Main

Chengliang Tang, Yujie Wang, Diane Lu, Tian Zheng

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.

```r
#Test Branch created
if(!require("EBImage")){
  install.packages("BiocManager")
  BiocManager::install("EBImage")
}
if(!require("R.matlab")){
  install.packages("R.matlab")
}
if(!require("readxl")){
  install.packages("readxl")
}

if(!require("dplyr")){
  install.packages("dplyr")
}
if(!require("readxl")){
  install.packages("readxl")
}

if(!require("ggplot2")){
  install.packages("ggplot2")
}

if(!require("caret")){
  install.packages("caret")
}

if(!require("glmnet")){
  install.packages("glmnet")
}

if(!require("WeightedROC")){
  install.packages("WeightedROC")
}

library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(glmnet)
library(WeightedROC)
```

New libraries

```r
if(!require("randomForest")){
 install.packages("randomForest")
}
```

```
## Warning: package 'randomForest' was built under R version 4.0.4
```

```r
if(!require("xgboost")){
 install.packages("xgboost")
}
```

```
## Warning: package 'xgboost' was built under R version 4.0.4
```

```r
if(!require("tibble")){
 install.packages("tibble")
}
if(!require("ROSE")){
 install.packages("ROSE")
}
```

```
## Warning: package 'ROSE' was built under R version 4.0.4
```

```r
if(!require("ggplot2")){
 install.packages("ggplot2")
}
if(!require("tidyverse")){
 install.packages("tidyverse")
}

if(!require("AUC")){
 install.packages("AUC")
}
if(!require("e1071")){
 install.packages("e1071")
}
if(!require("OpenImageR")){
 install.packages("OpenImageR")
}
```

```
## Warning: package 'OpenImageR' was built under R version 4.0.4
```

```r
if(!require("caTools")){
  install.packages("caTools")
}
```

```
## Warning: package 'caTools' was built under R version 4.0.4
```

```
library(OpenImageR)
library(AUC)
library(e1071)
library(randomForest)
library(xgboost)
library(tibble)
library(ROSE)
library(ggplot2)
library(tidyverse)
library(AUC)
library(e1071)
library(caTools)
```

Step 0 set work directories

```
set.seed(2020)
setwd("../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.

```
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
- (T/F) Use balanced data for both train and test set

```
run.cv <- F # run cross-validation on the training set
sample.reweight <- TRUE # run sample reweighting in model training
K <- 5  # number of CV folds
run.feature.train <- F # process features for training set
run.test <- TRUE # run evaluation on an independent test set
run.feature.test <- TRUE # process features for test set
run.balanced.data <- F
train.random.forest <- F
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications. In this Starter Code, we tune parameter lambda (the amount of shrinkage) for logistic regression with LASSO penalty.

```
lmbd = c(1e-3, 5e-3, 1e-2, 5e-2, 1e-1)
model_labels = paste("LASSO Penalty with lambda =", lmbd)
```

Step 2: import data and train-test split

```
#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.

```
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.

```
#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.

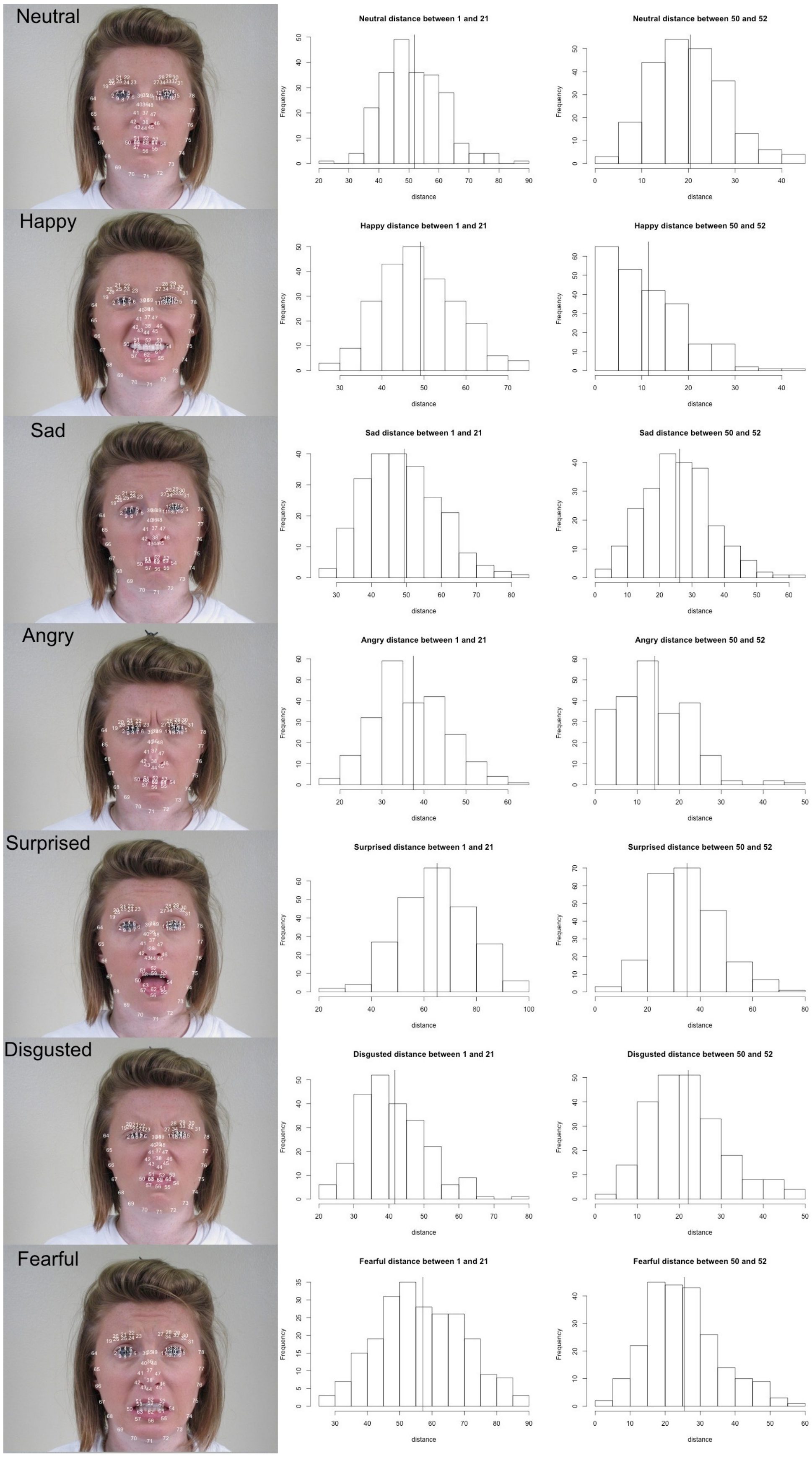

Figure 1: Figure1

`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

```r
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")
}
# transfer label column from factor to numeric
dat_train$label <- as.numeric(dat_train$label)-1
dat_test$label <- as.numeric(dat_test$label)-1
#Rebalancing training data-Bootstrap Random Over-Sampling Examples Technique (ROSE) source
if(run.balanced.data){
dat_train_balanced_rose<-ROSE(label~., dat_train,seed=2020)$data
save(dat_train_balanced_rose, file="../output/balanced_data.RData")
}else{
  load(file="../output/balanced_data.RData")
}
table(dat_train_balanced_rose$label)
```

```
##
##    0    1
## 1223 1177
```

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

Call the train model and test model from library.

`train.R` and `test.R` should be wrappers for all your model training steps and your classification/prediction steps.

- `train.R`
  - Input: a data frame containing features and labels and a parameter list.
  - Output:a trained model
- `test.R`

  - Input: the fitted classification model using training data and processed features from testing images
  - Input: an R object that contains a trained classifier.
  - Output: training model specification

- In this Starter Code, we use logistic regression with LASSO penalty to do classification.

```
source("../lib/train.R")
source("../lib/test.R")
```

Model selection with cross-validation

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

```
# source("../lib/cross_validation.R")
# feature_train = as.matrix(dat_train[, -6007])
# label_train = as.integer(dat_train$label)
######## The code below has error!!!

# if(run.cv){
#   res_cv <- matrix(0, nrow = length(lmbd), ncol = 4)
#   for(i in 1:length(lmbd)){
#     cat("lambda = ", lmbd[i], "\n")
#     res_cv[i,] <- cv.function(features = feature_train, labels = label_train, K,
#                               l = lmbd[i], reweight = sample.reweight)
#   save(res_cv, file="../output/res_cv.RData")
#   }
# }else{
#   load("../output/res_cv.RData")
# }
```

Random Forest Cross Validation

```
# source("../lib/cross_validation_random_forest.R")
# feature_train_rf = as.matrix(dat_train_balanced_rose[, -6007])
# label_train_rf = as.integer(dat_train_balanced_rose$label)
#
# if(run.cv){
#   res_cv_rf <- matrix(0, nrow = length(lmbd), ncol = 4)
#   for(i in 1:length(lmbd)){
#     cat("lambda = ", lmbd[i], "\n")
#     res_cv_rf[i,] <- cv.function.rf(features = feature_train_rf, labels = label_train_rf, K, l = lmbd
#   save(res_cv_rf, file="../output/res_cv_rf.RData")
#   }
# }else{
#   load("../output/res_cv_rf.RData")
# }
```

Visualize cross-validation results.

```
# res_cv_rf  <- as.data.frame(res_cv_rf )
# colnames(res_cv_rf ) <- c("mean_error", "sd_error", "mean_AUC", "sd_AUC")
# res_cv_rf$k = as.factor(lmbd)
#
# if(run.cv){
#   p1 <- res_cv_rf  %>%
#     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 <- res_cv_rf  %>%
#     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)
# }
# lambda=0.01 is the best
```

- Choose the "best" parameter value

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

- 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 (sample.reweight){
#   tm_train <- system.time(fit_train <- train(feature_train, label_train, w = weight_train, par_best))
# } else {
#   tm_train <- system.time(fit_train <- train(feature_train, label_train, w = NULL, par_best))
# }
# save(fit_train, file="../output/fit_train.RData")
```

Advanced Models:

Create weight test

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

Random Forest:

Tune RF

```
source("../lib/random_forest.R")
if(run.cv){
time.rf.tune <- system.time(rf.tune <- random_forest_tune(dat_train_balanced_rose))
save(rf.tune, file="../output/rf_tune.RData")
}else(
  load("../output/rf_tune.RData")
)
```

```
## [1] "rf.tune"
```

```
rf.tune
```

```
##         mtry   OOBError
## 39.OOB    39 0.09833333
## 77.OOB    77 0.04791667
## 154.OOB  154 0.02958333
## 308.OOB  308 0.03166667
```

mtry = 154 is the best.

Find the best ntrees

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

#Train 500
if(run.cv){
time.rf.train <- system.time(random_forest_fit_500 <- random_forest_train_500(dat_train_balanced_rose,m
save(random_forest_fit_500, file = "../output/rf_train_500_trees.RData")
}
#Test 500
random_forest_test_prep=NA
if(run.cv){
 load(file="../output/rf_train_500_trees.RData")
 time.rf.test <- system.time(
   random_forest_test_prep <- random_forest_test(
     model = random_forest_fit_500,testset = dat_test)
               )

random_forest_test_prep <- as.numeric(as.character(random_forest_test_prep))
accu_rf_test <- mean(random_forest_test_prep == dat_test$label)
random_forest_label<-round(random_forest_test_prep)
accu_rf <- sum(weight_test * (random_forest_label == label_test)) / sum(weight_test)
#prob_pred <- lable_pred
tpr.fpr <- WeightedROC(random_forest_test_prep, label_test, weight_test)
auc_rf <- WeightedAUC(tpr.fpr)
cat("The AUC of model after reweighting: RF", "is", auc_rf, ".\n")
cat("The accuracy of model: Random Forest on imbalanced testing data", "is", accu_rf_test*100, "%.\n")
cat("The accuracy of model: Random Forest on balanced testing data", "is", accu_rf*100, "%.\n")
```

```r
cat("Time for training model Random Forest = ", time.rf.train[1], "s \n")
cat("Time for testing model Random Forest = ",time.rf.test[1], "s \n")
}
# The AUC of model after reweighting: RF is 0.5031999 .
# The accuracy of model: Random Forest on imbalanced testing data is 80.33333 %.
# The accuracy of model: Random Forest on balanced testing data is 50.31999 %.
# Time for training model Random Forest =  20.95 s
# Time for testing model Random Forest =  0.09 s

#Train 1000
if(run.cv){
time.rf.train <- system.time(random_forest_fit_1000 <- random_forest_train_1000(dat_train_balanced_rose
save(random_forest_fit_1000, file = "../output/rf_train_1000_trees.RData")
}
#Test 1000
random_forest_test_prep=NA
if(run.cv){
 load(file="../output/rf_train_1000_trees.RData")
 time.rf.test <- system.time(
   random_forest_test_prep <- random_forest_test(
     model = random_forest_fit_1000,testset = dat_test)
               )

random_forest_test_prep <- as.numeric(as.character(random_forest_test_prep))
accu_rf_test <- mean(random_forest_test_prep == dat_test$label)
random_forest_label<-round(random_forest_test_prep)
accu_rf <- sum(weight_test * (random_forest_label == label_test)) / sum(weight_test)
#prob_pred <- lable_pred
tpr.fpr <- WeightedROC(random_forest_test_prep, label_test, weight_test)
auc_rf <- WeightedAUC(tpr.fpr)
cat("The AUC of model after reweighting: RF", "is", auc_rf, ".\n")
cat("The accuracy of model: Random Forest on imbalanced testing data", "is", accu_rf_test*100, "%.\n")
cat("The accuracy of model: Random Forest on balanced testing data", "is", accu_rf*100, "%.\n")
cat("Time for training model Random Forest = ", time.rf.train[1], "s \n")
cat("Time for testing model Random Forest = ",time.rf.test[1], "s \n")
}

#Train 1500
if(run.cv){
time.rf.train <- system.time(random_forest_fit_1500 <- random_forest_train_1500(dat_train_balanced_rose
save(random_forest_fit_1500, file = "../output/rf_train_1500_trees.RData")
}
#Test 1500
random_forest_test_prep=NA
if(run.cv){
 load(file="../output/rf_train_1500_trees.RData")
 time.rf.test <- system.time(
   random_forest_test_prep <- random_forest_test(
     model = random_forest_fit_1500,testset = dat_test)
               )

random_forest_test_prep <- as.numeric(as.character(random_forest_test_prep))
accu_rf_test <- mean(random_forest_test_prep == dat_test$label)
```

```
random_forest_label<-round(random_forest_test_prep)
accu_rf <- sum(weight_test * (random_forest_label == label_test)) / sum(weight_test)
#prob_pred <- lable_pred
tpr.fpr <- WeightedROC(random_forest_test_prep, label_test, weight_test)
auc_rf <- WeightedAUC(tpr.fpr)
cat("The AUC of model after reweighting: RF", "is", auc_rf, ".\n")
cat("The accuracy of model: Random Forest on imbalanced testing data", "is", accu_rf_test*100, "%.\n")
cat("The accuracy of model: Random Forest on balanced testing data", "is", accu_rf*100, "%.\n")
cat("Time for training model Random Forest = ", time.rf.train[1], "s \n")
cat("Time for testing model Random Forest = ",time.rf.test[1], "s \n")
}

#Train 2000
if(run.cv){
time.rf.train <- system.time(random_forest_fit_2000 <- random_forest_train_2000(dat_train_balanced_rose
save(random_forest_fit_2000, file = "../output/rf_train_2000_trees.RData")
}
#Test 2000
random_forest_test_prep=NA
if(run.cv){
 load(file="../output/rf_train_2000_trees.RData")
 time.rf.test <- system.time(
   random_forest_test_prep <- random_forest_test(
     model = random_forest_fit_2000,testset = dat_test)
               )

random_forest_test_prep <- as.numeric(as.character(random_forest_test_prep))
accu_rf_test <- mean(random_forest_test_prep == dat_test$label)
random_forest_label<-round(random_forest_test_prep)
accu_rf <- sum(weight_test * (random_forest_label == label_test)) / sum(weight_test)
#prob_pred <- lable_pred
tpr.fpr <- WeightedROC(random_forest_test_prep, label_test, weight_test)
auc_rf <- WeightedAUC(tpr.fpr)
cat("The AUC of model after reweighting: RF", "is", auc_rf, ".\n")
cat("The accuracy of model: Random Forest on imbalanced testing data", "is", accu_rf_test*100, "%.\n")
cat("The accuracy of model: Random Forest on balanced testing data", "is", accu_rf*100, "%.\n")
cat("Time for training model Random Forest = ", time.rf.train[1], "s \n")
cat("Time for testing model Random Forest = ",time.rf.test[1], "s \n")
}
#Train 2500
if(run.cv){
time.rf.train <- system.time(random_forest_fit_2500 <- random_forest_train_2500(dat_train_balanced_rose
save(random_forest_fit_2500, file = "../output/rf_train_2500_trees.RData")
}
#Test 2500
random_forest_test_prep=NA
if(run.cv){
 load(file="../output/rf_train_2500_trees.RData")
 time.rf.test <- system.time(
   random_forest_test_prep <- random_forest_test(
     model = random_forest_fit_2500,testset = dat_test)
               )
random_forest_test_prep <- as.numeric(as.character(random_forest_test_prep))
accu_rf_test <- mean(random_forest_test_prep == dat_test$label)
random_forest_label<-round(random_forest_test_prep)
accu_rf <- sum(weight_test * (random_forest_label == label_test)) / sum(weight_test)
#prob_pred <- lable_pred
tpr.fpr <- WeightedROC(random_forest_test_prep, label_test, weight_test)
auc_rf <- WeightedAUC(tpr.fpr)
cat("The AUC of model after reweighting: RF", "is", auc_rf, ".\n")
cat("The accuracy of model: Random Forest on imbalanced testing data", "is", accu_rf_test*100, "%.\n")
cat("The accuracy of model: Random Forest on balanced testing data", "is", accu_rf*100, "%.\n")
cat("Time for training model Random Forest = ", time.rf.train[1], "s \n")
cat("Time for testing model Random Forest = ",time.rf.test[1], "s \n")
}
```

Testing Result: When trees = 500: The AUC of model after reweighting: RF is 0.5116745 . The accuracy of model: Random Forest on imbalanced testing data is 80.66667 %. The accuracy of model: Random Forest on balanced testing data is 51.16745 %. Time for training model Random Forest = 713.63 s Time for testing model Random Forest = 0.19 s

When trees = 1000 The AUC of model after reweighting: RF is 0.5201491 . The accuracy of model: Random Forest on imbalanced testing data is 81 %. The accuracy of model: Random Forest on balanced testing data is 52.01491 %. Time for training model Random Forest = 1367.94 s Time for testing model Random Forest = 0.28 s

When trees = 1500 The AUC of model after reweighting: RF is 0.5201491 . The accuracy of model: Random Forest on imbalanced testing data is 81 %. The accuracy of model: Random Forest on balanced testing data is 52.01491 %. Time for training model Random Forest = 2077.56 s Time for testing model Random Forest = 0.36 s

When trees = 2000 The AUC of model after reweighting: RF is 0.5201491 . The accuracy of model: Random Forest on imbalanced testing data is 81 %. The accuracy of model: Random Forest on balanced testing data is 52.01491 %. Time for training model Random Forest = 3142.77 s Time for testing model Random Forest = 0.56 s

When trees = 2500 The AUC of model after reweighting: RF is 0.5159118 . The accuracy of model: Random Forest on imbalanced testing data is 80.83333 %. The accuracy of model: Random Forest on balanced testing data is 51.59118 %. Time for training model Random Forest = 3963.67 s Time for testing model Random Forest = 0.62 s

Therefore, we should use trees = 1000.

Train RF with tuning parameters:

```
source("../lib/random_forest.R")
if(train.random.forest){
  time.rf.train <- system.time(random_forest_fit <- random_forest_train(dat_train_balanced_rose,mtry = 
  save(random_forest_fit, file = "../output/random_forest_train.RData")
  save(time.rf.train,file = "../output/random_forest_train_time.RData")
}else{
  load(file = "../output/random_forest_train_time.RData")
  load(file = "../output/random_forest_train.RData")
}
```

Test RF with tuning parameters

```r
random_forest_test_prep=NA
if(run.test){
 load(file="../output/random_forest_train.RData")
 time.rf.test <- system.time(
   random_forest_test_prep <- random_forest_test(
     model = random_forest_fit,testset = dat_test)
               )
}
random_forest_test_prep <- as.numeric(as.character(random_forest_test_prep))
accu_rf_test <- mean(random_forest_test_prep == dat_test$label)
```

Calculate weightedAUC on testing split

```r
random_forest_label<-round(random_forest_test_prep)
#prob_pred <- lable_pred
tpr.fpr <- WeightedROC(random_forest_test_prep, label_test, weight_test)
auc_rf <- WeightedAUC(tpr.fpr)
```

Summary of RF

```r
cat("The AUC of model after reweighting: RF", "is", auc_rf, ".\n")
```

```
## The AUC of model after reweighting: RF is 0.5217069 .
```

```r
cat("The accuracy of model: Random Forest on testing data", "is", accu_rf_test*100, "%.\n")
```

```
## The accuracy of model: Random Forest on testing data is 82.33333 %.
```

```r
cat("Time for training model Random Forest = ", time.rf.train[1], "s \n")
```

```
## Time for training model Random Forest =  1567.63 s
```

```r
cat("Time for testing model Random Forest = ",time.rf.test[1], "s \n")
```

```
## Time for testing model Random Forest =  0.3 s
```

```r
#label_test
```

Step 5: Run test on test images

```
# tm_test = NA
# feature_test <- as.matrix(dat_test[, -6007])
# if(run.test){
#   load(file="../output/fit_train.RData")
#   tm_test <- system.time({label_pred <- as.integer(test(fit_train, feature_test, pred.type = 'class')
#                           prob_pred <- test(fit_train, feature_test, pred.type = 'response')})
# }
```

- evaluation

```
## reweight the test data to represent a balanced label distribution
# label_test <- as.integer(dat_test$label)
# weight_test <- rep(NA, length(label_test))
# for (v in unique(label_test)){
#   weight_test[label_test == v] = 0.5 * length(label_test) / length(label_test[label_test == v])
# }
#
# accu <- sum(weight_test * (label_pred == label_test)) / sum(weight_test)
# tpr.fpr <- WeightedROC(prob_pred, label_test, weight_test)
# auc <- WeightedAUC(tpr.fpr)
#
#
# cat("The accuracy of model:", model_labels[which.min(res_cv$mean_error)], "is", accu*100, "%.\n")
# cat("The AUC of model:", model_labels[which.min(res_cv$mean_error)], "is", auc, ".\n")
#
```

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

###Reference - Du, S., Tao, Y., & Martinez, A. M. (2014). Compound facial expressions of emotion. Proceedings of the National Academy of Sciences, 111(15), E1454-E1462.