Main

Ran (Rena) Lu

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")
}
```

```
## Warning: package 'R.matlab' was built under R version 3.6.3
```

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

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

```
## Warning: package 'dplyr' was built under R version 3.6.3
```

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

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

```
## Warning: package 'ggplot2' was built under R version 3.6.3
```

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

```
## Warning: package 'caret' was built under R version 3.6.3
```

```
## Warning: package 'lattice' was built under R version 3.6.3
```

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

```
## Warning: package 'randomForest' was built under R version 3.6.3
```

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

```
## Warning: package 'LncFinder' was built under R version 3.6.3
```

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

```
## Warning: package 'gbm' was built under R version 3.6.3
```

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

```
## Warning: package 'MASS' was built under R version 3.6.3
```

```
#library(tidyverse)
library(randomForest)
library(LncFinder)
library(gbm)
```

Step 0 set work directories

```
set.seed(0)
setwd("C:/Users/59482/Desktop/Columbia/Second Term/ads/Project3/Spring2020-Project3-ads-spring2020-project3-group3")
# 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="")
```

Part 1: Baseline Model

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
- (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

```
run.cv=TRUE # run cross-validation on the training set
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
```

Using cross-validation, we compare the performance of models with different specifications. In this Code, we tune parameter k (number of stumps) for Boosted Decision Machine.

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

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)
set.seed(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.

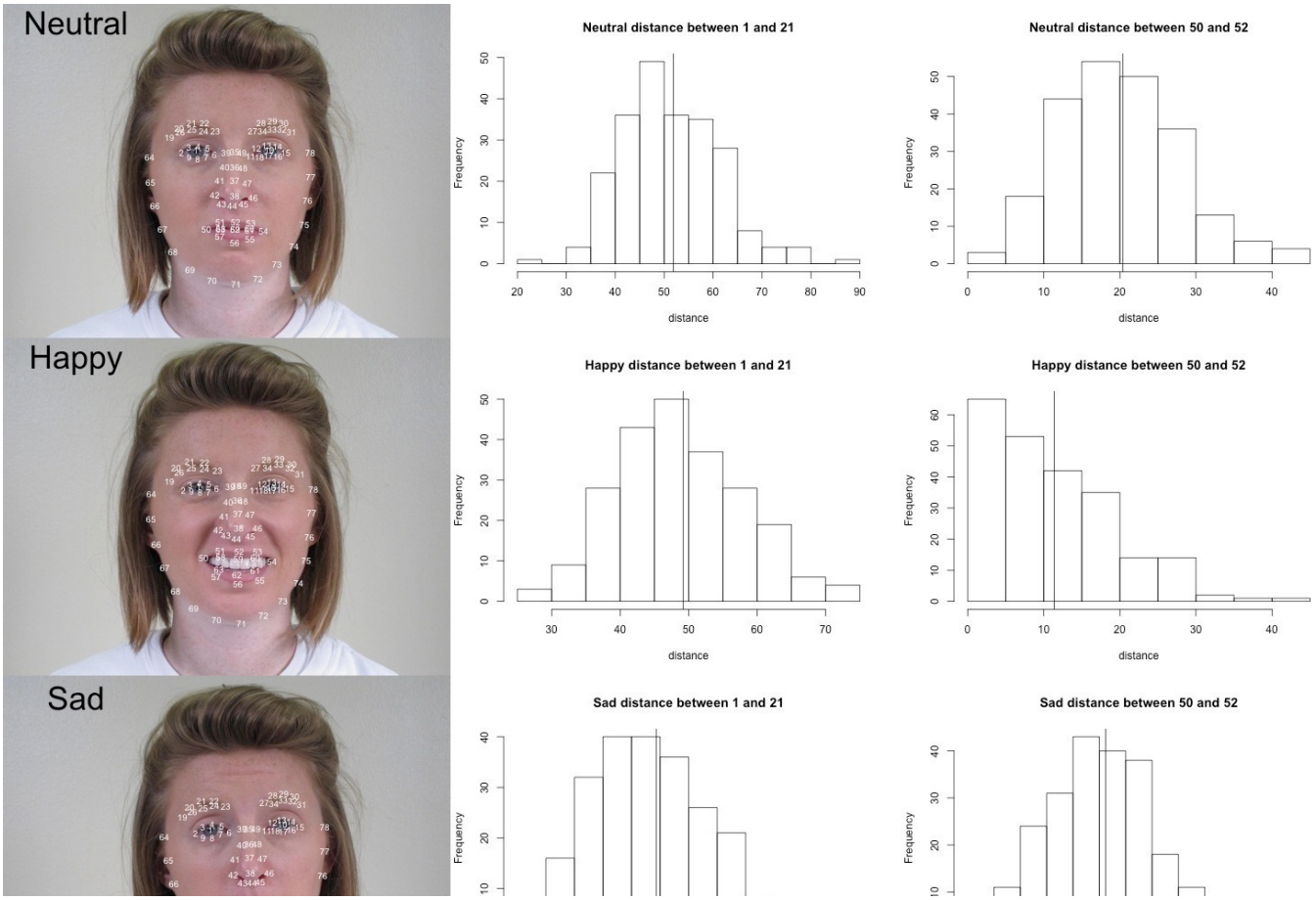
```
#function to read fiducial points
#input: index
#output: matrix of fiducial points corresponding to the index
n_files <- length(list.files(train_image_dir))

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)
```

```
## Warning in readMat(paste0(train_pt_dir, sprintf("%04d", index), ".mat")):
## strings not representable in native encoding will be translated to UTF-8
```

```
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.



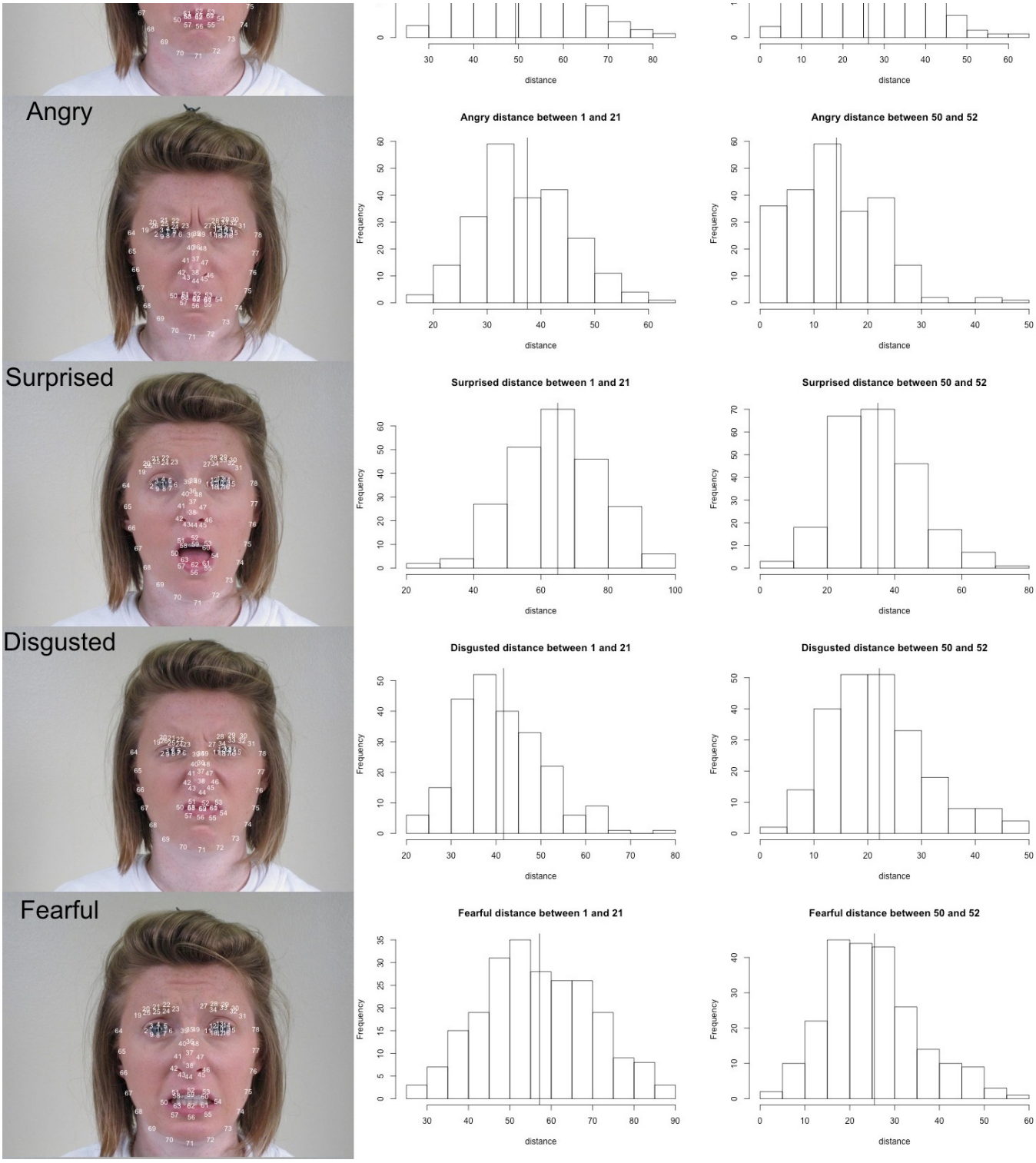


Figure1

`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

```
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))
}

tm_feature_test <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature(fiducial_pt_list, test_idx))
}

save(dat_train, file="../output/feature_train.RData")
save(dat_test, 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
- In this Starter Code, we use KNN to do classification.

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

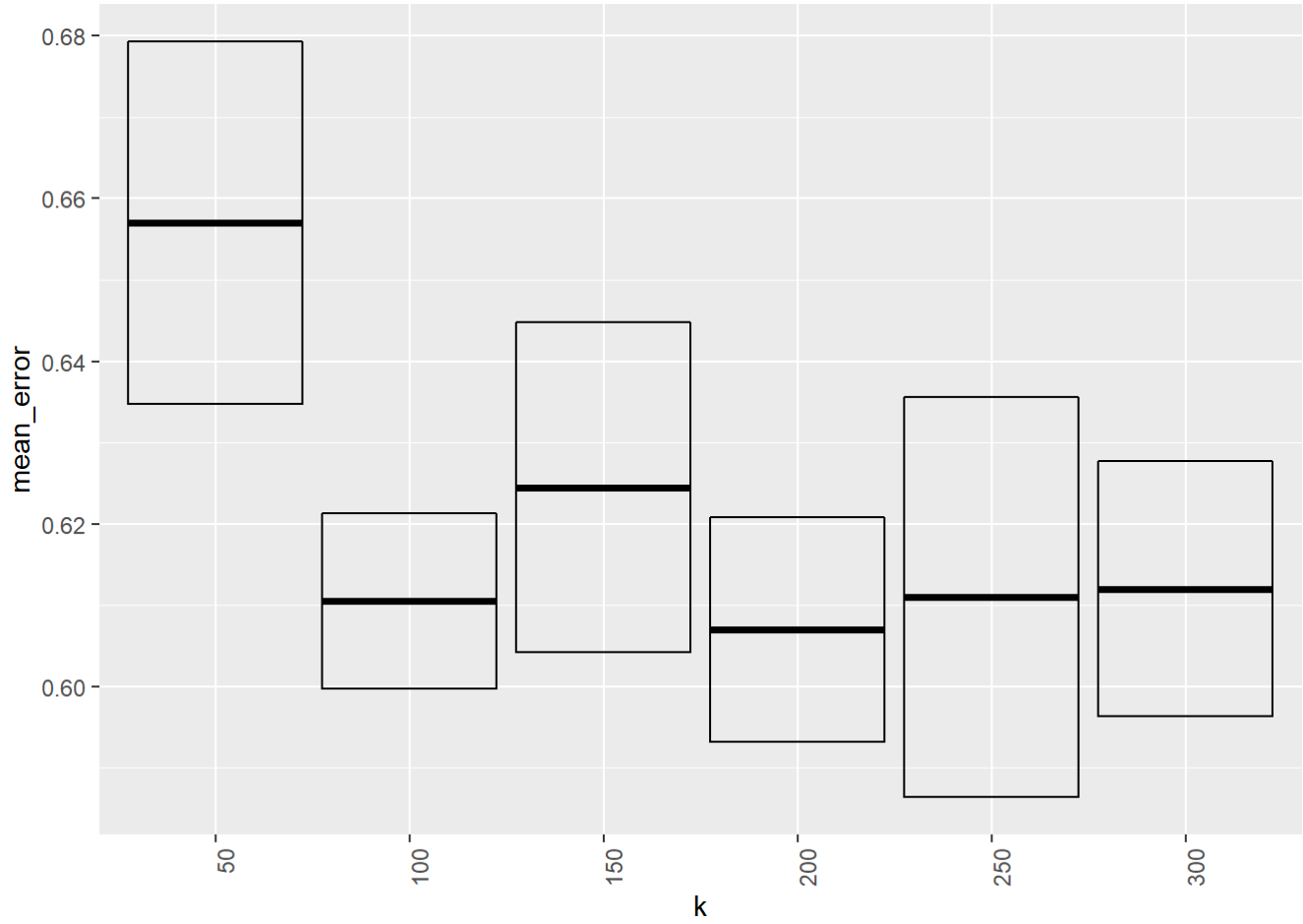
Model selection with cross-validation

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

```
# source("../lib/cross_validation.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, K, k[i])
#   save(err_cv, file="../output/err_cv.RData")
#   }
# }
```

Visualize cross-validation results.

```
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
e data
  model_best <- k[which.min(err_cv[,1])]
}
par_best <- list(k = model_best)
# k best model = 200
```

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

```
tm_train=NA
tm_train <- system.time(fit_train <- train(dat_train, par_best))
save(fit_train, file="../output/fit_train.RData")
```

Step 5: Run test on test images

```
tm_test=NA
if(run.test){
  load(file="../output/fit_train.RData")
  tm_test <- system.time(pred_gbm <- test(fit_train, dat_test))
}
```

- evaluation

```
pred <- factor(pred_gbm)
accu <- mean(dat_test$emotion_idx == pred)
cat("The accuracy of model:", model_labels[which.min(err_cv[,1])], "is", accu*100, "%.\n")
```

```
## The accuracy of model: Boosted Decision Machine with number of trees K = 200 is 44.4 %.
```

```
#pred <- factor(pred_gbm)
library(caret)
confusionMatrix(pred, dat_test$emotion_idx)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
##         1  13  0  4  1  0  1  0  0  0  0  0  0  0  0  0  1  0  3  2  0  0  0
##         2   0 12  0  0  0  0  0  1  6  0  0  0  0  0  0  0  1  0  1  1  0  0
##         3   1  0  9  1  0  0  0  0  0  3  0  0  0  0  0  4  0  0  0  0  0  0
##         4   1  0  3 11  1  2  0  0  0  2  3  1  3  0  0  1  0  0  0  0  0  0
##         5   0  0  0  0 13  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0
##         6   0  1  1  2  0  8  0  0  0  1  1  5  2  0  0  1  0  0  0  0  0  0
##         7   0  1  0  0  2  0 15  2  2  0  0  0  0  0  2  0  1  0  3  0  2  0
##         8   0  2  0  0  0  0  0 18  2  0  0  0  0  0  0  0  0  0  0  2  0  0
##         9   0  0  0  0  0  1  0  1 16  0  0  0  0  0  0  0  0  0  0  1  0  0
##         10  1  0  1  2  0  1  0  0  0  4  4  1  3  0  0  1  0  0  0  0  2  2
##         11  0  0  0  2  0  1  0  0  1  2 12  0  1  0  0  0  0  0  0  0  1  1
##         12  0  0  0  0  0  4  0  0  0  0  3  9  1  0  0  0  0  0  1  0  0  2
##         13  0  0  2  0  0  2  0  0  0  2  1  7  2  0  0  1  0  0  1  0  0  2
##         14  0  0  0  0  2  0  0  0  0  0  0  0  1 18  5  0  1  0  0  1  1  0
##         15  0  0  0  0  6  0  0  0  0  0  0  1  1  3  5  1  0  1  1  0  0  0
##         16  0  0  2  0  0  1  1  1  0  0  0  1  0  0  0 12  0  0  0  0  0  2
##         17  0  0  0  0  3  0  1  3  0  0  0  0  0  0  1  1 10  4  1  4  1  0
##         18  0  0  0  0  2  0  0  0  0  0  0  0  1  0  0  0  6  7  3  0  1  0
##         19  0  0  1  0  0  0  2  0  0  1  0  0  0  0  2  0  0  1  6  4  2  1
##         20  0  1  0  0  0  0  2  1  0  0  0  0  1  1  0  0  2  1  5  6  2  3
##         21  0  0  0  0  0  0  2  0  1  0  2  1  3  0  2  0  1  0  3  1 11  4
##         22  0  0  2  0  0  0  0  0  0  1  2  2  1  0  0  0  0  1  4  2  0  5
##
## Overall Statistics
##
##                Accuracy : 0.444
##                  95% CI : (0.3999, 0.4888)
##     No Information Rate : 0.062
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4178
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.8125   0.7059   0.3600   0.5789   0.4483   0.3810
## Specificity            0.9752   0.9793   0.9811   0.9647   0.9936   0.9708
## Pos Pred Value         0.5200   0.5455   0.5000   0.3929   0.8125   0.3636
## Neg Pred Value         0.9937   0.9895   0.9668   0.9831   0.9669   0.9728
## Prevalence             0.0320   0.0340   0.0500   0.0380   0.0580   0.0420
## Detection Rate         0.0260   0.0240   0.0180   0.0220   0.0260   0.0160
## Detection Prevalence   0.0500   0.0440   0.0360   0.0560   0.0320   0.0440
## Balanced Accuracy      0.8939   0.8426   0.6705   0.7718   0.7210   0.6759
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.6522   0.6667   0.5714    0.2500    0.4286    0.3214
## Specificity            0.9686   0.9873   0.9936    0.9628    0.9809    0.9767
## Pos Pred Value         0.5000   0.7500   0.8421    0.1818    0.5714    0.4500
## Neg Pred Value         0.9830   0.9811   0.9751    0.9749    0.9666    0.9604
## Prevalence             0.0460   0.0540   0.0560    0.0320    0.0560    0.0560
## Detection Rate         0.0300   0.0360   0.0320    0.0080    0.0240    0.0180
## Detection Prevalence   0.0600   0.0480   0.0380    0.0440    0.0420    0.0400
```

```
## Balanced Accuracy      0.8104   0.8270   0.7825   0.6064   0.7048   0.6491
##                     Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity            0.1000   0.8182   0.2941   0.5000   0.4348
## Specificity            0.9625   0.9770   0.9710   0.9832   0.9602
## Pos Pred Value         0.1000   0.6207   0.2632   0.6000   0.3448
## Neg Pred Value         0.9625   0.9915   0.9751   0.9750   0.9724
## Prevalence             0.0400   0.0440   0.0340   0.0480   0.0460
## Detection Rate         0.0040   0.0360   0.0100   0.0240   0.0200
## Detection Prevalence   0.0400   0.0580   0.0380   0.0400   0.0580
## Balanced Accuracy      0.5312   0.8976   0.6326   0.7416   0.6975
##                     Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity            0.3684   0.1935   0.2727   0.4783   0.2273
## Specificity            0.9730   0.9701   0.9603   0.9581   0.9686
## Pos Pred Value         0.3500   0.3000   0.2400   0.3548   0.2500
## Neg Pred Value         0.9750   0.9479   0.9663   0.9744   0.9646
## Prevalence             0.0380   0.0620   0.0440   0.0460   0.0440
## Detection Rate         0.0140   0.0120   0.0120   0.0220   0.0100
## Detection Prevalence   0.0400   0.0400   0.0500   0.0620   0.0400
## Balanced Accuracy      0.6707   0.5818   0.6165   0.7182   0.5979
```

Note that the accuracy is not high but is better than that of ramdom guess(4.5%).

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 baseline model features=", tm_feature_train[1], "s \n")
```

```
## Time for constructing training baseline model features= 1.4 s
```

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

```
## Time for constructing testing baseline model features= 0.11 s
```

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

```
## Time for training baseline model= 524.64 s
```

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

```
## Time for testing baseline model= 10.39 s
```

Part 2: Improved Model

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

- (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

```
run.cv=TRUE # run cross-validation on the training set
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
```

Using cross-validation, we compare the performance of models with different specifications. In this Code, we tune parameter k (number of stumps) for Boosted Decision Machine.

```
l = c(50,100,150,200,250,300) # change the labels
model_labels_improved = paste("Boosted Decision Machine with number of trees K =", l)  # change the la
bels
```

Step 2: import data and train-test split

Same as Baseline model

Step 3: construct features and responses

Feature extraction is the same as Baseline model, and PCA is used for feature selection.

`feature_improved.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_improved.R`
+ Input: train/test data
+ Output: an RData file that contains extracted features and corresponding responses

```
source("../lib/feature_improved.R") # change file name

dat_test_improved <- dat_test[,-ncol(dat_test)]
dat_train_improved <- dat_train[,-ncol(dat_train)]

tm_feature_train_improved <- NA
if(run.feature.train){
  tm_feature_train_improved <- system.time(dat_train_improved <- feature_improved(dat_train_improved))
}

tm_feature_test_improved <- NA
if(run.feature.test){
  tm_feature_test_improved <- system.time(dat_test_improved <- feature_improved(dat_test_improved))
}

save(dat_train_improved, file="../output/feature_train_improved.RData")
save(dat_test_improved, file="../output/feature_test_improved.RData")
```

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

```
source("../lib/train_svm.R")
source("../lib/test_svm.R")
source("../lib/cross_validation_svm.R")
```

```
#SVM Cross-validation
cost=seq(0.01, 0.1, length=10)
err_svm <- matrix(0, nrow = length(cost), ncol = 2)
for(i in 1:length(cost)){
    cat("cost=", cost[i], "\n")
    err_svm[i,] <- svm_cv(dat_train, K=5, cost[i])
    save(err_svm, file="../output/err_svm.RData")
  }
```

```
#Load visualization of cross validation results of svm
load("../output/err_svm.RData")
err_svm <- as.data.frame(err_svm)
colnames(err_svm) <- c("mean_error", "sd_error")
cost=seq(0.001, 0.01, length=10)
err_svm$cost = as.factor(cost)
err_svm %>%
  ggplot(aes(x = cost, 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))
```

```
cost_best_svm <- cost[which.min(err_svm[,1])]
par_best_svm <- list(cost=cost_best_svm)
# Training
tm_train_svm=NA
tm_train_svm <- system.time(fit_train_svm <- svm_train(dat_train, par_best_svm, probability = TRUE))
```

```
## Warning: package 'e1071' was built under R version 3.6.2
```

```
#Save and load model
saveRDS(fit_train_svm, "../output/fit_train_svm.RDS")
```

```
fit_train_svm<-readRDS("../output/fit_train_svm.RDS")
# Testing
tm_test_svm=NA
tm_test_svm <- system.time(pred_svm <- svm_test(fit_train_svm, dat_test))
# Evaluation
accu_svm <- mean(dat_test$emotion_idx == pred_svm)
confusionMatrix(pred_svm, dat_test$emotion_idx)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
##         1  12  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0
##         2   0 12  0  0  0  0  0  1  3  0  0  0  0  0  0  0  0  0  0  0  0
##         3   2  0 12  0  0  0  0  0  0  1  0  0  0  0  0  1  1  0  2  0  0  2
##         4   1  0  0 11  1  2  0  0  0  3  5  1  5  0  0  1  0  0  0  0  1  0
##         5   0  0  0  0 19  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
##         6   1  1  1  3  0 11  0  0  0  0  1  2  0  0  0  0  0  0  0  0  0  2
##         7   0  0  0  0  2  0 14  0  0  0  0  0  0  0  0  0  2  0  1  0  3  0
##         8   0  2  0  0  1  0  1 25  0  0  0  0  0  0  0  0  2  0  0  0  0  0
##         9   0  2  0  0  0  1  0  0 22  0  0  0  0  0  0  0  0  0  1  2  0  0
##         10  0  0  2  2  0  0  0  0  0  7  1  1  0  0  0  1  0  0  0  0  0  3
##         11  0  0  1  0  0  1  0  0  0  1 14  2  2  0  0  0  0  0  0  0  0  0
##         12  0  0  0  1  0  3  0  0  0  0  3 14  3  0  0  0  0  0  0  0  0  2
##         13  0  0  2  1  0  1  0  0  0  3  2  6  4  0  0  0  0  0  1  0  0  1
##         14  0  0  0  0  0  1  0  0  1  0  0  1  2 18  3  0  1  0  2  0  1  0
##         15  0  0  0  0  0  0  1  0  0  0  0  0  0  2 12  0  0  1  4  0  2  0
##         16  0  0  2  0  0  0  1  0  0  0  1  0  2  0  0 17  0  0  1  1  0  0
##         17  0  0  0  0  3  0  0  1  0  0  0  0  0  0  0  0  6  2  1  2  0  0
##         18  0  0  0  0  3  0  1  0  0  0  0  0  0  1  1  3  8 12  1  0  0  0
##         19  0  0  2  0  0  0  0  0  0  1  0  0  0  0  0  0  1  2  8  1  2  1
##         20  0  0  0  0  0  0  1  0  2  0  0  0  0  1  0  0  1  0  4  7  4  2
##         21  0  0  0  0  0  1  3  0  0  0  0  0  2  0  1  0  1  0  1  6  8  5
##         22  0  0  2  0  0  0  0  0  0  0  1  1  0  0  0  1  0  0  3  3  2  4
##
## Overall Statistics
##
##                Accuracy : 0.538
##                  95% CI : (0.4932, 0.5824)
##     No Information Rate : 0.062
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5158
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.7500   0.7059   0.4800   0.5789   0.6552   0.5238
## Specificity            0.9917   0.9917   0.9811   0.9584   0.9958   0.9770
## Pos Pred Value         0.7500   0.7500   0.5714   0.3548   0.9048   0.5000
## Neg Pred Value         0.9917   0.9897   0.9729   0.9829   0.9791   0.9791
## Prevalence             0.0320   0.0340   0.0500   0.0380   0.0580   0.0420
## Detection Rate         0.0240   0.0240   0.0240   0.0220   0.0380   0.0220
## Detection Prevalence   0.0320   0.0320   0.0420   0.0620   0.0420   0.0440
## Balanced Accuracy      0.8709   0.8488   0.7305   0.7687   0.8255   0.7504
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.6087   0.9259   0.7857    0.4375    0.5000    0.5000
## Specificity            0.9832   0.9873   0.9873    0.9793    0.9852    0.9746
## Pos Pred Value         0.6364   0.8065   0.7857    0.4118    0.6667    0.5385
## Neg Pred Value         0.9812   0.9957   0.9873    0.9814    0.9708    0.9705
## Prevalence             0.0460   0.0540   0.0560    0.0320    0.0560    0.0560
## Detection Rate         0.0280   0.0500   0.0440    0.0140    0.0280    0.0280
## Detection Prevalence   0.0440   0.0620   0.0560    0.0340    0.0420    0.0520
```

```
## Balanced Accuracy      0.7960   0.9566   0.8865   0.7084   0.7426   0.7373
##                     Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity            0.2000    0.8182    0.7059    0.7083    0.2609
## Specificity            0.9646    0.9749    0.9793    0.9832    0.9811
## Pos Pred Value         0.1905    0.6000    0.5455    0.6800    0.4000
## Neg Pred Value         0.9666    0.9915    0.9895    0.9853    0.9649
## Prevalence             0.0400    0.0440    0.0340    0.0480    0.0460
## Detection Rate         0.0080    0.0360    0.0240    0.0340    0.0120
## Detection Prevalence   0.0420    0.0600    0.0440    0.0500    0.0300
## Balanced Accuracy      0.5823    0.8965    0.8426    0.8458    0.6210
##                     Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity            0.6316    0.2581    0.3182    0.3478    0.1818
## Specificity            0.9626    0.9787    0.9686    0.9581    0.9728
## Pos Pred Value         0.4000    0.4444    0.3182    0.2857    0.2353
## Neg Pred Value         0.9851    0.9523    0.9686    0.9682    0.9627
## Prevalence             0.0380    0.0620    0.0440    0.0460    0.0440
## Detection Rate         0.0240    0.0160    0.0140    0.0160    0.0080
## Detection Prevalence   0.0600    0.0360    0.0440    0.0560    0.0340
## Balanced Accuracy      0.7971    0.6184    0.6434    0.6529    0.5773
```

```
cat("The accuracy of model: cost =", cost[which.min(err_svm[,1])], "is", accu_svm*100, "%.\n")
```

```
## The accuracy of model: cost = 0.002 is 53.8 %.
```

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 improved model features=", tm_feature_train_improved[1], "s \n")
```

```
## Time for constructing training improved model features= 59.6 s
```

```
cat("Time for constructing testing improved model features=", tm_feature_test_improved[1], "s \n")
```

```
## Time for constructing testing improved model features= 3.11 s
```

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

```
## Time for training improved model= 163.73 s
```

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

```
## Time for testing improved model= 7.34 s
```

```
labels_prediction <- data.frame(index=1:500,Baseline=pred_gbm,Advanced=pred_svm)
write.csv(labels_prediction,'../output/labels_prediction.csv')
```

###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.