Project 3: Facial Expression Predictive Modeling

Jingbin Cao, Chuanchuan Liu, Dennis Shpits, Yingyao Wu, Zikun Zhuang

```
#Test Branch created
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("DMwR")){
  install.packages("DMwR")
}
if(!require("OpenImageR")){
 install.packages("OpenImageR")
}
if(!require("AUC")){
 install.packages("AUC")
}
if(!require("e1071")){
 install.packages("e1071")
}
if(!require("randomForest")){
 install.packages("randomForest")
}
if(!require("xgboost")){
 install.packages("xgboost")
}
if(!require("tibble")){
 install.packages("tibble")
}
if(!require("ROSE")){
 install.packages("ROSE")
}
if(!require("tidyverse")){
 install.packages("tidyverse")
}
if(!require("caTools")){
  install.packages("caTools")
}
if(!require("prediction")){
  install.packages("prediction")
}
if(!require("pROC")){
  install.packages("pROC")
}

library(R.matlab)
library(readxl)
library(dplyr)
library(ggplot2)
library(caret)
library(glmnet)
library(WeightedROC)
library(gbm)
library(DMwR)
### new libraries
library(OpenImageR)
library(AUC)
library(e1071)
library(randomForest)
library(xgboost)
library(tibble)
library(ROSE)
library(tidyverse)
library(caTools)
library(prediction)
library(pROC)
```

Step 0 set work directories

```
set.seed(2020)
```

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) run improved gbm model
- (number) gbm.numtrees, the number of trees to use in GBM baseline
- (T/F) return polynomial features matrix only
- (T/F) add polynomial features to starter code features matrix
- (T/F) run svm model
- (T/F) perform model selection over a list of svm models
- (T/F) run evaluation on the test set
- (T/F) run random forest model
- (T/F) rebalance training set
- (T/F) train random forest model
- (T/F) tune hyperparameters for random forest model
- (T/F) run ridge model
- (0/1) alpha, alpha=0 for ridge regression, alpha=1 for lasso regression
- (T/F) train ridge model
- (T/F) run PCA+LDA model
- (T/F) run different principal components
- (T/F) run LDA on training set
- (T/F) run evaluation on the test set

```r
run.cv <- TRUE # run cross-validation on the training set
sample.reweight <- FALSE # 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

# gbm
run.gbm <- FALSE # gbm(imroved) is the chosen advanced model
gbm.numtrees <- 1000 #number of trees to use in gbm
run.poly.feature <- TRUE # process poly features
run.add.poly.feature <- TRUE # and poly features to dist matrix

# svm
run.svm <- FALSE # svm is the chosen advanced model
```

```r
model.selection <- TRUE # perform model selection on svm models
run.svm.test <- TRUE # evaluate performance on the test set

# random forest
run.rf <- FALSE # random forest is the chosen advanced model
run.balanced.data <- TRUE # whether or not balance the data
train.random.forest <- FALSE # train Random Forest Model
tune.random.forest <- FALSE # tune Random Forest Model

# ridge
run.ridge <- FALSE # ridge is the chosen advanced model
alpha <- 0 # ridge regression
train.ridge <- TRUE # train ridge model

# PCA + LDA
run.pca_lda <- FALSE # PCA + LDA is the chosen adcanced model
run.select_PC <- TRUE #run different PCs
run.lda <- TRUE # run lda on the training set
run.pca_lad.test <- TRUE # evaluate performance on the test set
```

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

Step 2: import data and train-test split

```r
#train-test split
info <- read.csv(train_label_path)
n <- nrow(info) #get number of rows from csv
n_train <- round(n*(4/5), 0) #use 4/5 amount of data for training
train_idx <- sample(info$Index, n_train, replace = F) #grab indexes used for training
test_idx <- setdiff(info$Index, train_idx) # get indexes not used for training
```

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
n_files <- length(list.files(train_image_dir,'*jpg'))
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` is the wrapper for all feature engineering functions and options. The function `feature( )` have options that correspond to different scenarios for the project and produces an R object that contains features and responses that are required by all the models that are going to be evaluated 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, run.poly.feature, run.a
  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, run.poly.feature, run.a
  save(dat_test, file="../output/feature_test.RData")
}else{
  load(file="../output/feature_test.RData")
}
```

Step 4: train classification models with training features and responses; run test on test images

Call the train model and test model from library.

`train.R` and `test.R` are wrappers for all model training steps and 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")
```

Baseline Model

——THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL.———-

Advanced Model 1: Improved GBM Model

- Model Training

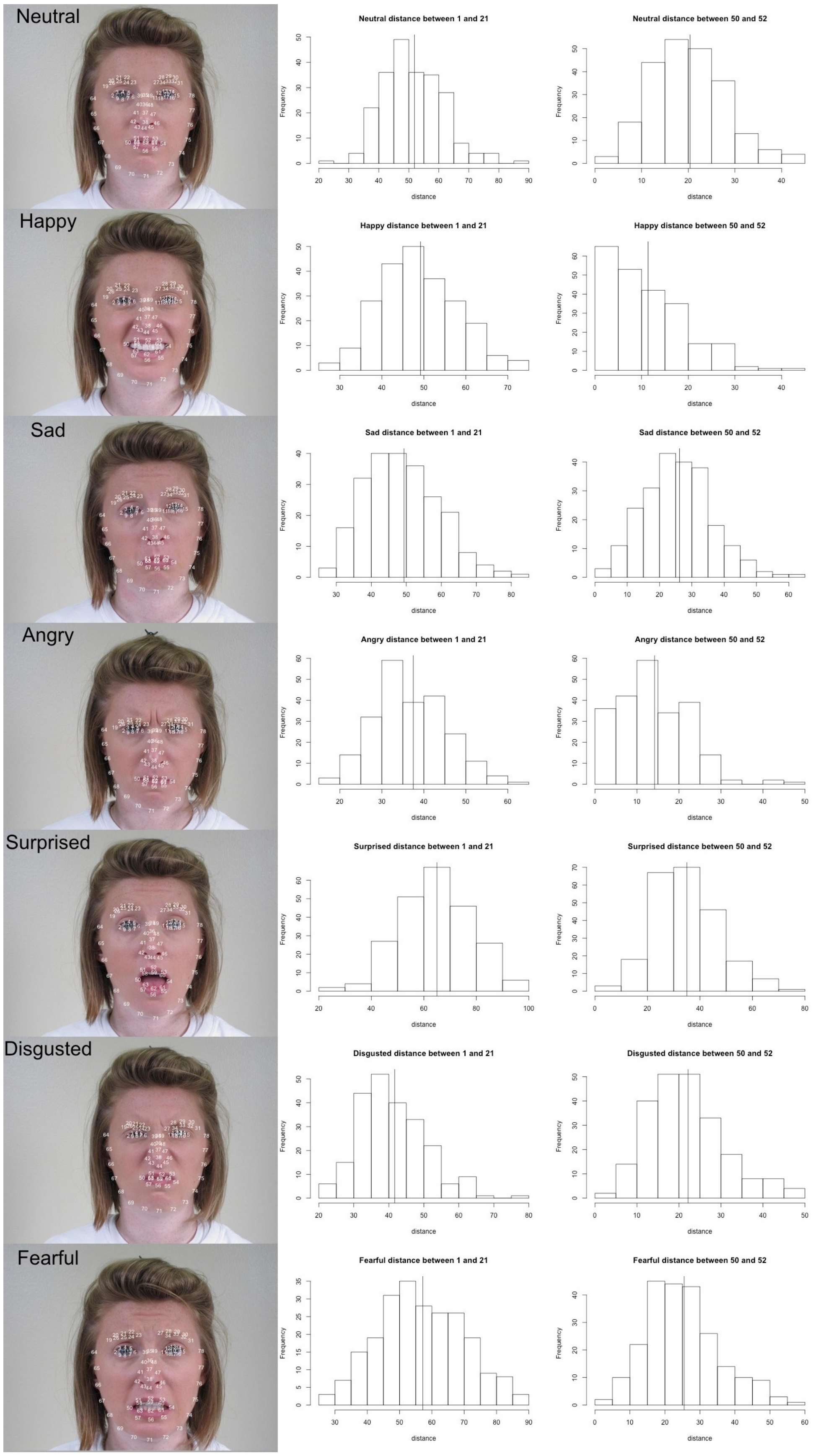

Figure 1: Figure1

```r
if (run.gbm){
  if (sample.reweight){
    tm_train <- system.time(fit_train <- train_gbm(dat_train, s=0.1, K=K, n=gbm.numtrees,w = weight_trai
  } else {
    tm_train <- system.time(fit_train <- train_gbm(dat_train, s=0.1, K=K, n=gbm.numtrees,w = NULL))
  }

  # plot the performance
  best.iter.oob <- gbm.perf(fit_train,method="OOB")  # returns out-of-bag estimated best number of trees
  print(best.iter.oob)
  best.iter.cv <- gbm.perf(fit_train,method="cv")   # returns K-fold cv estimate of best number of trees
  print(best.iter.cv)

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

- Evaluation on Test Set

```r
if(run.gbm){
  tm_test = NA
  feature_test <- as.matrix(dat_test[, 1:ncol(dat_test)-1])
  if(run.test){
    load(file="../output/fit_train.RData")
    tm_test <- system.time(prob_pred<-test_gbm(fit_train,as.data.frame(feature_test),n=best.iter.cv,pred

    label_pred <- colnames(prob_pred)[apply(prob_pred, 1, which.max)]

  } else {
    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')})
  }
}
```

———THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL.————-

Advanced Model 2: Random Forest

- Rebalance Train Set

```r
if(run.rf){
  # 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)
}
```

- Tune Parameters for Random Forest

```r
if(run.rf){
  source("../lib/random_forest.R")
  if(tune.random.forest){
  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")
  )
  rf.tune
}
```

mtry = 154 is the best.

- Find the Best ntrees

```r
if(run.rf){
  source("../lib/random_forest.R")
  #Train 500
  if(tune.random.forest){
  time.rf.train <- system.time(random_forest_fit_500 <- random_forest_train_500(dat_train_balanced_rose
  save(random_forest_fit_500, file = "../output/rf_train_500_trees.RData")
  }
  #Test 500
  random_forest_test_prep=NA
  if(tune.random.forest){
    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, "%.\
    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")
    }
  # 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(tune.random.forest){
    time.rf.train <- system.time(random_forest_fit_1000 <- random_forest_train_1000(dat_train_balanced_
    save(random_forest_fit_1000, file = "../output/rf_train_1000_trees.RData")
    }
  #Test 1000
```

```
random_forest_test_prep=NA
if(tune.random.forest){
  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, "%.\
  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(tune.random.forest){
    time.rf.train <- system.time(random_forest_fit_1500 <- random_forest_train_1500(dat_train_balance
    save(random_forest_fit_1500, file = "../output/rf_train_1500_trees.RData")
    }
#Test 1500
random_forest_test_prep=NA
if(tune.random.forest){
  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, "%.\
  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(tune.random.forest){
  time.rf.train <- system.time(random_forest_fit_2000 <- random_forest_train_2000(dat_train_balanced_
    save(random_forest_fit_2000, file = "../output/rf_train_2000_trees.RData")
    }
  #Test 2000
  random_forest_test_prep=NA
  if(tune.random.forest){
    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, "%.\
    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(tune.random.forest){
    time.rf.train <- system.time(random_forest_fit_2500 <- random_forest_train_2500(dat_train_balanced_
    save(random_forest_fit_2500, file = "../output/rf_train_2500_trees.RData")
    }
  #Test 2500
  random_forest_test_prep=NA
  if(tune.random.forest){
    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, "%.\
    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 Random Forest with Tuned Parameters

```r
if(run.rf){
  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 Random Forest with Tuned Parameters

```r
if(run.rf){
  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)
    )
  }
  ## reweight the test data to represent a balanced label distribution
  if (run.gbm){
    accu <- mean(dat_test$label == label_pred)
    cat("The accuracy of GBM baseline model is", mean(dat_test$label == label_pred)*100, "%.\n")
  } else {
    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")
  }
  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 Set

```
if(run.rf){
  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 Random Forest

```
if(run.rf){
  cat("The AUC of model after reweighting: RF", "is", auc_rf, ".\n")
  cat("The accuracy of model: Random Forest on testing data", "is", accu_rf_test*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")
  #label_test

  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")
```

——THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL.———-

Advanced Model 3: SVM Model

- Balance the Training Set

```
if(run.svm){
  tm_svm_rebalanced_train <- NA
  if(sample.reweight){
    tm_svm_rebalanced_train <- system.time(svm_training_data <- ROSE(label ~ ., data = dat_train)$data)
    save(svm_training_data, file="../output/svm_training_data.RData")
    save(tm_svm_rebalanced_train, file="../output/tm_svm_rebalanced_train.RData")
  } else {
    svm_training_data <- dat_train
    tm_svm_rebalanced_train <- tm_feature_train
  }
} #else {
  #load(file="../output/tm_svm_rebalanced_train.RData")
#}
```

- Model Selection

```
if(run.svm){
  tm_svm_linear_mod <- NA
  tm_svm_radial_mod <- NA

  if(model.selection){
    svm_model_auc <- rep(NA, 2)

    ### linear kernel
    if(run.cv){
      #best.linear.cost <- svm_linear_cost_tune(svm_training_data)
      #cat("The best cost for svm model with linear kernel is: ", best.linear.cost$best.parameters$cost
      tm_svm_linear_mod <- system.time(svm_linear_mod <- svm_linear_train(svm_training_data, 0.01, K))
      save(svm_linear_mod, file="../output/svm_linear_mod.RData")
      save(tm_svm_linear_mod, file="../output/tm_svm_linear_mod.RData")
    } else {
      load(file="../output/svm_linear_mod.RData")
      load(file="../output/tm_svm_linear_mod.RData")
    }
    svm_linear_pred <- svm_test(svm_linear_mod, svm_training_data, TRUE)
    #mean(round(svm_linear_pred == svm_training_data$label))
    svm_linear_accu <- mean(round(svm_linear_pred == svm_training_data$label))
    tpr.fpr_linear <- WeightedROC(as.numeric(svm_linear_pred), svm_training_data$label)
    svm_model_auc[1] <- WeightedAUC(tpr.fpr_linear)


    ### radial basis kernel
    if(run.cv){
      #best.radial.cost <- svm_radial_cost_tune(svm_training_data)
      #radial_cost = best.radial.cost$best.parameters$cost
      #radial_gamma = best.radial.cost$best.parameters$gamma

      tm_svm_radial_mod < system.time(svm_radial_mod <- svm_radial_train(svm_training_data, 1, K))
      save(svm_radial_mod, file="../output/svm_radial_mod.RData")
      save(tm_svm_radial_mod, file="../output/tm_svm_radial_mod.RData")
    } else {
      load(file="../output/svm_radial_mod.RData")
      load(file="../output/tm_svm_radial_mod.RData")
    }
    svm_radial_pred <- svm_test(svm_radial_mod, svm_training_data, TRUE)
    # evaluate performance
    svm_radial_accu <- mean(round(svm_radial_pred == svm_training_data$label))
    tpr.fpr_default <- WeightedROC(as.numeric(svm_radial_pred), svm_training_data$label)
    svm_model_auc[2] <- WeightedAUC(tpr.fpr_default)
  } else {
    load(file="../output/svm_linear_mod.RData")
    load(file="../output/tm_svm_linear_mod.RData")
    load(file="../output/svm_radial_mod.RData")
    load(file="../output/tm_svm_radial_mod.RData")
  }


  ### Evaluation on Testing Data
  tm_svm_rebalanced_test <- NA
  if(sample.reweight){
    tm_svm_rebalanced_test <- system.time(svm_testing_data <- ROSE(label ~ ., data = dat_test)$data)
    save(svm_testing_data, file="../output/svm_testing_data.RData")
    save(tm_svm_rebalanced_test, file="../output/tm_svm_rebalanced_test.RData")
  } else {
    svm_testing_data <- dat_test
    tm_svm_rebalanced_test <- tm_feature_test
  }

  if(run.svm.test){
    svm_auc <- rep(NA, 2)
    svm_accu <- rep(NA, 2)
    ## linear
    tm_svm_linear_test <- system.time(svm_linear_pred <- svm_test(svm_linear_mod, svm_testing_data))
    svm_accu[1] = mean(round(svm_linear_pred == svm_testing_data$label))
    tpr.fpr.linear <- WeightedROC(as.numeric(svm_linear_pred), svm_testing_data$label)
    svm_auc[1] = WeightedAUC(tpr.fpr.linear)
    ## rbf
    tm_svm_rbf_test <- system.time(svm_rbf_pred <- svm_test(svm_radial_mod, svm_testing_data))
    svm_accu[2] = mean(round(svm_rbf_pred == svm_testing_data$label))
    tpr.fpr.rbf <- WeightedROC(as.numeric(svm_linear_pred), svm_testing_data$label)
    svm_auc[2] = WeightedAUC(tpr.fpr.rbf)

    save(tm_svm_radial_mod, file="../output/tm_svm_linear_test.RData")

    ## performance
    svm_auc
    svm_accu

    cat("The accuracy of svm model is", svm_accu[2]*100, "%.\n")
    cat("The AUC of svm model is", svm_auc[2], ".\n")
  } else {
    load(file="../output/tm_svm_rebalanced_test.RData")
  }
}# else {
  #load(file="../output/svm_radial_mod.RData")
  #load(file="../output/tm_svm_radial_mod.RData")
#}
```

- Summarize Running Time

```r
if(run.svm){
  #cat("Time for rebalancing training data =", tm_svm_rebalanced_train[1], "s \n")
  #cat("Time for rebalancing testing data =", tm_svm_rebalanced_test[1], "s \n")
  #cat("Time for training svm model =", tm_svm_radial_mod[1], "s \n")
  cat("Time for testing svm model=", tm_svm_rbf_test[1], "s \n")
}
```

——THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL.———-

Advanced Model 4: Ridge Model

- Apply Constructed Ridge Model to the Training Set

```r
if(run.ridge){
  tm_ridge_train <- NA
  if (train.ridge){
    dat_train_rebalanced <- ROSE(label ~ ., data = dat_train, seed=2020)$data
    tm_ridge_train <- system.time(ridge_cv_model<-ridge_train(train_data=dat_train_rebalanced, alpha=al
    save(ridge_cv_model, file="../output/ridge_cv_model.RData")
    save(tm_ridge_train, file="../output/ridge_train_time.RData")
  } else {
    load(file="../output/ridge_cv_model.RData")
    load(file="../output/ridge_train_time.RData")
  }
}
```

- Use Cross-Validation to Choose the Optimal Lambda with Smallest MSE

```r
if(run.ridge){
  if (run.cv){
    set.seed(2020)
    feature_train = as.matrix(dat_train[, -6007])
    label_train = as.integer(dat_train$label)
    ridge_model = cv.glmnet(x=feature_train, y=label_train, alpha=alpha, nfolds=K, lambda=lambda)
    opt_lambda = ridge_model$lambda.min
    save(opt_lambda, file="../output/ridge_optimal_lambda.RData")
  } else {
    load(file="../output/ridge_optimal_lambda.RData")
  }
}
```

- Predict on Testing Set with the Optimal Lambda

```r
if(run.ridge){
  tm_ridge_test = NA
  if(run.test){
    load("../output/ridge_cv_model.RData")
    feature_test <- as.matrix(dat_test[, -6007])
    tm_ridge_test <- system.time(label_pred<-as.integer(ridge_test(model=ridge_cv_model, features=featu
    save(tm_ridge_test, file="../output/ridge_test_time.RData")
  } else{
    load(file="../output/ridge_test_time.RData")
  }
}
```

- Summarize Running Time

```r
if(run.ridge){
  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 ridge model=", tm_ridge_train[1], "s \n")
  cat("Time for testing ridge model=", tm_ridge_test[1], "s \n")
}
```

- Evaluation on Independent Testing Data

```r
if(run.ridge){
  load("../output/ridge_cv_model.RData")
  feature_test <- as.matrix(dat_test[, -6007])
  label_pred = as.integer(predict(ridge_cv_model, s=opt_lambda, newx=feature_test, type='class'))
  label_test = as.integer(dat_test$label)
  compare <- cbind (label_test, label_pred)
  ridge_accuracy = mean(apply(compare, 1, min)/apply(compare, 1, max))
  cat("The accuracy of the ridge model is", ridge_accuracy*100, "%.\n")
  ridge_AUC = auc(roc(label_pred,label_test))
  cat("The AUC of the ridge model is", ridge_AUC, ".\n")
}
```

——THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL. THIS IS TO SEPARATE EACH MODEL.———-

Advanced Model 5: PCA + LDA

- Rebalance Training Set

```r
if(run.pca_lda){
  if(sample.reweight){
    balanced_train_data <- ROSE(label~.,data = dat_train)$data
    save(balanced_train_data, file="../output/feature_balanced_train.RData")
  } else {
    load(balanced_train_data, file="../output/feature_train.RData")
  }
}
```

- Perform PCA for Dimension Reduction

Since there are over 6000 features, we implement the PCA method to reduce dimension according to the covariance matrix. We only retain PCs with large variance.

```r
if(run.pca_lda){
  if(run.pca_lad.test){
    if(sample.reweight){
      balanced_test_data <- ROSE(label~.,data = dat_test)$data
      save(balanced_test_data, file="../output/feature_balanced_test.RData")
    } else {
      load(balanced_test_data, file="../output/feature_balanced_test.RData")
    }
  }

  if(run.select_PC){
    #separate the features from label
    dat_train_new <- balanced_train_data[,-dim(balanced_train_data)[2]]
    dat_test_new <- balanced_test_data[,-dim(balanced_test_data)[2]]
    #create a vector contain target number of PCs
    num.pca <- c(10,50,500,1000)
    train_pca <- function(num.pca){
      for(i in 1:length(num.pca)){
        #start time for training the model
        train.model.start = proc.time()
        #run PCA
        pca <- prcomp(dat_train_new)
        #store for each potential PC
        train_pca <- data.frame(pca$x[,1:num.pca[i]], label = balanced_train_data[dim(balanced_train_da
        pred_pca <- predict(pca,dat_test_new)
        test_pca <- data.frame(pred_pca[,1:num.pca[i]], label = balanced_test_data[dim(balanced_test_da
        #fitting the lda model
        lda_pca <- lda(label ~ ., data = train_pca)
        #stop time for training the model
        train.model.end = proc.time()
        #start time for testing the model
        test.model.start = proc.time()
        #predict lda model
        lda_pred_pca = predict(lda_pca,test_pca[-dim(test_pca)[2]])
        #end time for testing the model
        test.model.end = proc.time()
        #test accuracy
        test_accuracy=confusionMatrix(lda_pred_pca$class, test_pca$label)$overall[1]
        print(list(l1=train.model.end - train.model.start,
               l2=test.model.end - test.model.start,
               l3=test_accuracy))
      }
    }
  train_pca(num.pca)
  }
}
```

By comparing the training time, test time and accuracy, we use model with 50 PCs.

- Model Training

```
if(run.pca_lda){
  train.model.start = proc.time()
  if(run.lda){
    pca_10 <- prcomp(dat_train_new)
    train_pca_10 <- data.frame(pca_10$x[,1:50], label = balanced_train_data[dim(balanced_train_data)[2]
    pred_pca_10 <- predict(pca_10,dat_test_new)
    test_pca_10 <- data.frame(pred_pca_10[,1:50], label = balanced_test_data[dim(balanced_test_data)[2]
    save(train_pca_10, file="../output/feature_pca_train.RData")
    save(test_pca_10, file="../output/feature_pca_test.RData")
  } else {
    load(train_pca_10, file="../output/feature_pca_train.RData")
    load(test_pca_10, file="../output/feature_pca_test.RData")
  }
  #calculate the training time
  lda_pca_10 <- lda(label ~ ., data = train_pca_10, cv = TRUE)
  train.model.end = proc.time()
}
```

- Calculate the Training and Testing Accuracy of LDA Model

```
if(run.pca_lda){
  test.model.start = proc.time()
  pred_train_lda <- predict(lda_pca_10, train_pca_10[-dim(train_pca_10)[2]])
  accu_train_lda <- mean(pred_train_lda$class == train_pca_10$label)
  cat("The trainig accuracy of model: LDA", "is", accu_train_lda*100, "%.\n")
  #calculating the test time
  if(run.test){
    pred_test_lda <- predict(lda_pca_10, test_pca_10)
  }
  test.model.end = proc.time()
  save(pred_test_lda, file="../output/fit_train.RData")
  accu_test_lda <- mean(pred_test_lda$class == test_pca_10$label)
  cat("The accuracy of model: LDA", "is", accu_test_lda*100, "%.\n")
  tpr.fpr <- WeightedROC(as.numeric(pred_test_lda$class), test_pca_10$label)
  lda_auc = WeightedAUC(tpr.fpr)
  cat("The AUC of model: LDA is", lda_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.

```
if(run.pca_lda){
  tm_train <- train.model.end - train.model.start
  tm_test <- test.model.end - test.model.start
  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.