Main

Group 8

11/4/2020

Baseline Model (GBM with default feature)

– further down feature is updated and added to XGBoost as the proposed new model –

Packages needed for evaluating proposed Facial Expression Recognition framework. The "if statement" checks if it is needed. The Library statements are used to call forth the function.

```r
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("geometry")){
  install.packages("geometry")
}
```

```r
if(!require("gbm")){
  install.packages("gbm")
}
if(!require("smotefamily")){
  install.packages("smotefamily")
}
if(!require("ROSE")){
  install.packages("ROSE")
}
if(!require("xgboost")){
  install.packages("xgboost")
}
if(!require("tidyr")){
  install.packages("tidyr")
}
if(!require("e1071")){
  install.packages("e1071")
}
library(gbm)
library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(glmnet)
library(WeightedROC)
library(geometry)
library(smotefamily)
library(ROSE)
library(xgboost)
library(tidyr)
library(e1071)
```

Step 0 set work directories and set the seed.

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

Directories for training images and training fiducial points. The data is located in different subfolders.

```r
train_dir <- "../data/train_set/" # This will be modified for different data sets.
test_dir <- "../data/test_set_predict/" # For the presentation test set
train_image_dir <- paste(train_dir, "images/", sep="") # subfolder images
train_pt_dir <- paste(train_dir,  "points/", sep="") # subfolder points
train_label_path <- paste(train_dir, "label.csv", sep="") # not in a subfolder
test_image_dir <- paste(test_dir, "images/", sep="") # subfolder images
test_pt_dir <- paste(test_dir,  "points/", sep="") # subfolder points
test_label_path <- paste(test_dir, "label_prediction.csv", sep="") # not in subfolders
```

Step 1: set up controls for evaluation experiments.

```
run.default <- TRUE # run default gbm method
run.fiximage <- TRUE # change the position and zoom in the image
run.improved <- TRUE # improved feature
run.feature.train <- TRUE # process features for training set
run.test.claimed <- TRUE # run evaluation on an independent test set
run.feature.test <- TRUE # process features for test set
### Important Note
### Please set run.test.real to FALSE if you are going to train the data
run.test.real <- FALSE # run the test dataset on present day
sample.reweight <- TRUE # run sample reweighting in model training
run.cv <- TRUE # run cross-validation on the training set
K <- 5  # number of CV folds
run.xgb.rose <- FALSE # for evaluation using rose
run.xgb.smote <- TRUE # for evaluating using smote
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications. In this Base Model gbm, we tune parameter k (the amount of trees) for decision trees with boost gradient.

```
k = c(50,100,150,200,250,300) # number of trees
model_labels = paste("Boosted Decision Machine with number of trees K =", k)
```

Step 2: import data and train-test split

```
if ((run.default|run.improved|run.fiximage == TRUE) & (run.test.real == FALSE)){ # train-test split
  info <- read.csv(train_label_path) #reading in the data
  n <- nrow(info) #number of rows
  n_train <- round(n*(4/5), 0)
  train_idx <- sample(info$Index, n_train, replace = F) # train data
  test_idx <- setdiff(info$Index, train_idx) # test data
}
# For presentation day specifically
if (run.test.real == TRUE){
  info <- read.csv(test_label_path)
  info$label <- -1 # assign the label to avoid problem not having it
}
```

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.

```
if ((run.default|run.improved|run.fiximage == TRUE) & (run.test.real == FALSE)){ #parameters are set to
  n_files <- length(list.files(train_image_dir))
  image_list <- list()
  for(i in 1:100){ #repeat reading a smaller batch(e.g 100) and process them
```

```
    image_list[[i]] <- readImage(paste0(train_image_dir, sprintf("%04d", i), ".jpg"))
  }
}
if (run.test.real == TRUE){
  n_files <- length(list.files(test_image_dir))
  image_list <- list()
  for(i in 1:100){ # repeat reading a smaller batch(e.g 100) and process them.
    image_list[[i]] <- readImage(paste0(test_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
if ((run.default|run.improved|run.fiximage == TRUE) & (run.test.real == FALSE)){
  readMat.matrix <- function(index){ # read the matlab files
      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") # store data as a RData
}
if (run.test.real == TRUE){ #parameter set to true
  readMat.matrix <- function(index){ # read the matlab files
      return(round(readMat(paste0(test_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_test.RData") # store data as a RData
}
```

Step 3: construct features and responses (for baseline model)

```
source("../lib/feature_default.R") # + `feature_default.R` + Input: list of images or fiducial point  +
                                    # file that contains extracted features and corresponding responses
if(run.default){ # train responce
  tm_feature_train_default <- NA
  if(run.feature.train){
    tm_feature_train_default <- system.time(dat_train_default <- feature_default(fiducial_pt_list, train
    # fratures saved and the runtime saved
    save(dat_train_default, file="../output/feature_train_default.RData") # save as .Rdata
    save(tm_feature_train_default, file="../output/tm_feature_train_default.RData") # save as .Rdata
  }

  tm_feature_test_default <- NA # test responce
  if(run.feature.test){
    tm_feature_test_default <- system.time(dat_test_default <- feature_default(fiducial_pt_list, test_i
```

```r
    # fratures saved and the runtime saved
    save(dat_test_default, file="../output/feature_test_default.RData") # save as .Rdata
    save(tm_feature_test_default, file="../output/tm_feature_test_default.RData") # save as .Rdata
  }
}
```

Step 4: Train a classification model with training features and responses. Call the train model and test model from library.

```r
#In this Baseline, we use decision trees with gradient boost to do classification.
source("../lib/train_baseline_gbm.R") # + Input: a data frame containing features and labels and a para
                                      #  + Output:a trained model
source("../lib/test_baseline_gbm.R") #  + Input: the fitted classification model using training data an
                                      # + Input: an R object that contains a trained classifier.
                                      #  + Output: training model specification
```

```r
source("../lib/cross_validation_baseline.R")
if(run.cv){
  err_cv <- matrix(0, nrow = length(k), ncol = 2)
  for(i in 1:length(k)){
    cat("k=", k[i], "\n")
    err_cv[i,] <- cv.function(dat_train_default, K, k[i])
  save(err_cv, file="../output/err_cv.RData") #saved as RData
  }
}
```

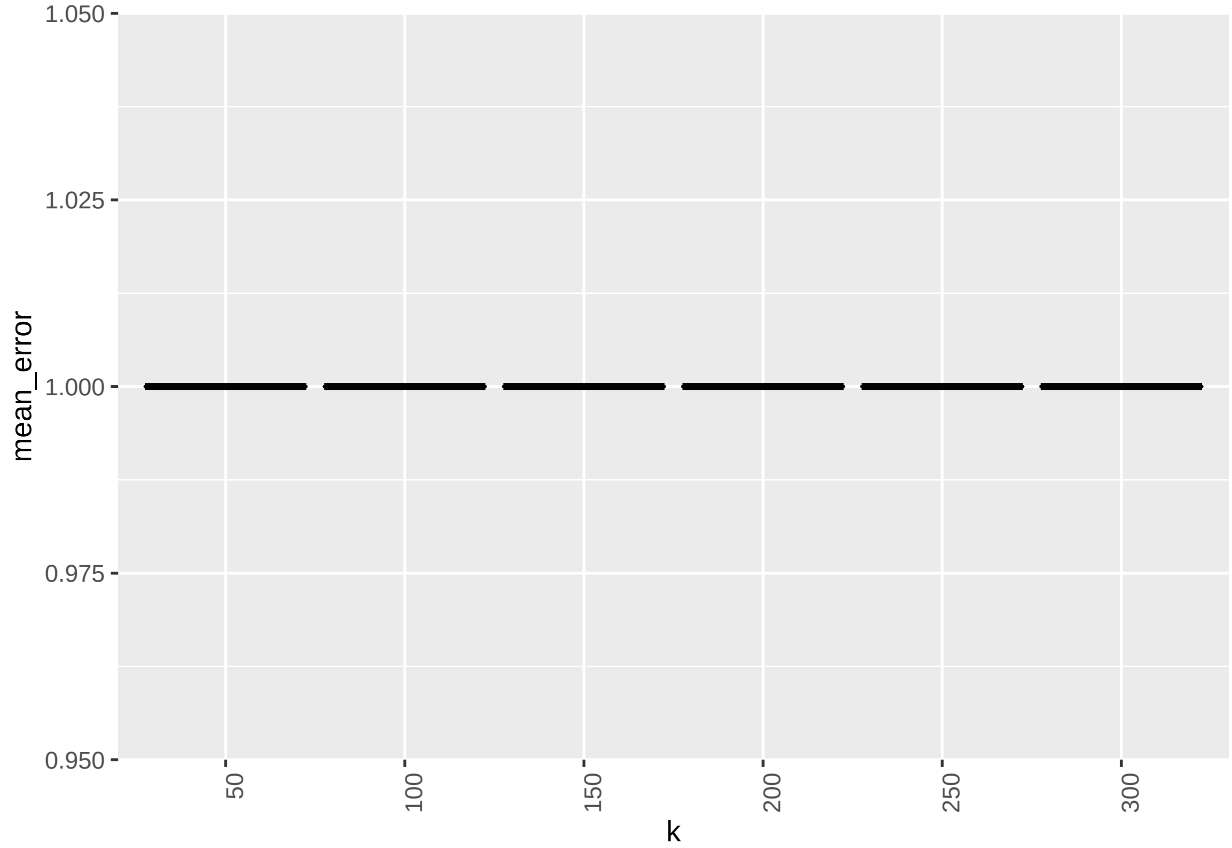
Model selection with cross-validation. Done model selection by choosing among different values of training model parameters.

```
## k= 50
## k= 100
## k= 150
## k= 200
## k= 250
## k= 300
```

Visualize cross-validation results.

```r
if(run.cv){
  load("../output/err_cv.RData")
  err_cv <- as.data.frame(err_cv)
  colnames(err_cv) <- c("mean_error", "sd_error")
  err_cv$k = as.factor(k)
  err_cv %>%
    ggplot(aes(x = k, 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))
}
```



Choose the "best" parameter value

```
if(run.cv){
  load("../output/err_cv.RData")
  err_cv <- as.data.frame(err_cv) # to save the time, can uncomment this two line to directly import th
  model_best <- k[which.min(err_cv[,1])] # best model
}
par_best <- list(k = model_best) # best parameter value
```

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

```
weight_train <- rep(NA, length(dat_train_default$label)) # training weights
for (v in unique(dat_train_default$label)){
  weight_train[dat_train_default$label == v] = 0.5 * length(dat_train_default$label) / length(dat_train_
}
tm_train_default <- system.time(fit_train <- train(dat_train_default, weight_train, par_best)) # run ti
save(fit_train, file="../output/fit_train.RData") # save as Rdata
```

Step 5: Run test on test images

```
tm_test_default=NA
if(run.test.claimed){
  load(file="../output/fit_train.RData") # save as RData
  tm_test_default <- system.time(pred_gbm <- test(fit_train, dat_test_default)) # run time
  weight_test <- rep(NA, length(dat_test_default$label))
  for (v in unique(dat_test_default$label)){ #unique vlaues selected
    weight_test[dat_test_default$label == v] = 0.5 * length(dat_test_default$label) / length(dat_test_de
  }
}
```

evaluation

```
### Training Accuracy
accu <- sum(weight_test * (pred_gbm[[2]] == dat_test_default$label))/sum(weight_test)
cat("The accuracy of model:", model_labels[which.min(err_cv[,1])], "is", accu*100, "%.\n") # accuracy
```

```
## The accuracy of model: Boosted Decision Machine with number of trees K = 50 is 66.07482 %.
```

```
auc <- WeightedROC(pred_gbm[[1]], dat_test_default$label, weight_test) %>% WeightedAUC # weighted
cat("The AUC of model:", model_labels[which.min(err_cv[,1])], "is", auc, ".\n") # AUC
```

```
## The AUC of model: Boosted Decision Machine with number of trees K = 50 is 0.7265519 .
```

Summarize Running Time

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

```
## Time for constructing default training features= 0.67 s
```

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

```
## Time for constructing default testing features= 0.14 s
```

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

```
## Time for training model= 25.99 s
```

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

```
## Time for testing model= 13.43 s
```

Proposed Imporved Model (XGBoost with improved features)

sourcing R files

```
source("../lib/xgb_tune.R")
source("../lib/xgb_train.R")
source("../lib/xgb_test.R")
```

Data Preprocessing: Rotate the image to be upright and zoom in to get the face and the fiducial points at the center.

```
if (run.fiximage){
  source("../lib/img_process.R") # source in script
  fiducial_pt_list_processed <- list() #empty list to store points in
  for (i in 1:n){
    fiducial_pt_list_processed[[i]] <- img_process(fiducial_pt_list[[i]])
  }
  save(fiducial_pt_list_processed, file = "../output/fiducial_pt_list_processed.RData") #save as Rdata
}
```

Acquire improved features

```
source("../lib/feature.R") # source improved feature script
load("../output/fiducial_pt_list_processed.RData") # load Rdata
tm_feature_train <- NA # feature train
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(fiducial_pt_list_processed, train_idx))
  # save feature data and save runtime
  save(dat_train, file="../output/feature_train.RData") # save Rdata
  save(tm_feature_train, file="../output/tm_feature_train.RData") # save Rdata
}
tm_feature_test <- NA # feature test
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature(fiducial_pt_list_processed, test_idx))
  # save feature data and save runtime
  save(dat_test, file="../output/feature_test.RData") # save Rdata
  save(tm_feature_test, file="../output/tm_feature_test.RData") # save Rdata
}
```

Oversample to tackle imbalance in train data

```
load("../output/feature_train.RData")
#Oversample using ROSE
dat_train_balanced_over_rose <-
  ovun.sample(label ~ ., data = dat_train, method = "over",
              N = as.numeric(table(dat_train$label)[1]) * 2)$data
table(dat_train_balanced_over_rose$label)
```

```
##
##    0    1
## 1926 1926
```

```r
save(dat_train_balanced_over_rose, file = "../output/dat_train_balanced_over_rose.RData") #save Rdata
#Oversample using SMOTE
dat_train_balanced_over_smote <-
  smotefamily::SMOTE(dat_train[ ,-152], as.numeric(dat_train$label))$data
colnames(dat_train_balanced_over_smote)[which(names(dat_train_balanced_over_smote) == "class")] <- "labe
dat_train_balanced_over_smote$label[which(dat_train_balanced_over_smote$label == 1)] <- 0
dat_train_balanced_over_smote$label[which(dat_train_balanced_over_smote$label == 2)] <- 1
table(dat_train_balanced_over_smote$label)
```

```
##
##    0    1
## 1926 1896
```

```r
save(dat_train_balanced_over_smote, file = "../output/dat_train_balanced_over_smote.RData") #save Rdata
```

```r
if(run.cv & run.xgb.rose){
  source("../lib/xgb_tune.R") #source file
  source("../lib/xgb_cv.R")
  load("../output/dat_train_balanced_over_rose.RData") #load data

  depth <- c(5, 10, 15) #set parameters
  child <- c(3, 5, 10) #set parameters
  xgb_result_cv <- xgb_tune(dat_train_balanced_over_rose, depth, child, K)
  xgb_err <- xgb_result_cv[[1]]
  tm.xgb.cv <- as.numeric(xgb_result_cv[[3]])
  xgb_err_tune <- xgb_result_cv[[1]] %>% as.data.frame()
  xgb_best_par <- xgb_result_cv[[2]] %>% as.data.frame()
}
if(run.cv & run.xgb.smote){
  source("../lib/xgb_tune.R") #source file
  source("../lib/xgb_cv.R")
  load("../output/dat_train_balanced_over_smote.RData") #load data

  depth <- c(5, 10, 15) #set parameters
  child <- c(3, 5, 10) #set parameters
  xgb_result_cv <- xgb_tune(dat_train_balanced_over_smote, depth, child, K)
  xgb_err <- xgb_result_cv[[1]]
  tm.xgb.cv <- as.numeric(xgb_result_cv[[3]])
  xgb_err_tune <- xgb_result_cv[[1]] %>% as.data.frame()
  xgb_best_par <- xgb_result_cv[[2]] %>% as.data.frame()
}
```

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

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

```r
if(run.xgb.rose){
  source("../lib/xgb_train.R") #source file
  load("../output/dat_train_balanced_over_rose.RData") #load data

  if(run.cv){
    xgb_result <- xgb_train(dat_train_balanced_over_rose, par = xgb_best_par)
    }
  else{
    xgb_result <- xgb_train(dat_train_balanced_over_rose)
    }
  xgb_model <- xgb_result[[1]]
  tm.xgb.train <- xgb_result[[2]]
  save(xgb_model, file = "../output/xgb_model.RData") #save Rdata
  save(tm.xgb.train, file = "../output/tm.xgb.train.RData") #save Rdata
}
if(run.xgb.smote){
  source("../lib/xgb_train.R")
  load("../output/dat_train_balanced_over_smote.RData")

  if(run.cv){
    xgb_result <- xgb_train(dat_train_balanced_over_smote, par = xgb_best_par)
    }
  else{
    xgb_result <- xgb_train(dat_train_balanced_over_smote)
    }
  xgb_model <- xgb_result[[1]]
  tm.xgb.train <- xgb_result[[2]]
  save(xgb_model, file = "../output/xgb_model.RData") #save Rdata
  save(tm.xgb.train, file = "../output/tm.xgb.train.RData") #save Rdata
}
```

Step 5: Run test on test images

```r
#Prediction of the xgb models
source("../lib/xgb_test.R")
load("../output/xgb_model.RData")
load("../output/feature_test.RData")
xgb_result_test <- xgb_test(xgb_model, dat_test[,-ncol(dat_test)])
xgb_pred <- xgb_result_test[[1]]
xgb_pred_class <- round(xgb_pred)
tm.xgb.test <- xgb_result_test[[2]]
save(tm.xgb.test, file = "../output/tm.xgb.test.RData") #save Rdata

weight_test_improved <- rep(NA, length(dat_test$label))
for (v in unique(dat_train$label)){
  weight_test_improved[dat_test$label == v] = 0.5 * length(dat_test$label) / length(dat_test$label[dat_t
}
```

evaluation

```r
accu.unweighted <- mean(dat_test$label == xgb_pred_class)
accu <- sum(weight_test_improved * (xgb_pred_class == dat_test$label))/sum(weight_test_improved)
cat("The accuracy of model",  accu*100, "%.\n")
```

```
## The accuracy of model 74.19694 %.
```

```r
auc <- WeightedROC(xgb_pred, dat_test$label, weight_test_improved) %>% WeightedAUC
cat("The AUC of model", "is", auc, ".\n")
```

```
## The AUC of model is 0.8618359 .
```

Summarize Running Time

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

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

```
## Time for constructing default training features= 18.87 s
```

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

```
## Time for constructing default testing features= 4.89 s
```

```r
cat("Time for training model=", tm.xgb.train[1], "s \n")
```

```
## Time for training model= 37.95 s
```

```r
cat("Time for testing model=", tm.xgb.test[1], "s \n")
```

```
## Time for testing model= 0 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.