Main-Random Forest Model for imbalanced binary classification

Weiwei Song

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.

```
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")
}

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

if(!require("randomForest")){
```

```
  install.packages("randomForest")
}
if(!require("grid")){
  install.packages("grid")
}
if(!require("gridExtra")){
  install.packages("gridExtra")
}
if(!require("ranger")){
  install.packages("ranger")
}
library(grid)
library(gridExtra)
library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(glmnet)
library(WeightedROC)
library(gbm)
library(randomForest)
library(ranger)
library(ROSE)
library(rpart)
library(ROCR)
library(pROC)
```

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.

```
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

```
K <- 5  # number of CV folds

run.fudicial.list <- TRUE

run.feature.train <- TRUE # process features for training set
run.feature.test <- TRUE # process features for test set
sample.reweight <- TRUE # run sample reweighting in model training

run.cv.WRF<-FALSE
run.train.WRF<-FALSE
run.test.WRF<-FALSE

run.cv.BRF<-FALSE
run.train.BRF<-TRUE
run.test.BRF<-TRUE
```

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

```
##Random Forest
hyper_grid_RF<- expand.grid(
 ntree=seq(100,300,by=50),
 mtry=seq(39,234,by=39)# start from 1/2sqrt(q) to 3sqrt(q)
)
dim(hyper_grid_RF)
```

```
## [1] 30  2
```

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"))
}
```

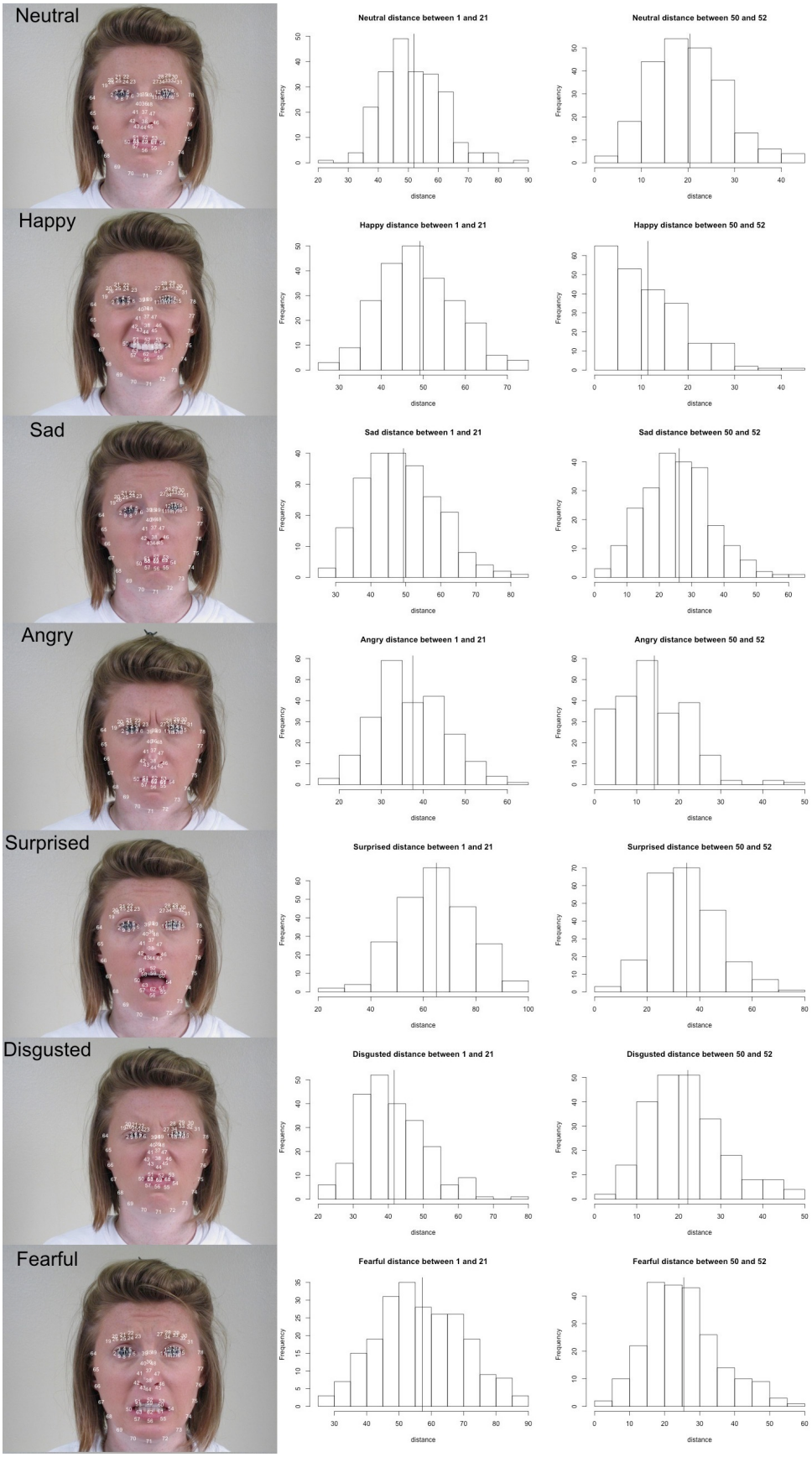
Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

```
#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.



`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")
}
```

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

i. Random Forest

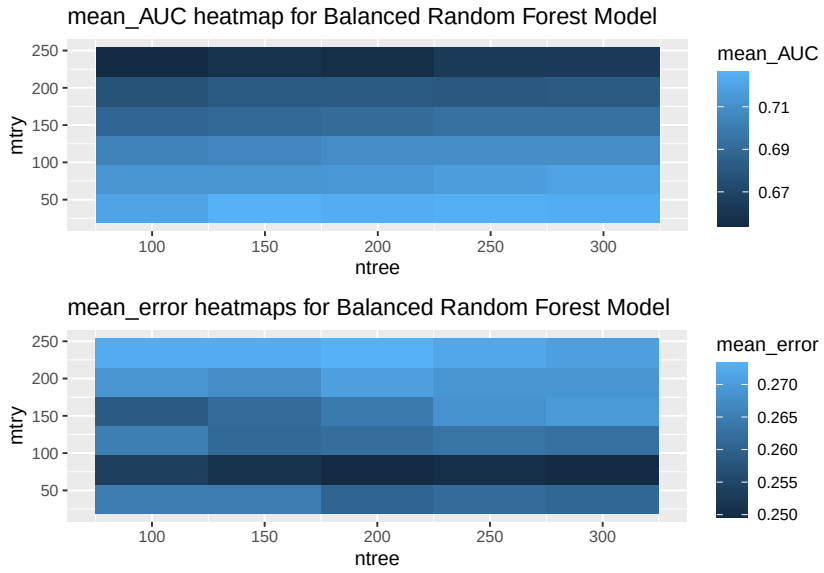
- `train_RF.R`
  - Input: a data frame containing features and labels and a parameter list.
  - Output:a trained model
- `test_RF.R`
  - Input: the fitted classification model using training data and processed features from testing images
  - Input: an R object that contains a trained classifier.
  - Output: training model specification
- Do Random forest tuning parameter searching using three parameters:
  - n.tree: number of trees. want enough trees to stabilize the error but using too many trees is unnecessarily inefficient
  - mtry: the number of variables to randomly sample as candidates at each split.
  - nodesize: minimum number of samples within the terminal nodes.Controls the complexity of the trees. Bigger node size allows for deeper, more complex trees and smaller node results in shadow trees. Bias-varaince trade-off because the deeper trees introduce more varaince, and shallower trees introduce more bias.

```
source("../lib/train_BRF.R")
source("../lib/test_BRF.R")
source("../lib/cross_validation_BRF.R")
```

###Balanced Random Forest

```
feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)
if(run.cv.BRF){
  res_cv_BRF <- matrix(0, nrow = nrow(hyper_grid_RF), ncol = 4)
  for(i in 1:nrow(hyper_grid_RF)){
```

```
    cat("n_tree =", hyper_grid_RF$ntree[i],
        ", mtry=", hyper_grid_RF$mtry[i],"\n")
    res_cv_BRF[i,] <- cv.function(features = feature_train,
                                  labels = label_train, K,
                                  num_tree=hyper_grid_RF$ntree[i],
                                  mtry=hyper_grid_RF$mtry[i])
  save(res_cv_BRF, file="../output/res_cv_BRF.RData")
  }
}else{
  load("../output/res_cv_BRF.RData")
}
```



- Choose the "best" parameter value

```
par_best_BRF <-hyper_grid_RF[which.max(result_cv_BRF$mean_AUC),]
par_best_ntree<- par_best_BRF$ntree
par_best_mtry<- par_best_BRF$mtry
```

```
if (run.train.BRF) {
    tm_train_BRF <- system.time({
      dat_train.ROSE<- ROSE(label~.,data=dat_train)$data
      feature_train<-dat_train.ROSE[,-6007]
      label_train<- dat_train.ROSE[,6007]
      fit_train_BRF <- train(features = feature_train, labels = label_train,
                             num_tree = par_best_ntree,
                             mtry = par_best_mtry)})

  save(fit_train_BRF, tm_train_BRF, file="../output/fit_train_BRF.RData")
}else {
  load(file="../output/fit_train_BRF.RData")
}
```

```
tm_test_BRF = NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test.BRF){
  load(file="../output/fit_train_BRF.RData")
  tm_test_BRF <- system.time({ prob_pred <- test(fit_train_BRF, feature_test)$predictions
```

```
                              label_pred <- ifelse(prob_pred[,1]>prob_pred[,2],1,2)})
}
```

```
label_test <- as.integer(dat_test$label)
weight_test<-rep(1, length(label_test))
accu <- sum(label_pred == label_test)/length(label_test)
confusionMatrix(factor(label_pred),factor(label_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 440  62
##          2  51  47
##
##                Accuracy : 0.8117
##                  95% CI : (0.778, 0.8422)
##     No Information Rate : 0.8183
##     P-Value [Acc > NIR] : 0.6862
##
##                   Kappa : 0.3407
##
##  Mcnemar's Test P-Value : 0.3468
##
##             Sensitivity : 0.8961
##             Specificity : 0.4312
##          Pos Pred Value : 0.8765
##          Neg Pred Value : 0.4796
##              Prevalence : 0.8183
##          Detection Rate : 0.7333
##    Detection Prevalence : 0.8367
##       Balanced Accuracy : 0.6637
##
##        'Positive' Class : 1
##
```

```
weight_test<-rep(1,length(label_test))

 tpr.fpr <- WeightedROC(prob_pred[,2],label_test,weight_test)
auc <- WeightedAUC(tpr.fpr)
cat("The accuracy of BRF model:", "ntree=", par_best_ntree ,", mtry=", par_best_mtry, "is", accu*100, "
```

```
## The accuracy of BRF model: ntree= 150 , mtry= 39 is 81.16667 %.
```

```
cat("The AUC of BRF model:", "ntree=", par_best_ntree ,", mtry=", par_best_mtry, "is", auc, ".\n")
```

```
## The AUC of BRF model: ntree= 150 , mtry= 39 is 0.7570582 .
```

```
cat("Time for training Balanced Random Forest model=", tm_train_BRF[1], "s \n")
```

```
## Time for training Balanced Random Forest model= 138.146 s
```

```
cat("Time for testing Balanced Random Forest model=", tm_test_BRF[1], "s \n")
```

```
## Time for testing Balanced Random Forest model= 0.651 s
```

```
source("../lib/train_WRF.R")
source("../lib/test_WRF.R")
source("../lib/cross_validation_WRF.R")
```

Model selection with cross-validation

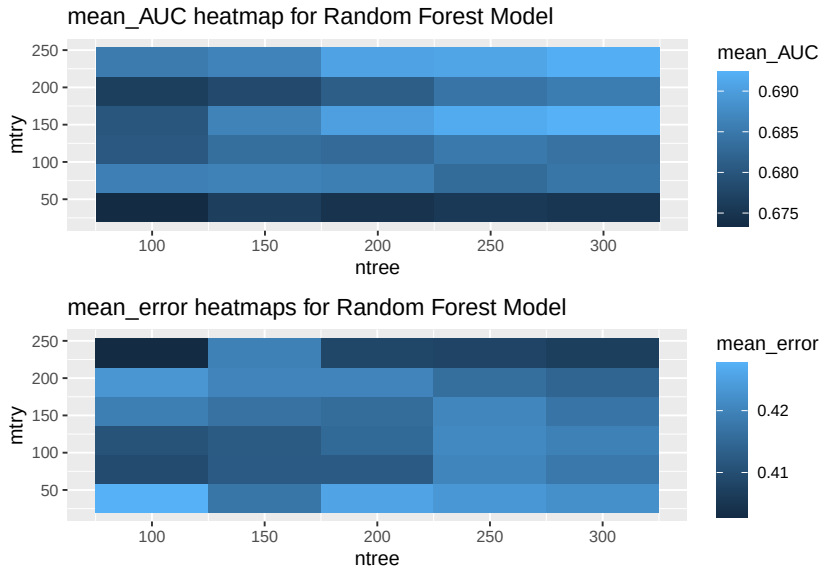
- Do model selection by choosing among different values of training model parameters.

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

if(run.cv.WRF){
  res_cv_WRF <- matrix(0, nrow = nrow(hyper_grid_RF), ncol = 4)
  for(i in 1:nrow(hyper_grid_RF)){
    cat("n_tree =", hyper_grid_RF$ntree[i],
        ", mtry=", hyper_grid_RF$mtry[i],"\n")
    res_cv_WRF[i,] <- cv.function(features = feature_train,
                                  labels = label_train, K,
                                  num_tree=hyper_grid_RF$ntree[i],
                                  mtry=hyper_grid_RF$mtry[i],
                                  reweight = sample.reweight)
  save(res_cv_WRF, file="../output/res_cv_WRF.RData")
  }
}else{
  load("../output/res_cv_WRF.RData")
}
# trial<-ranger(label_train~.,
#        data=data.frame(cbind(label_train,feature_train)),
#        num.trees=100,
#        mtry=39,
#         classification=TRUE,
#                   probability = T,
#                   verbose = F,
#                   write.forest = T,
#                   regularization.factor = 1,
#                   importance = "none",
#                   min.node.size = 10)
# prob_pred<-predict(trial,feature_test)$predictions
# label_pred <- ifelse(prob_pred[,1]>=prob_pred[,2],1,2)
# table(label_pred)
# nrow(prob_pred)
# length(label_test)
# cv.error <- 1 - sum(weight_test*(label_pred == label_test))/sum(weight_test)
# tpr.fpr <- WeightedROC(prob_pred[,2],label_test,weight_test)
# cv.AUC <- WeightedAUC(tpr.fpr)
#
#
# trial1<-ranger(label_train~.,
#        data=data.frame(cbind(label_train,feature_train)),
#        num.trees=100,
#        mtry=39,
#         classification=TRUE,
#                   verbose = F,
#                   write.forest = T,
```

```
#                   regularization.factor = 1,
#                   importance = "none",
#                   min.node.size = 10)
# ?ranger()
# labels_pred3<-predict(trial1,feature_test,type="response")$predictions
# table(labels_pred3)
```

Visualize cross-validation results.



- Choose the "best" parameter value

```
par_best_WRF <-hyper_grid_RF[which.max(result_cv_WRF$mean_AUC),]
par_best_ntree<- par_best_WRF$ntree
par_best_mtry<- par_best_WRF$mtry
```

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

```
#training weights
if (run.train.WRF) {
  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_WRF <- system.time(
      fit_train_WRF <- train(features = feature_train, labels = label_train,
                             w =  weight_train,
                             num_tree = par_best_ntree,
                             mtry = par_best_mtry))
  }else{
    tm_train_WRF <- system.time(
      fit_train_WRF <- train(features = feature_train, labels = label_train,
                             w =  NULL,
                            num_tree = par_best_ntree,
```

```
                                  mtry = par_best_mtry))
  }
  save(fit_train_WRF, tm_train_WRF, file="../output/fit_train_WRF.RData")
}else {
  load(file="../output/fit_train_WRF.RData")
}
```

Step 5: Run test on test images

```
 tm_test_WRF = NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test.WRF){
  load(file="../output/fit_train_WRF.RData")
  tm_test_WRF <- system.time({ prob_pred <- test(fit_train_WRF, feature_test)$predictions
                               label_pred <- ifelse(prob_pred[,1]>prob_pred[,2],1,2)})
}
```

- 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[,2], label_test, weight_test)
 auc <- WeightedAUC(tpr.fpr)
 cat("The accuracy of WRF model:", "ntree=", par_best_ntree ,", mtry=", par_best_mtry, "is", accu*100,
```

```
## The accuracy of WRF model: ntree= 300 , mtry= 156 is 66.36615 %.
```

```
 cat("The AUC of WRF model:", "ntree=", par_best_ntree ,", mtry=", par_best_mtry, "is", auc, ".\n")
```

```
## The AUC of WRF model: ntree= 300 , mtry= 156 is 0.7570582 .
```

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 training Weighed Random Forest model=", tm_train_WRF[1], "s \n")
```

```
## Time for training Weighed Random Forest model= 38.715 s
```

```
cat("Time for testing Weighted Random Forest model=", tm_test_WRF[1], "s \n")
```

```
## Time for testing Weighted Random Forest model= NA s
```

###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.