auc(r_prob, min=0, max=1)
cat("The auc of svm prob. radial model:", auc_prob)
```

```
## The auc of svm prob. radial model: 0.9297853
```

Running Times:

```r
cat("Time for training model=", tm_svm_radial_prob_train[1], "s \n")
```

```
## Time for training model= 215.916 s
```

```r
cat("Time for testing model=", tm_svm_radial_prob_test[1], "s \n")
```

```
## Time for testing model= 15.377 s
```

- Sigmoid SVM:

```r
run.train.svm.sigmoid <- FALSE
plot.svm.sigmoid <- FALSE
```

Train:

```r
tm_svm_sigmoid_train <- NA
if (run.train.svm.sigmoid) {
  tm_svm_sigmoid_train <- system.time(sclassifier.sigmoid <- svm(formula=label~., data=dat_train, type=
  save(tm_svm_sigmoid_train, sclassifier.sigmoid, file="../output/svmclassifier_sigmoid.RData")
} else {
  load(file="../output/svmclassifier_sigmoid.RData")
  }
```

Test:

```r
tm_svm_sigmoid_test <- NA
testset <- as.matrix(dat_test[, -6007])
tm_svm_sigmoid_test <- system.time(ypred <- predict(sclassifier.sigmoid, testset))
```

Plot (feature 1 and feature 2) :

```r
if (plot.svm.sigmoid) {
  plot(sclassifier.sigmoid, dat_train, formula=feature1~feature2, fill=T)
}
```

Evaluate:

```r
tab_svm <- table(ypred, dat_test$label)
n1 <- tab_svm[1]
n2 <- tab_svm[2]
n3 <- tab_svm[3]
n4 <- tab_svm[4]
num <- n1+n4
den <- n1+n2+n3+n4
acc <- num/den
cat("The accuracy of svm sigmoid model:", acc*100, "%.\n")
```

```
## The accuracy of svm sigmoid model: 72.33333 %.
```

Running Times:

```r
cat("Time for training model=", tm_svm_sigmoid_train[1], "s \n")
```

```
## Time for training model= 59.99 s
```

```r
cat("Time for testing model=", tm_svm_sigmoid_test[1], "s \n")
```

```
## Time for testing model= 5.668 s
```

- Probabilistic Sigmoid SVM:

```
run.train.svm.sigmoid.prob <- FALSE
plot.svm.sigmoid.prob <- FALSE
```

Train:

```
tm_svm_sigmoid_prob_train <- NA
if (run.train.svm.sigmoid.prob) {
  tm_svm_sigmoid_prob_train <- system.time(sclassifier.sigmoid.prob <- svm(formula=label~., data=dat_tra
  save(tm_svm_sigmoid_prob_train, sclassifier.sigmoid.prob, file="../output/svmclassifier_sigmoid_prob.R
} else {
  load(file="../output/svmclassifier_sigmoid_prob.RData")
  }
```

Test:

```
tm_svm_sigmoid_prob_test <- NA
testset <- as.matrix(dat_test[, -6007])
tm_svm_sigmoid_prob_test <- system.time(ypred_prob <- predict(sclassifier.sigmoid.prob, testset, decisi
```

Plot (feature 1 and feature 2) :

```
if (plot.svm.sigmoid.prob) {
  plot(sclassifier.sigmoid.prob, dat_train, formula=feature1~feature2, fill=T)
}
```

Evaluate accuracy and auc:

```
tab_svm <- table(ypred_prob, dat_test$label)
n1 <- tab_svm[1]
n2 <- tab_svm[2]
n3 <- tab_svm[3]
n4 <- tab_svm[4]
num <- n1+n4
den <- n1+n2+n3+n4
acc <- num/den
cat("The accuracy of svm probabilistic sigmoid model:", acc*100, "%.\n")
```

```
## The accuracy of svm probabilistic sigmoid model: 79.16667 %.
```

```
# AUC
prob_vec <- attr(ypred_prob, "probabilities")
prob_vec_0 <- prob_vec_poly[,1]
prob_vec_1 <- prob_vec_poly[,2]
r_prob <- roc(predictor= prob_vec_1, response= factor(dat_test$label))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc_prob <- auc(r_prob, min=0, max=1)
cat("The auc of svm prob. sigmoid model:", auc_prob)
```

```
## The auc of svm prob. sigmoid model: 0.9297853
```

Running Times:

```
cat("Time for training model=", tm_svm_sigmoid_prob_train[1], "s \n")
```

```
## Time for training model= 137.306 s
```

```
cat("Time for testing model=", tm_svm_sigmoid_prob_test[1], "s \n")
```

```
## Time for testing model= 10.244 s
```