GBM model

Project3 Group4

```
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")
}
```

```
## Warning: package 'readxl' was built under R version 3.5.2
```

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

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

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

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

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

```
## Warning: package 'caret' was built under R version 3.5.2
```

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

```
## Warning: package 'gbm' was built under R version 3.5.2
```

```
library(class)
```

```
## Warning: package 'class' was built under R version 3.5.2
```

Step 0 set work directories

```
set.seed(0)
setwd("~/Documents/GitHub/Spring2020-Project3-group4/doc")
```

```
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=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 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: construct features and responses

- The follow plots show how pairwise distance between fiducial points can work as feature for facial emotion recognition.
- In the first column, 78 fiducials points of each emotion are marked in order.
- In the second column distributions of vertical distance between right pupil(1) and right brow peak(21) are shown in histograms. For example, the distance of an angry face tends to be shorter than that of a surprised face.
- The third column is the distributions of vertical distances between right mouth corner(50) and the midpoint of the upper lip(52). For example, the distance of an happy face tends to be shorter than that of a sad face.

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features and responses that are required by all the models you are going to evaluate later.

- `feature.R`
- Input: list of images or fiducial point
- Output: an RData file that contains extracted features and corresponding responses

```
source("../lib/feature.R")
tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(fiducial_pt_list, train_idx))
}

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

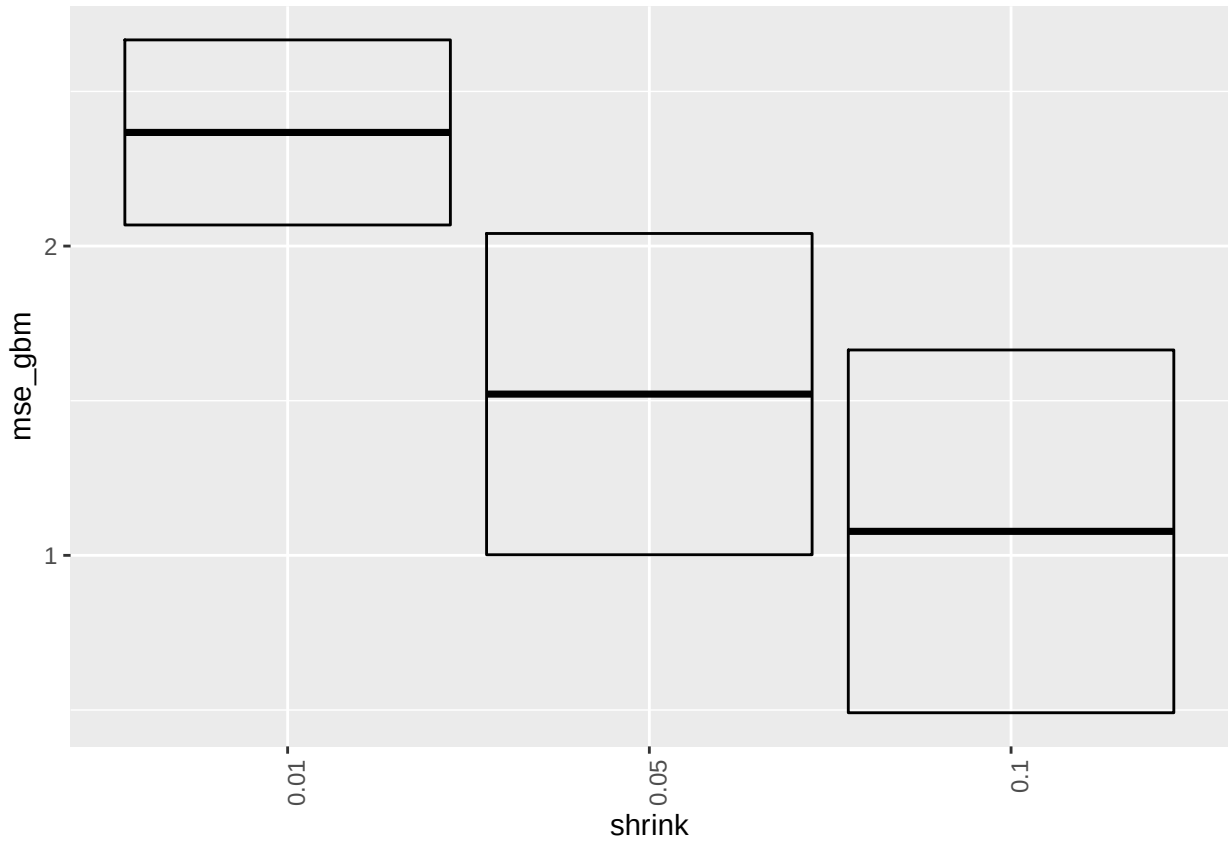
```
shrink = c(0.10,0.05,0.01)
model_labels = paste("GBM with Shrink =", shrink)
```

cross-validation to choose shrink parameter

```
#source("../lib/tuning_parameter_gbm.R")
#if(run.cv){
#     err_cv_gbm <- matrix(0, nrow = length(shrink), ncol = 2)
#     for(i in 1:length(shrink)){
#         cat("Shrink =", shrink[i], "\n")
#         err_cv_gbm[i,] <- cv.function.gbm(dat_train, shrink[i])
#         save(err_cv_gbm, file="../output/err_cv_gbm.RData")
#     }
# }
```

Visualize cross-validation results.

```
if(run.cv){
  load("../output/err_cv_gbm.RData")
  mse_cv_gbm <- as.data.frame(err_cv_gbm)
  colnames(mse_cv_gbm) <- c("mse_gbm", "sd_gbm")
  mse_cv_gbm$shrink = as.factor(shrink)
  mse_cv_gbm %>%
    ggplot(aes(x = shrink, y = mse_gbm,
               ymin = mse_gbm - sd_gbm, ymax = mse_gbm + sd_gbm)) +
    geom_crossbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
}
```



- Choose the "best" parameter value

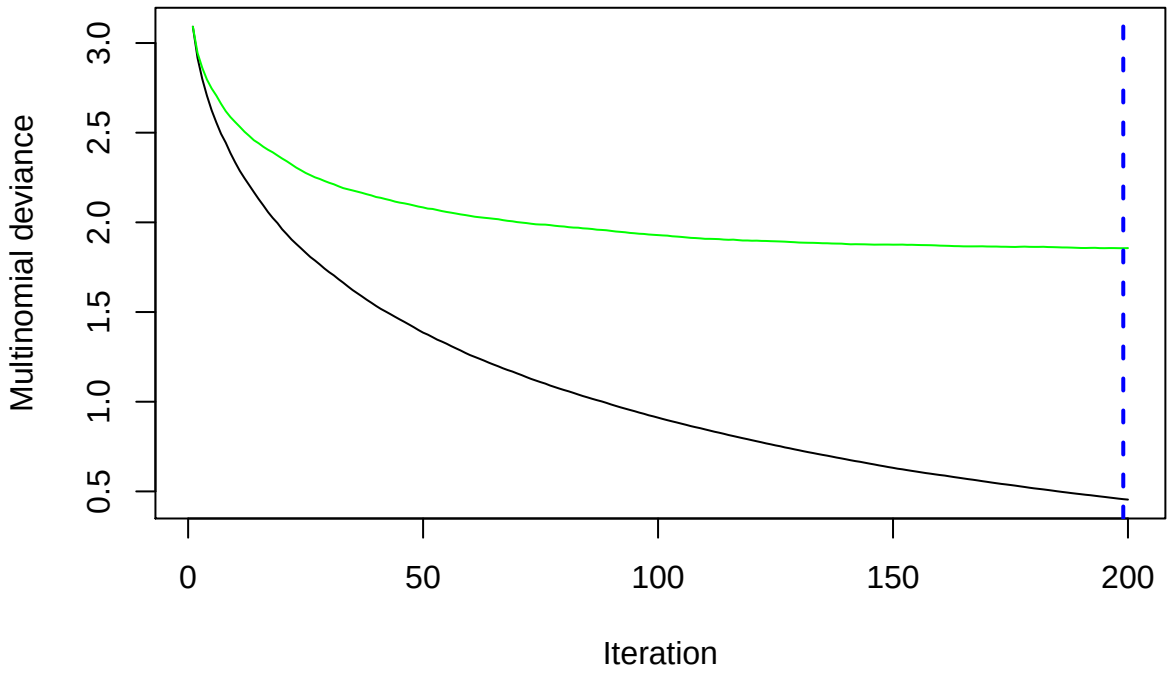
```
if(run.cv){
  model_best_gbm <- shrink[which.min(err_cv_gbm[,1])]
}
par_best_gbm <- list(shrink = model_best_gbm)
```

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

```
#Google Drive link: https://drive.google.com/file/d/16ZQ-hkR1sJURZNX_NIpXcOsRSNsXgwyC/view?usp=sharing
source("../lib/train_gbm.R")
###Traing
#gbm.fit<-gbm_train(dat_train)
#saveRDS(gbm.fit, "~/Desktop/Spring2020-Project3-group4/output/gbm.RDS")
```

Step 5: Run test on test images

```
source("../lib/test_gbm.R")
gbm.fit<-readRDS("../output/gbm.RDS")
pred_gbm<-gbm_test(gbm.fit[[1]],dat_test)
```



Evaluation

```
pred_class<-apply(pred_gbm[[1]],1,which.max)
confusionMatrix(dat_test$emotion_idx,as.factor(pred_class))
```

```
## 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  22  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         2   0 23  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3   2  0 24  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
##         4   0  0  0 18  0  1  1  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0
##         5   0  0  0  0 21  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         6   0  0  0  0  0 15  0  0  0  0  1  0  1  0  0  0  0  0  0  0  0  0
##         7   0  0  0  0  0  0 18  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
##         8   0  2  0  0  0  0  0 24  0  0  0  0  0  0  0  0  1  0  1  0  0  0
##         9   0  0  0  0  0  1  0  1 23  0  0  0  0  0  1  0  0  0  0  1  0  0
##         10  0  0  0  0  0  0  0  0  0 18  0  1  1  0  0  0  0  0  0  0  0  1
##         11  0  0  0  2  0  1  0  0  0  1 23  2  0  1  0  0  0  0  0  0  0  0
##         12  0  0  0  0  0  0  0  0  0  0  0 25  4  1  0  0  0  0  0  0  0  0
##         13  0  0  0  0  0  0  0  0  0  0  0  0 16  0  0  0  0  0  0  0  0  0
##         14  0  0  0  0  0  0  0  0  0  0  0  0  0 16  1  0  1  0  0  0  1  0
##         15  1  0  0  0  0  0  0  0  0  0  0  0  1  0  6  0  0  0  0  1  0  0
##         16  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0 22  0  0  0  0  0  0
##         17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 24  4  0  0  0  0
##         18  0  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  1 18  0  0  0  0
##         19  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  1 22  0  2  1
##         20  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1 16  0  0
##         21  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0 25  1
##         22  0  0  1  1  0  0  1  0  0  0  2  0  0  0  0  0  0  0  1  0  0 13
##
## Overall Statistics
```

```
##
##                Accuracy : 0.864
##                  95% CI : (0.8308, 0.8928)
##     No Information Rate : 0.06
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8572
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.8800   0.8846   0.8889   0.8182   0.9545   0.7895
## Specificity            0.9958   0.9958   0.9937   0.9916   1.0000   0.9958
## Pos Pred Value         0.9167   0.9200   0.8889   0.8182   1.0000   0.8824
## Neg Pred Value         0.9937   0.9937   0.9937   0.9916   0.9979   0.9917
## Prevalence             0.0500   0.0520   0.0540   0.0440   0.0440   0.0380
## Detection Rate         0.0440   0.0460   0.0480   0.0360   0.0420   0.0300
## Detection Prevalence   0.0480   0.0500   0.0540   0.0440   0.0420   0.0340
## Balanced Accuracy      0.9379   0.9402   0.9413   0.9049   0.9773   0.8927
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.8571   0.9231   0.9583    0.8571    0.8519    0.8333
## Specificity            0.9979   0.9916   0.9916    0.9937    0.9852    0.9894
## Pos Pred Value         0.9474   0.8571   0.8519    0.8571    0.7667    0.8333
## Neg Pred Value         0.9938   0.9958   0.9979    0.9937    0.9915    0.9894
## Prevalence             0.0420   0.0520   0.0480    0.0420    0.0540    0.0600
## Detection Rate         0.0360   0.0480   0.0460    0.0360    0.0460    0.0500
## Detection Prevalence   0.0380   0.0560   0.0540    0.0420    0.0600    0.0600
## Balanced Accuracy      0.9275   0.9573   0.9750    0.9254    0.9185    0.9113
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity             0.6957    0.8889    0.7500    1.0000    0.8889
## Specificity             1.0000    0.9938    0.9939    0.9958    0.9915
## Pos Pred Value          1.0000    0.8421    0.6667    0.9167    0.8571
## Neg Pred Value          0.9855    0.9958    0.9959    1.0000    0.9936
## Prevalence              0.0460    0.0360    0.0160    0.0440    0.0540
## Detection Rate          0.0320    0.0320    0.0120    0.0440    0.0480
## Detection Prevalence    0.0320    0.0380    0.0180    0.0480    0.0560
## Balanced Accuracy       0.8478    0.9413    0.8720    0.9979    0.9402
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity             0.7500    0.8800    0.8889    0.8621    0.8125
## Specificity             0.9937    0.9895    0.9938    0.9958    0.9876
## Pos Pred Value          0.8571    0.8148    0.8421    0.9259    0.6842
## Neg Pred Value          0.9875    0.9937    0.9958    0.9915    0.9938
## Prevalence              0.0480    0.0500    0.0360    0.0580    0.0320
## Detection Rate          0.0360    0.0440    0.0320    0.0500    0.0260
## Detection Prevalence    0.0420    0.0540    0.0380    0.0540    0.0380
## Balanced Accuracy       0.8718    0.9347    0.9413    0.9289    0.9001
```

```
acc_gbm<-mean(dat_test$emotion_idx==pred_class)
cat("The accuracy for gbm model is", acc_gbm*100, "%.\n")
```

```
## The accuracy for gbm model is 86.4 %.
```

Summarize Running Time

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

```
## Time for constructing training features= 0.791 s
```

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

```
## Time for constructing testing features= 0.167 s
```

```
cat("Time for training model=", gbm.fit[[2]][1], "s \n")
```

```
## Time for training model= 805.146 s
```

```
cat("Time for test model=", pred_gbm[[2]][1], "s \n")
```

```
## Time for test model= 15.143 s
```