Main

Fall2019-proj3-grp7

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.

```r
if(!require("EBImage")){
  source("https://bioconductor.org/biocLite.R")
  biocLite("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("OpenImageR")){
  install.packages("OpenImageR")
}

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

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

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

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

if(!require("class")){
  install.packages("class")
}
if(!require("MASS")){
  install.packages("MASS")
}
if(!require("e1071")){
  install.packages("e1071")
}
library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(OpenImageR)
library(FSelectorRcpp)
library(mlr)
library(kernlab)
library(gbm)
library(class)

library("e1071")
library("MASS")
```

Step 0 set work directories, extract paths, summarize

```r
set.seed(0)
# setwd("~/Desktop/5243/Project 3/fall2019-proj3-sec2--grp7-master/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
# setwd("../doc")
```

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

- (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=FALSE # 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.feature.test.test=FALSE # process features for test_test set
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications. In this Starter Code, we tune parameter k (number of neighbours) for KNN.

Step 2: import data and train-test split

```
#train-test split
info <- read.csv(train_label_path)
n <- nrow(info)
n_train <- round(n*(4/5), 0)
train_idx <- sample(info$Index, n_train, replace = F)
test_idx <- setdiff(info$Index,train_idx)
```

If you choose to extract features from images, such as using Gabor filter, R memory will exhaust all images are read together. The solution is to repeat reading a smaller batch(e.g 100) and process them.

```
n_files <- length(list.files(train_image_dir))

image_list <- list()
for(i in 1:100){
   image_list[[i]] <- readImage(paste0(train_image_dir, sprintf("%04d", i), ".jpg"))
}
```

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

```
#function to read fiducial points
#input: index
#output: matrix of fiducial points corresponding to the index
readMat.matrix <- function(index){
     return(round(readMat(paste0(train_pt_dir, sprintf("%04d", index), ".mat"))[[1]],0))
}

#load fiducial points
fiducial_pt_list <- lapply(1:n_files, readMat.matrix)
save(fiducial_pt_list, file="../output/fiducial_pt_list.RData")
```

Step 3: feature selection

`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){
  ## Distance calculation
  tm_feature_train <- system.time(dat_train <- feature_train(fiducial_pt_list, train_idx))
  dat_train <- cbind(dat_train, as.factor(info$emotion_idx[train_idx]))
  colnames(dat_train)[dim(dat_train)[2]] <- "emotion_idx"
  dat_train <- as.data.frame(dat_train)
  colnames(dat_train)<-make.names(colnames(dat_train),unique=T)

  ## Normalize
  tm_feature_train <- tm_feature_train +
    system.time(dat_train_stand <- feature_normalization(dat_train[,c(-dim(dat_train)[2])]))
 dat_train_stand <- cbind(dat_train_stand,dat_train$emotion_idx)
 colnames(dat_train_stand)[dim(dat_train_stand)[2]] <- "emotion_idx"

  ## Feature selection
  tm_feature_train <- tm_feature_train +
    system.time(feature_name <- feature_selection(dat_train_stand,"emotion_idx"))
  dat_train_selected <- dat_train[,feature_name]

  ## Calculate size from all selected distance
  tm_feature_train <- tm_feature_train +
    system.time(dat_train_double <- feature_selection_size(dat_train_selected))

  ## Add manually selected feature
  tm_feature_train <- tm_feature_train +
    system.time(dat_train_ratio <- manually_feature(fiducial_pt_list, train_idx))

  dat_train_selected_stand <- cbind(dat_train_selected,dat_train_double,dat_train_ratio)
  dat_train_selected_stand <- feature_normalization(dat_train_selected_stand)
  dat_train_selected_stand <- cbind(dat_train_selected_stand,dat_train$emotion_idx)
  colnames(dat_train_selected_stand)[dim(dat_train_selected_stand)[2]] <- "emotion_idx"

}

tm_feature_test <- NA
if(run.feature.test){
  ## This is the result from test

  feature_name <- c("point.7.to.point.21","point.10.to.point.13","point.10.to.point.33",
                    "point.11.to.point.49","point.12.to.point.55","point.14.to.point.18",
                    "point.23.to.point.50","point.34.to.point.46","point.50.to.point.62",
                    "point.59.to.point.62")

  ## Distance Feature
  tm_feature_test <- system.time(dat_test <- feature_train(fiducial_pt_list, test_idx))
```

```r
  dat_test <- cbind(dat_test, as.factor(info$emotion_idx[test_idx]))
  colnames(dat_test)[dim(dat_test)[2]] <- "emotion_idx"
  dat_test <- as.data.frame(dat_test)
  colnames(dat_test)<-make.names(colnames(dat_test),unique=T)
  dat_test_selected <- dat_test[,feature_name]

  ## Size Feature
  tm_feature_test <- tm_feature_test +
    system.time(dat_test_double <- feature_selection_size(dat_test_selected))

  ## Add manually selected feature
  tm_feature_test <- tm_feature_test +
    system.time(dat_test_ratio <- manually_feature(fiducial_pt_list, test_idx))

  dat_test_selected_stand <- cbind(dat_test_selected,dat_test_double,dat_test_ratio)
  dat_test_selected_stand <- feature_normalization(dat_test_selected_stand)
  dat_test_selected_stand <- cbind(dat_test_selected_stand,dat_test$emotion_idx)
  colnames(dat_test_selected_stand)[dim(dat_test_selected_stand)[2]] <- "emotion_idx"
}

tm_feature_test_test <- NA
if(run.feature.test.test){
  test_test_idx = c(1:2500)
  ## This is the result from test
  feature_name <- c("point.7.to.point.21","point.10.to.point.13","point.10.to.point.33",
                    "point.11.to.point.49","point.12.to.point.55","point.14.to.point.18",
                    "point.23.to.point.50","point.34.to.point.46","point.50.to.point.62",
                    "point.59.to.point.62")

  ## Distance Feature
  tm_feature_test_test <- system.time(dat_test_test <- feature_train(fiducial_pt_list, test_test_idx))
  colnames(dat_test_test)<-make.names(colnames(dat_test_test),unique=T)
  dat_test_test_selected <- dat_test_test[,feature_name]

  ## Size Feature
  tm_feature_test_test <- tm_feature_test_test +
    system.time(dat_test_test_double <- feature_selection_size(dat_test_test_selected))

  ## Add manually selected feature
  tm_feature_test_test <- tm_feature_test_test +
    system.time(dat_test_test_ratio <- manually_feature(fiducial_pt_list, test_test_idx))

  dat_test_selected_stand_test <- cbind(dat_test_test_selected,dat_test_test_double,
                                       dat_test_test_ratio)
  dat_test_selected_stand_test <- feature_normalization(dat_test_selected_stand_test)

}


##save(dat_train_selected_stand, file="../output/feature_train.RData")
## Because feature train takes over 10 hours, we do not knit this part.
## The previous feature selection train is already included in the output file.
save(dat_test_selected_stand, file="../output/feature_test.RData")
```

```r
##save(dat_test_selected_stand_test, file="../output/feature_test_test.RData")
```

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

```r
load("../output/feature_train.RData")
dat_train_selected <- dat_train_selected_ratio_stand55
dat_test_selected <- dat_test_selected_stand
```

Call the train models and test models from library:

- 1. KNN
- 2. LDA
- 3. SVM with radial kernel (improved model)
- 4. GBM with tree stumps (baseline Model)

1. KNN

- Do model selection by choosing among different values of training model parameters.
- Choose the "best" parameter value
- Train accuracy:
- KNN: Run test on test images
- evaluation
- Summarize Running Time

2. LDA

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

```r
source("../lib/train_lda.R")
tm_train=NA
tm_train <- system.time(fit_train <- train(dat_train_selected, par_best))
save(fit_train, file="../output/fit_train.RData")
```

- Train Accuracy:

```r
source("../lib/test_lda.R")
tm_test=NA
if(run.test){
  pred_train <- test(fit_train, dat_train_selected)
}

accu <- mean(dat_train_selected$emotion_idx == pred_train)
accu
```

```
## [1] 0.533
```

- LDA: Run test on test images

```
source("../lib/test_lda.R")
tm_test=NA
if(run.test){
  load(file="../output/fit_train.RData")
  tm_test <- system.time(pred <- test(fit_train, dat_test_selected))
}
```

- evaluation

```
accu <- mean(dat_test_selected$emotion_idx == pred)
cat("The accuracy of model:", "is", accu*100, "%.\n")
```

```
## The accuracy of model: is 47.6 %.
```

```
library(caret)
confusionMatrix(pred, dat_test_selected$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
##         1  15  0  2  0  1  1  3  0  0  1  0  1  1  0  0  0  0  1  1  0  1
##         2   0 17  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3   5  0 11  3  0  1  0  0  0  2  0  0  2  0  0  0  0  0  0  0  0
##         4   0  0  0 10  0  0  0  0  0  2  2  0  2  0  0  0  0  0  0  1  0
##         5   0  0  0  0 15  0  0  2  0  0  0  0  0  0  0  0  2  0  0  0  0
##         6   5  0  2  1  1 13  0  0  1  6  4  7  8  1  1  1  3  0  0  2  2
##         7   0  0  0  0  0  0 13  0  1  0  0  0  0  1  1  0  1  2  1  2  0
##         8   0  2  0  0  0  0  0 15  0  0  0  0  0  0  0  0  1  0  0  1  0
##         9   0  7  0  0  0  0  0  1 12  0  0  0  0  0  0  0  0  0  0  0  1
##         10  0  0  2  2  0  0  0  0  0 12  3  1  2  0  0  1  0  0  1  0  2
##         11  0  0  0  0  0  0  0  0  0  0  8  1  0  0  0  0  0  0  0  0  0
##         12  1  0  0  1  0  2  0  0  0  1  3 10  4  0  0  0  0  0  0  0  0
##         13  0  0  1  6  0  1  0  0  0  2  1  3  3  0  1  0  0  0  0  0  0
##         14  0  0  0  0  0  2  0  0  0  0  0  1  0 13  4  0  1  0  0  1  3
##         15  0  0  0  0  1  0  0  0  0  0  0  0  0  5 10  0  0  0  1  1  0
##         16  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0 14  3  2  0  0  0
##         17  0  0  0  0  0  0  1  1  0  0  0  0  0  0  2  0  9  2  1  2  2
##         18  0  0  0  0  2  0  0  0  0  0  0  0  0  1  0  1  6 10  0  1  0
##         19  0  0  0  0  0  0  2  0  0  0  0  0  0  0  2  0  1  3  5  3  1
##         20  0  1  0  0  0  0  0  0  1  0  0  0  0  1  0  1  2  1  1  6  0
##         21  0  1  0  0  0  1  4  0  1  0  0  0  0  0  1  0  1  2  3  3 12
##         22  0  0  0  2  0  0  0  0  0  0  2  0  0  0  0  1  1  0  0  1  0
##           Reference
## Prediction 22
##         1   0
##         2   1
```

```
##          3   6
##          4   0
##          5   0
##          6   8
##          7   0
##          8   0
##          9   0
##          10  0
##          11  0
##          12  0
##          13  0
##          14  0
##          15  0
##          16  2
##          17  0
##          18  1
##          19  1
##          20  0
##          21  2
##          22  5
##
## Overall Statistics
##
##                Accuracy : 0.476
##                  95% CI : (0.4315, 0.5208)
##     No Information Rate : 0.062
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4512
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.5769   0.6071   0.5500   0.4000   0.7500   0.6190
## Specificity            0.9726   0.9936   0.9604   0.9853   0.9917   0.8894
## Pos Pred Value         0.5357   0.8500   0.3667   0.5882   0.7895   0.1970
## Neg Pred Value         0.9767   0.9771   0.9809   0.9689   0.9896   0.9816
## Prevalence             0.0520   0.0560   0.0400   0.0500   0.0400   0.0420
## Detection Rate         0.0300   0.0340   0.0220   0.0200   0.0300   0.0260
## Detection Prevalence   0.0560   0.0400   0.0600   0.0340   0.0380   0.1320
## Balanced Accuracy      0.7747   0.8004   0.7552   0.6926   0.8708   0.7542
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity            0.5652   0.7143   0.7500    0.4615    0.3478
## Specificity            0.9811   0.9916   0.9814    0.9705    0.9979
## Pos Pred Value         0.5909   0.7895   0.5714    0.4615    0.8889
## Neg Pred Value         0.9791   0.9875   0.9916    0.9705    0.9695
## Prevalence             0.0460   0.0420   0.0320    0.0520    0.0460
## Detection Rate         0.0260   0.0300   0.0240    0.0240    0.0160
## Detection Prevalence   0.0440   0.0380   0.0420    0.0520    0.0180
## Balanced Accuracy      0.7732   0.8530   0.8657    0.7160    0.6729
##                      Class: 12 Class: 13 Class: 14 Class: 15 Class: 16
## Sensitivity             0.4167    0.1364    0.5909    0.4545    0.7368
```

```
## Specificity            0.9748    0.9686    0.9749    0.9833    0.9813
## Pos Pred Value         0.4545    0.1667    0.5200    0.5556    0.6087
## Neg Pred Value         0.9707    0.9606    0.9811    0.9751    0.9895
## Prevalence             0.0480    0.0440    0.0440    0.0440    0.0380
## Detection Rate         0.0200    0.0060    0.0260    0.0200    0.0280
## Detection Prevalence   0.0440    0.0360    0.0500    0.0360    0.0460
## Balanced Accuracy      0.6957    0.5525    0.7829    0.7189    0.8591
##                     Class: 17 Class: 18 Class: 19 Class: 20 Class: 21
## Sensitivity            0.2903    0.4348    0.3571    0.2500    0.5000
## Specificity            0.9765    0.9748    0.9733    0.9832    0.9601
## Pos Pred Value         0.4500    0.4545    0.2778    0.4286    0.3871
## Neg Pred Value         0.9542    0.9728    0.9813    0.9630    0.9744
## Prevalence             0.0620    0.0460    0.0280    0.0480    0.0480
## Detection Rate         0.0180    0.0200    0.0100    0.0120    0.0240
## Detection Prevalence   0.0400    0.0440    0.0360    0.0280    0.0620
## Balanced Accuracy      0.6334    0.7048    0.6652    0.6166    0.7300
##                     Class: 22
## Sensitivity            0.1923
## Specificity            0.9852
## Pos Pred Value         0.4167
## Neg Pred Value         0.9570
## Prevalence             0.0520
## Detection Rate         0.0100
## Detection Prevalence   0.0240
## Balanced Accuracy      0.5888
```

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.

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

```
## Time for training model= 0.077 s
```

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

```
## Time for testing model= 0.005 s
```

3. SVM (*improved model*)

- Tune the SVM model with cross-validation:

```r
## must have selected features first
tm_train=NA
tm_train <- system.time(tuned_parameters <- tune.svm(emotion_idx~.,
                                                     data = dat_train_selected,
                                                     gamma = 10^(-5:-1),
                                                     cost = c(30,35,40),
                                                     tunecontrol = tune.control(cross = 12)
                                                     ))
summary(tuned_parameters)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 12-fold cross validation
##
## - best parameters:
##  gamma cost
##  0.001   35
##
## - best performance: 0.5394843
##
## - Detailed performance results:
##    gamma cost     error dispersion
## 1  1e-05   30 0.7414899 0.01624426
## 2  1e-04   30 0.5805437 0.03553754
## 3  1e-03   30 0.5409873 0.02619165
## 4  1e-02   30 0.5654865 0.03189845
## 5  1e-01   30 0.6474581 0.04071884
## 6  1e-05   35 0.7314918 0.01769932
## 7  1e-04   35 0.5765307 0.03410334
## 8  1e-03   35 0.5394843 0.02718324
## 9  1e-02   35 0.5714896 0.02705938
## 10 1e-01   35 0.6474581 0.04071884
## 11 1e-05   40 0.7164797 0.02016448
## 12 1e-04   40 0.5735156 0.03079428
## 13 1e-03   40 0.5414893 0.02932813
## 14 1e-02   40 0.5759986 0.02377225
## 15 1e-01   40 0.6474581 0.04071884
```

- Train the model

```r
source("../lib/train_svm.R")
par_best=NULL
fit_train_final_svm <- train(dat_train_selected, tuned_parameters$best.parameters)
save(fit_train_final_svm, file="../output/fit_train_final.RData")
```

- Train accuracy:

```r
source("../lib/test_svm.R")
load("../output/fit_train_final.RData")

if(run.test){
  pred_train <- test(fit_train_final_svm, dat_train_selected)
}

accu <- mean(dat_train_selected$emotion_idx == pred_train)
accu
```

```
## [1] 0.5725
```

- SVM: Run test on test images

```r
source("../lib/test_svm.R")
tm_test=NA
if(run.test){
  load(file="../output/fit_train.RData")
  tm_test <- system.time(pred <- test(fit_train_final_svm, dat_test_selected))
}
```

************ SVM: Run test_test on test images

```r
source("../lib/test_svm.R")
tm_test=NA
if(run.test){
  load(file="../output/fit_train_final.RData")
  tm_test <- system.time(pred <- test(fit_train_final_svm, dat_test_selected))
}
```

- evaluation

```r
accu <- mean(dat_test_selected$emotion_idx == pred)
cat("The accuracy of model:", "is", accu*100, "%.\n")
```

```
## The accuracy of model: is 55 %.
```

```r
library(caret)
confusionMatrix(pred, dat_test_selected$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
##         1  20  0  2  0  0  0  1  0  0  0  0  0  1  0  0  0  0  1  0  0  0
##         2   0 23  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  1  0  0
##         3   2  0 13  3  0  3  0  0  0  1  1  0  1  0  0  0  0  0  1  0  0
##         4   0  0  2 15  0  0  0  0  0  5  4  2  8  0  0  0  0  0  0  0  0
##         5   0  0  0  0 17  0  0  0  0  0  0  0  0  0  0  0  4  1  0  0  0
##         6   0  0  0  1  0  7  1  0  0  1  1  0  3  0  0  2  0  0  0  0  0
##         7   1  0  0  0  0  0  9  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         8   0  2  0  0  0  0  1 17  0  0  0  0  0  0  0  0  2  0  0  1  0
##         9   0  3  0  0  0  0  0  1 14  0  0  0  0  0  0  0  0  0  0  1  1
##         10  0  0  1  4  0  0  0  0  0 15  4  2  2  0  1  1  0  0  1  0  2
##         11  0  0  0  0  0  1  0  0  0  1  9  1  1  0  0  0  0  0  0  0  0
##         12  1  0  0  1  0  5  0  0  0  0  2 13  0  1  0  0  0  0  0  0  0
##         13  2  0  1  1  0  2  0  0  0  3  2  3  4  0  0  0  0  0  0  0  1
##         14  0  0  0  0  0  3  0  0  0  0  0  1  0 17  3  0  1  0  1  2  5
##         15  0  0  0  0  1  0  0  0  0  0  0  0  1  3 16  0  0  0  2  0  0
##         16  0  0  1  0  0  0  0  0  0  0  0  1  0  0  0 14  1  1  0  0  1
##         17  0  0  0  0  0  0  2  2  0  0  0  0  0  1  0  0 13  1  0  2  2
##         18  0  0  0  0  2  0  1  0  0  0  0  0  0  0  0  1  4 13  1  0  0
##         19  0  0  0  0  0  0  2  0  0  0  0  0  0  0  2  0  1  3  2  3  0
##         20  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  5  2  2  8  2
##         21  0  0  0  0  0  0  6  0  0  0  0  1  0  0  0  1  0  1  3  4 10
```

```
##          22  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  3  0
##            Reference
## Prediction 22
##         1   0
##         2   1
##         3   3
##         4   1
##         5   0
##         6   1
##         7   0
##         8   0
##         9   0
##         10  3
##         11  2
##         12  0
##         13  0
##         14  1
##         15  0
##         16  3
##         17  1
##         18  0
##         19  2
##         20  1
##         21  1
##         22  6
##
## Overall Statistics
##
##                Accuracy : 0.55
##                  95% CI : (0.5052, 0.5942)
##     No Information Rate : 0.062
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5283
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.7692   0.8214   0.6500   0.6000   0.8500   0.3333
## Specificity            0.9895   0.9915   0.9688   0.9537   0.9896   0.9791
## Pos Pred Value         0.8000   0.8519   0.4643   0.4054   0.7727   0.4118
## Neg Pred Value         0.9874   0.9894   0.9852   0.9784   0.9937   0.9710
## Prevalence             0.0520   0.0560   0.0400   0.0500   0.0400   0.0420
## Detection Rate         0.0400   0.0460   0.0260   0.0300   0.0340   0.0140
## Detection Prevalence   0.0500   0.0540   0.0560   0.0740   0.0440   0.0340
## Balanced Accuracy      0.8793   0.9065   0.8094   0.7768   0.9198   0.6562
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity            0.3913   0.8095   0.8750    0.5769    0.3913
## Specificity            0.9979   0.9875   0.9876    0.9557    0.9874
## Pos Pred Value         0.9000   0.7391   0.7000    0.4167    0.6000
## Neg Pred Value         0.9714   0.9916   0.9958    0.9763    0.9711
## Prevalence             0.0460   0.0420   0.0320    0.0520    0.0460
```

```
## Detection Rate         0.0180   0.0340   0.0280    0.0300    0.0180
## Detection Prevalence   0.0200   0.0460   0.0400    0.0720    0.0300
## Balanced Accuracy      0.6946   0.8985   0.9313    0.7663    0.6894
##                      Class: 12 Class: 13 Class: 14 Class: 15 Class: 16
## Sensitivity             0.5417    0.1818    0.7727    0.7273    0.7368
## Specificity             0.9790    0.9686    0.9644    0.9854    0.9834
## Pos Pred Value          0.5652    0.2105    0.5000    0.6957    0.6364
## Neg Pred Value          0.9769    0.9626    0.9893    0.9874    0.9895
## Prevalence              0.0480    0.0440    0.0440    0.0440    0.0380
## Detection Rate          0.0260    0.0080    0.0340    0.0320    0.0280
## Detection Prevalence    0.0460    0.0380    0.0680    0.0460    0.0440
## Balanced Accuracy       0.7603    0.5752    0.8686    0.8563    0.8601
##                      Class: 17 Class: 18 Class: 19 Class: 20 Class: 21
## Sensitivity             0.4194    0.5652    0.1429    0.3333    0.4167
## Specificity             0.9765    0.9811    0.9733    0.9727    0.9643
## Pos Pred Value          0.5417    0.5909    0.1333    0.3810    0.3704
## Neg Pred Value          0.9622    0.9791    0.9753    0.9666    0.9704
## Prevalence              0.0620    0.0460    0.0280    0.0480    0.0480
## Detection Rate          0.0260    0.0260    0.0040    0.0160    0.0200
## Detection Prevalence    0.0480    0.0440    0.0300    0.0420    0.0540
## Balanced Accuracy       0.6980    0.7732    0.5581    0.6530    0.6905
##                      Class: 22
## Sensitivity             0.2308
## Specificity             0.9916
## Pos Pred Value          0.6000
## Neg Pred Value          0.9592
## Prevalence              0.0520
## Detection Rate          0.0120
## Detection Prevalence    0.0200
## Balanced Accuracy       0.6112
```

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 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")
```

```
## Time for training model= 259.809 s
```

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

```
## Time for testing model= 0.119 s
```

4. GBM (*Baseline Model*)

- Tune GBM.

```r
hyper_grid <- expand.grid(
  shrinkage = c(.001, .01),
  interaction.depth = c(1, 3),
  n.minobsinnode = c(5, 10),
  bag.fraction = c(.65, .8),
  optimal_trees = 0,               # a place to dump results
  min_RMSE = 0                     # a place to dump results
)

# randomize data
random_index <- sample(1:nrow(dat_train_selected), nrow(dat_train_selected))
random_train <- dat_train_selected[random_index, ]

# grid search
for(i in 1:nrow(hyper_grid)) {

  # reproducibility
  set.seed(123)

  # train model
  gbm.tune <- gbm(
    formula =  emotion_idx~.,
    distribution = "multinomial",
    data = random_train,
    n.trees = 100,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    train.fraction = .75,
    n.cores = NULL, # will use all cores by default
    verbose = FALSE
  )

  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid$min_RMSE[i] <- sqrt(min(gbm.tune$valid.error))
}

hyper_grid %>%
  dplyr::arrange(min_RMSE) %>%
  head(10)
```

```
##    shrinkage interaction.depth n.minobsinnode bag.fraction optimal_trees
## 1      0.010                 3             10         0.65           100
## 2      0.010                 3             10         0.80           100
## 3      0.010                 3              5         0.65           100
## 4      0.010                 3              5         0.80           100
## 5      0.010                 1             10         0.65           100
## 6      0.010                 1              5         0.65           100
## 7      0.010                 1             10         0.80           100
## 8      0.010                 1              5         0.80           100
## 9      0.001                 3             10         0.65           100
```

```
## 10     0.001                  3               5          0.65            100
##    min_RMSE
## 1  1.487001
## 2  1.491959
## 3  1.492670
## 4  1.497072
## 5  1.581901
## 6  1.583199
## 7  1.583916
## 8  1.584736
## 9  1.690817
## 10 1.692194
```

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

```
source("../lib/train_gbm.R")
tm_train=NA
tm_train <- system.time(fit_train_baseline <- train(dat_train_selected, par = NULL))
save(fit_train_baseline, file="../output/fit_train_baseline_final.RData")
```

- Train Error:

```
source("../lib/test_gbm.R")
load("../output/fit_train_baseline_final.RData")

tm_test=NA
if(run.test){
  tm_test <- system.time(pred_train <- test(fit_train_baseline, dat_train_selected))
}

labels = colnames(pred_train)[apply(pred_train, 1, which.max)]
accu <- mean(dat_train_selected$emotion_idx == labels)
accu
```

```
## [1] 0.7705
```

- GBM: Run test on test images

```
source("../lib/test_gbm.R")
tm_test=NA
if(run.test){
  load(file="../output/fit_train.RData")
  tm_test <- system.time(pred <- test(fit_train_baseline, dat_test_selected))
}
```

- GBM: Run test_test on test images

```
source("../lib/test_gbm.R")
tm_test_test=NA
if(run.feature.test.test){
  load(file="../output/fit_train_baseline_final.RData")
  tm_test <- system.time(pred <- test(fit_train_baseline, dat_test_selected))
}
```

- evaluation

```r
labels = colnames(pred)[apply(pred, 1, which.max)]
accu <- mean(dat_test_selected$emotion_idx == labels)
cat("The accuracy of model:", "is", accu*100, "%.\n")
```

```
## The accuracy of model: is 59.6 %.
```

```r
library(caret)
confusionMatrix(as.factor(labels), dat_test_selected$emotion_idx)
```

```
## Warning in confusionMatrix.default(as.factor(labels),
## dat_test_selected$emotion_idx): Levels are not in the same order for
## reference and data. Refactoring data to match.
```

```
## 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
##         1  22  0  3  1  0  1  0  0  0  1  1  2  2  0  0  1  1  0  0  0  0
##         2   0 19  0  0  0  0  0  1  4  0  0  0  0  0  0  0  0  0  1  0  0
##         3   1  0 14  2  0  1  0  0  0  4  3  0  1  0  0  0  0  0  1  0  0
##         4   0  0  0 17  0  0  0  0  0  2  1  1  5  0  0  0  0  0  0  0  0
##         5   0  0  0  0 18  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0
##         6   1  0  0  0  0 10  0  0  1  2  1  2  1  0  0  0  0  0  0  0  0
##         7   1  0  0  0  0  0 14  0  0  0  0  0  0  0  1  0  1  1  1  1  1
##         8   0  1  0  0  0  0  2 16  0  0  0  0  0  0  0  0  1  0  0  1  0
##         9   0  7  0  0  0  0  0  1 11  0  0  0  0  0  0  0  0  0  0  1  2
##         10  0  0  1  4  0  0  0  0  0 16  2  2  2  0  0  1  0  0  1  0  1
##         11  0  0  1  0  0  1  0  0  0  0 14  3  2  0  0  0  0  0  0  0  0
##         12  1  0  0  0  0  4  0  0  0  0  1 11  1  0  0  0  0  0  0  0  0
##         13  0  0  0  1  0  1  0  0  0  0  0  0  6  0  0  0  0  0  0  0  1
##         14  0  0  0  0  0  2  0  0  0  0  0  0  0 17  3  0  1  0  1  1  2
##         15  0  0  0  0  1  0  0  0  0  0  0  0  1  3 15  0  1  0  2  0  0
##         16  0  0  1  0  0  0  0  0  0  0  0  1  0  0  0 15  2  2  0  0  0
##         17  0  0  0  0  1  0  0  1  0  0  0  0  0  1  0  0 15  1  0  1  0
##         18  0  0  0  0  0  0  1  0  0  0  0  0  0  0  1  1  5 13  0  0  0
##         19  0  0  0  0  0  0  2  0  0  0  0  0  0  0  1  1  0  3  5  2  2
##         20  0  1  0  0  0  0  1  2  0  0  0  0  1  1  0  0  2  2  1 13  5
##         21  0  0  0  0  0  1  3  0  0  0  0  0  0  0  1  0  1  0  1  4 10
##         22  0  0  0  0  0  0  0  0  0  1  0  2  0  0  0  0  0  0  0  0  0
##           Reference
## Prediction 22
##         1   1
##         2   0
##         3   3
##         4   0
##         5   0
##         6   0
##         7   0
##         8   0
##         9   1
```

```
##         10  1
##         11  1
##         12  1
##         13  0
##         14  0
##         15  0
##         16  2
##         17  1
##         18  0
##         19  3
##         20  0
##         21  5
##         22  7
##
## Overall Statistics
##
##                Accuracy : 0.596
##                  95% CI : (0.5515, 0.6393)
##     No Information Rate : 0.062
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5766
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.8462   0.6786   0.7000   0.6800   0.9000   0.4762
## Specificity            0.9705   0.9873   0.9667   0.9811   0.9958   0.9833
## Pos Pred Value         0.6111   0.7600   0.4667   0.6538   0.9000   0.5556
## Neg Pred Value         0.9914   0.9811   0.9872   0.9831   0.9958   0.9772
## Prevalence             0.0520   0.0560   0.0400   0.0500   0.0400   0.0420
## Detection Rate         0.0440   0.0380   0.0280   0.0340   0.0360   0.0200
## Detection Prevalence   0.0720   0.0500   0.0600   0.0520   0.0400   0.0360
## Balanced Accuracy      0.9083   0.8329   0.8333   0.8305   0.9479   0.7297
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity            0.6087   0.7619   0.6875    0.6154    0.6087
## Specificity            0.9853   0.9896   0.9752    0.9684    0.9832
## Pos Pred Value         0.6667   0.7619   0.4783    0.5161    0.6364
## Neg Pred Value         0.9812   0.9896   0.9895    0.9787    0.9812
## Prevalence             0.0460   0.0420   0.0320    0.0520    0.0460
## Detection Rate         0.0280   0.0320   0.0220    0.0320    0.0280
## Detection Prevalence   0.0420   0.0420   0.0460    0.0620    0.0440
## Balanced Accuracy      0.7970   0.8757   0.8314    0.7919    0.7960
##                      Class: 12 Class: 13 Class: 14 Class: 15 Class: 16
## Sensitivity             0.4583    0.2727    0.7727    0.6818    0.7895
## Specificity             0.9832    0.9937    0.9791    0.9833    0.9834
## Pos Pred Value          0.5789    0.6667    0.6296    0.6522    0.6522
## Neg Pred Value          0.9730    0.9674    0.9894    0.9853    0.9916
## Prevalence              0.0480    0.0440    0.0440    0.0440    0.0380
## Detection Rate          0.0220    0.0120    0.0340    0.0300    0.0300
## Detection Prevalence    0.0380    0.0180    0.0540    0.0460    0.0460
## Balanced Accuracy       0.7208    0.6332    0.8759    0.8325    0.8864
```

```
##                      Class: 17 Class: 18 Class: 19 Class: 20 Class: 21
## Sensitivity             0.4839    0.5652    0.3571    0.5417    0.4167
## Specificity             0.9872    0.9832    0.9712    0.9664    0.9664
## Pos Pred Value          0.7143    0.6190    0.2632    0.4483    0.3846
## Neg Pred Value          0.9666    0.9791    0.9813    0.9766    0.9705
## Prevalence              0.0620    0.0460    0.0280    0.0480    0.0480
## Detection Rate          0.0300    0.0260    0.0100    0.0260    0.0200
## Detection Prevalence    0.0420    0.0420    0.0380    0.0580    0.0520
## Balanced Accuracy       0.7355    0.7742    0.6642    0.7540    0.6915
##                      Class: 22
## Sensitivity             0.2692
## Specificity             0.9937
## Pos Pred Value          0.7000
## Neg Pred Value          0.9612
## Prevalence              0.0520
## Detection Rate          0.0140
## Detection Prevalence    0.0200
## Balanced Accuracy       0.6315
```

###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.