Main

Daizy Lam, Peter Kwauk, Qizhen Yang, Ellen Chen, Daryl Kow

In this project, we tested different classification model for facial emotion recognition. Our group tried out six different machine learning algorithms, trained them on the given data set, cross-validated to find the optimized parameters, and provided fair evaluation for all the models. The evaluation process considered the prediction error (and accuracy), the area under the ROC curve (or AUC), and running time (testing and training) to choose the most improved model. In this main file, we only show the results of baseline model(GBM) and the final improved model(XGboost). As for the other models, results are in working_main file.

```r
if(!require("EBImage")){
  install.packages("BiocManager")
  BiocManager::install("EBImage")
}
```

```
## Warning: package 'EBImage' was built under R version 4.0.3
```

```r
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")
}
```

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

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

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

library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(glmnet)
library(WeightedROC)
library(e1071)
library(xgboost)
library(randomForest)
```

Step 0 set work directories

```r
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
run.cv.baseline <- FALSE # run cross-validation on the gbm baseline
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.test <- TRUE # run evaluation on an independent test set
run.feature.test <- TRUE # process features for test set
run.cv.xgboost <- FALSE
```

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

```
#gbm parameters tuning:
n.trees = c(10,50,100,200)
shrinkage = c(0.01,0.05,0.1,0.15)

#xgboost parameters tuning
n_iterations <- 100
```

Step 2: import data and train-test split

```
set.seed(2020)
#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

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

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

```
## Loading required package: gbm
```

```
## Loaded gbm 2.1.8
```

```
source("../lib/test_gbm.R")
source("../lib/fit_train_xgboost.R")
```

Model selection with cross-validation

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

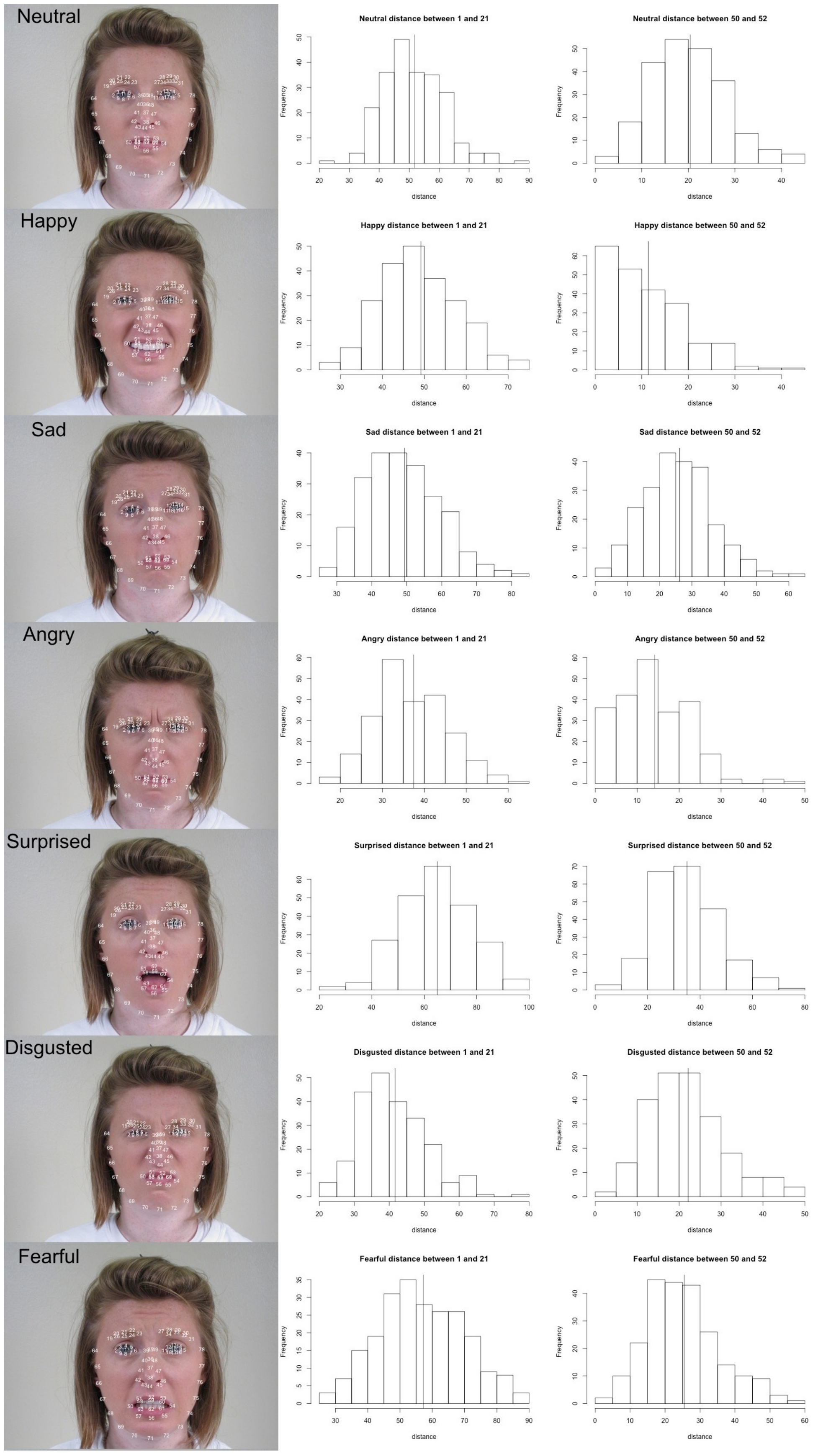

Figure 1: Figure1

Baseline Model

- Baseline/GBM

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

source("../lib/cross_validation.R")
source("../lib/cross_validation_SVM.R")
source("../lib/cv_gbm.R")

if(run.cv.baseline){

  mean_error_cv <- matrix(0, nrow = length(n.trees), ncol = length(shrinkage))
  sd_error_cv <- matrix(0, nrow = length(n.trees), ncol = length(shrinkage))
  mean_auc_cv <- matrix(0, nrow = length(n.trees), ncol = length(shrinkage))
  sd_auc_cv <- matrix(0, nrow = length(n.trees), ncol = length(shrinkage))

  for(i in 1:length(n.trees)){
    cat("n.trees =", n.trees[i],"\n")
  for(k in 1:length(shrinkage)){
    cat("shrinkage =", shrinkage[k],"\n")

res_cv_gbm <- cv_gbm(features = feature_train, labels = label_train, K,  n.trees = n.trees[i],shrinkage

    mean_error_cv[i,k]<-res_cv_gbm[1]
     sd_error_cv[i,k]<-res_cv_gbm[2]
      mean_auc_cv[i,k]<-res_cv_gbm[3]
        sd_auc_cv[i,k]<-res_cv_gbm[4]

    save(mean_error_cv, file="../output/mean_error_cv.RData")
    save(sd_error_cv, file="../output/sd_error_cv.RData")
    save(mean_auc_cv, file="../output/mean_auc_cv.RData")
    save(sd_auc_cv, file="../output/sd_auc_cv.RData")
    }}
 } else{
  load("../output/mean_error_cv.RData")
  load("../output/sd_error_cv.RData")
  load("../output/mean_auc_cv.RData")
  load("../output/sd_auc_cv.RData")
    }
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
df_mean_error=data.frame(mean_error_cv)%>%
setNames(shrinkage)%>%
mutate(n.trees=n.trees)%>%
gather(shrinkage,mean_error,`0.01`:`0.15`)
```

```
df_sd_error=data.frame(sd_error_cv)%>%
setNames(shrinkage)%>%
mutate(n.trees=n.trees)%>%
gather(shrinkage,sd_error,‘0.01‘:‘0.15‘)

df_mean_auc=data.frame(mean_auc_cv)%>%
setNames(shrinkage)%>%
mutate(n.trees=n.trees)%>%
gather(shrinkage,mean_auc,‘0.01‘:‘0.15‘)

df_sd_auc=data.frame(sd_auc_cv)%>%
setNames(shrinkage)%>%
mutate(n.trees=n.trees)%>%
gather(shrinkage,sd_auc,‘0.01‘:‘0.15‘)

res_cv_gbm <- df_mean_error%>%mutate(sd_error=df_sd_error$sd_error,
                                     mean_auc=df_mean_auc$mean_auc,
                                     sd_auc=df_sd_auc$sd_auc)
save(res_cv_gbm,file = "../output/res_cv_gbm.RData")
```

Visualize cross-validation results.

```
load("../output/res_cv_gbm.RData")


if(run.cv.baseline){
  p1 <- res_cv_gbm %>%
    ggplot(aes(x = n.trees, y = mean_error,
               ymin = mean_error - sd_error, ymax = mean_error  +sd_error)) +
    geom_crossbar() +
    facet_wrap(~shrinkage) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

  p2 <- res_cv_gbm %>%
    ggplot(aes(x = n.trees, y = mean_auc,
               ymin = mean_auc - sd_auc, ymax = mean_auc + sd_auc)) +     facet_wrap(~shrinkage) +
    geom_crossbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

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

- Choose the "best" parameter value for baseline model

```
best_n.trees = as.numeric(res_cv_gbm[which.min(res_cv_gbm$mean_error),1])
best_shrinkage = as.numeric(res_cv_gbm[which.min(res_cv_gbm$mean_error),2])
```

Improved Model

- XGBoost

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

best_param = list()
```

```r
lowest_error <- Inf

if(run.cv.xgboost){
  feature_train = as.matrix(dat_train[, -6007])
  label_train = as.integer(dat_train$label)
  label_train_xgb <- label_train
  label_train_xgb[label_train_xgb == 2] <- 0
  K <- 5
  set_rounds <- 50
  for(j in 1: n_iterations){
    params <- list(booster = "gbtree", objective = "binary:logistic",
                  max_depth = sample(6:10, 1),eta = runif(1, .01, .3),
                  gamma = runif(1, 0.0, 0.2),
                  subsample = runif(1, .6, .9),
                  colsample_bytree = runif(1, .5, .8),
                  min_child_weight = sample(1:40, 1),
                  max_delta_step = sample(1:10, 1))
    xgb_cv <- cv_xgboost(init_params = params, features = feature_train, labels = label_train_xgb,
                        rounds = set_rounds, K = K)
    min_error <- min(xgb_cv$evaluation_log$test_error_mean)

    if(min_error < lowest_error){
      lowest_error <- min_error
      best_param <- xgb_cv$params[c(1, 3:11)]
    }
  }
  new_params <- best_param

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

Train models

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

```r
# 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_baseline <- system.time(fit_train_baseline <- train_gbm(feature_train,
                                   label_train,
                                   w = weight_train,
                                   best_n.trees,
                                   best_shrinkage))
} else {
  tm_train_baseline <- system.time(fit_train_baseline <- train_gbm(feature_train,
                                   label_train,
                                   w = NULL,
                                   best_n.trees,
                                    best_shrinkage))
}
save(fit_train_baseline, file="../output/fit_train_baseline.RData")
```

- Train the XGBoost model(Improved model) with optimal parameters

```
load("../output/res_cv_xgboost.RData")
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])
}

feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)
label_train_xgb <- label_train
label_train_xgb[label_train_xgb == 2] <- 0
set_rounds  <- 50

if (sample.reweight){
 xgb_train_time <- system.time(fit_train_xgb <- xgboost_train(features = feature_train,
                                                              labels = label_train_xgb,
                                                              params = new_params,
                                                              rounds =  set_rounds,
                                                      spec_weights = weight_train))
} else {
  xgb_train_time <- system.time(fit_train_xgb <- xgboost_train(features = feature_train, labels = label_
}
```

```
## [13:45:57] WARNING: amalgamation/../src/learner.cc:541:
## Parameters: { early_stop_round } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
##
##
## [1]  train-error:0.281575
## [2]  train-error:0.190297
## [3]  train-error:0.168283
## [4]  train-error:0.147509
## [5]  train-error:0.135642
## [6]  train-error:0.128067
## [7]  train-error:0.120428
## [8]  train-error:0.096280
## [9]  train-error:0.088769
## [10] train-error:0.082965
## [11] train-error:0.082861
## [12] train-error:0.068771
## [13] train-error:0.060821
## [14] train-error:0.052948
## [15] train-error:0.049290
## [16] train-error:0.046110
## [17] train-error:0.041340
## [18] train-error:0.040164
## [19] train-error:0.035135
## [20] train-error:0.031438
## [21] train-error:0.030960
## [22] train-error:0.028077
## [23] train-error:0.023565
## [24] train-error:0.023565
## [25] train-error:0.021756
## [26] train-error:0.020204
## [27] train-error:0.019170
## [28] train-error:0.017619
## [29] train-error:0.017102
## [30] train-error:0.016326
## [31] train-error:0.015034
## [32] train-error:0.014517
## [33] train-error:0.014517
## [34] train-error:0.012966
## [35] train-error:0.012707
## [36] train-error:0.012190
## [37] train-error:0.010897
## [38] train-error:0.009824
## [39] train-error:0.009307
## [40] train-error:0.008273
## [41] train-error:0.008273
## [42] train-error:0.008790
## [43] train-error:0.008532
## [44] train-error:0.007497
## [45] train-error:0.007239
## [46] train-error:0.007497
## [47] train-error:0.006722
## [48] train-error:0.006980
## [49] train-error:0.006205
## [50] train-error:0.006463
```

```
save(fit_train_xgb, file="../output/fit_train_xgb.RData")
```

Step 5: Run test on test images

*Baseline model

```
tm_test_baseline = NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test){
  load(file="../output/fit_train_baseline.RData")
  tm_test_baseline <- system.time({label_pred_baseline <- as.integer(test_gbm(fit_train_baseline,featur
                          prob_pred_baseline <- test_gbm(fit_train_baseline, feature_test,best_n.trees,
}
```

*XGBoost (Improved model)

```
tm_test_xgb = NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test){
  load(file="../output/fit_train_xgb.RData")
  tm_test_xgb <- system.time({label_pred_xgb <- predict(fit_train_xgb, feature_test, pred.type = 'class
                          label_pred_xgb[label_pred_xgb >= 0.5] <- 1;
                          label_pred_xgb[label_pred_xgb < 0.5] <-  0;
                          prob_pred_xgb <- predict(fit_train_xgb, feature_test, pred.type = 'response'))
}
```

Evaluation

*Baseline Model

```
## 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_baseline <- mean(label_pred_baseline == label_test)
tpr.fpr.baseline <- WeightedROC(prob_pred_baseline, label_test, weight_test)
auc_baseline <- WeightedAUC(tpr.fpr.baseline)


cat("The accuracy of model GBM: with n.trees=",best_n.trees,"and shrinkage =", best_shrinkage, "is", acc
```

```
## The accuracy of model GBM: with n.trees= 100 and shrinkage = 0.05 is 78.16667 %.
```

```
cat("The AUC of model GBM: with n.trees=", best_n.trees,"and shrinkage =", best_shrinkage, "is", auc_bas
```

```
## The AUC of model GBM: with n.trees= 100 and shrinkage = 0.05 is 0.7322261 .
```

*Improved Model

```
label_test <- as.integer(dat_test$label)
label_test_xgb <- label_test
label_test_xgb[label_test_xgb==2] = 0

weight_test <- rep(NA, length(label_test_xgb))
for (v in unique(label_test_xgb)){
  weight_test[label_test_xgb == v] = 0.5 * length(label_test_xgb) / length(label_test_xgb[label_test_xgb
}

accu_xgb <- mean((label_pred_xgb == label_test_xgb))
tpr.fpr_xgb <- WeightedROC(prob_pred_xgb, label_test_xgb, weight_test)
auc_xgb <- WeightedAUC(tpr.fpr_xgb)


cat("The accuracy of the XGBoost model:", "is", accu_xgb*100, "%.\n")
```

```
## The accuracy of the XGBoost model: is 80.83333 %.
```

```
cat("The AUC of the XGBoost model:", "is", auc_xgb, ".\n")
```

```
## The AUC of the XGBoost model: is 0.8119496 .
```

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.

*Baseline Model

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

```
## Time for constructing training features= 1.198 s
```

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

```
## Time for constructing testing features= 0.22 s
```

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

```
## Time for training model= 84.262 s
```

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

```
## Time for testing model= 0.383 s
```

- Improved Model

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

```
## Time for constructing training features= 1.198 s
```

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

```
## Time for constructing testing features= 0.22 s
```

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

```
## Time for training model= 50.402 s
```

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

```
## Time for testing model= 0.31 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.
- Oshiro, T. & Perez, P. & Baranauskas, J. (2012). How Many Trees in a Random Forest?. Lecture notes in computer science. 7376. 10.1007/978-3-642-31537-4_13.