GBM model

Code

project3 group4

Hide

```
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")
}
```

```
package 'readxl' was built under R version 3.5.2
```

Hide

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

```
package 'dplyr' was built under R version 3.5.2
```

Hide

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

```
package 'caret' was built under R version 3.5.2
```

Hide

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

```
package 'gbm' was built under R version 3.5.2
```

Step 0 set work directories

Hide

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

Hide

```
train_dir <- "../data/" # 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

Hide

```
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

Hide

```
#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.

Hide

```
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.

Hide

```
#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")
load("../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

Hide

```
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

Hide

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

cross-validation to choose shrink parameter

Hide

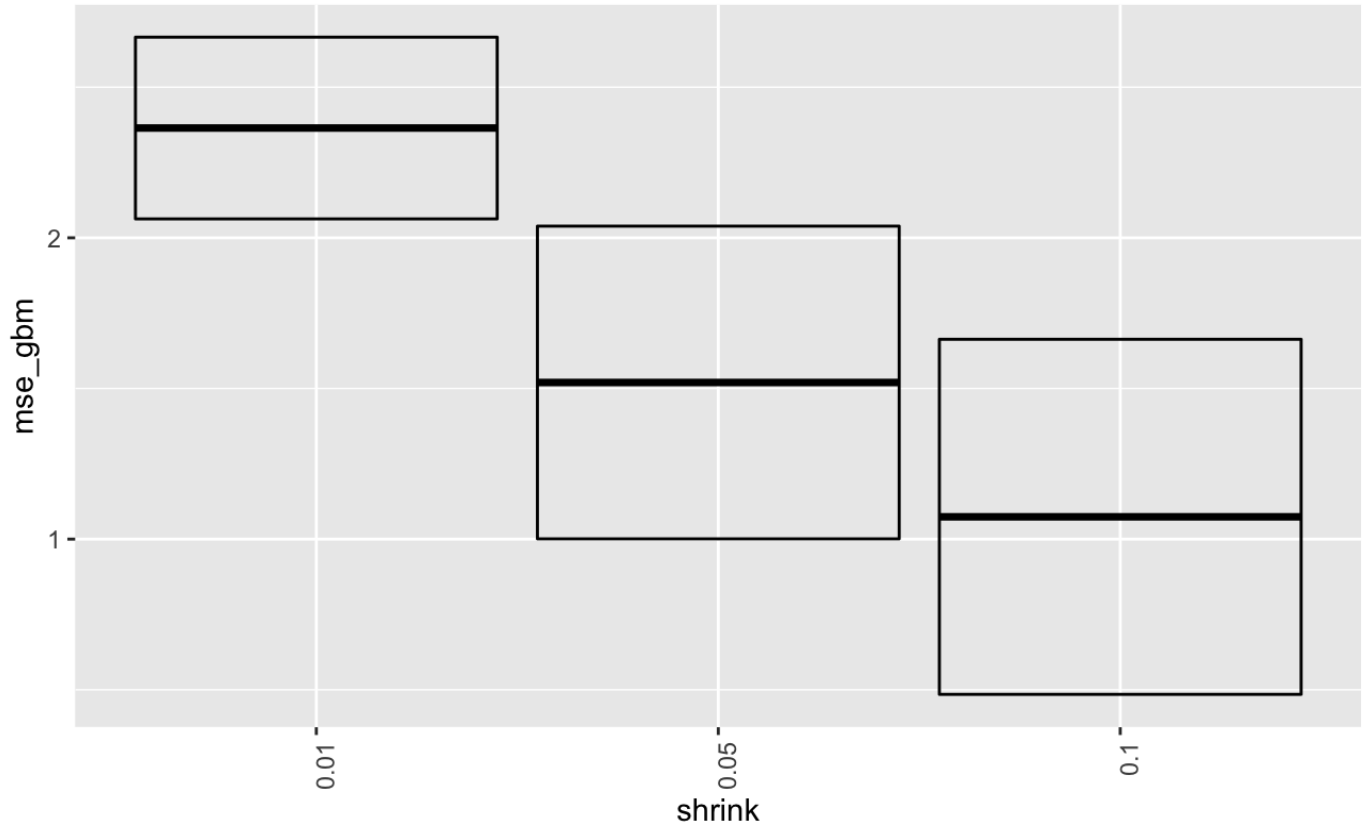
```
source("../lib/tuning_parameter_gbm.R")
if(run.cv){
     err_cv_gbm <- matrix(0, nrow = length(shrink), ncol = 2)
     for(i in 1:length(shrink)){
         cat("Shrink =", shrink[i], "\n")
         err_cv_gbm[i,] <- cv.function.gbm(dat_train, shrink[i])
#         save(err_cv_gbm, file="../output/err_cv_gbm.RData")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

```
Shrink = 0.1
[1] 1.0776051 0.5865022
Shrink = 0.05
[1] 1.5213593 0.5193543
Shrink = 0.01
[1] 2.3673650 0.2991708
```

Visualize cross-validation results.

Hide

```
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

Hide

```
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.

Hide

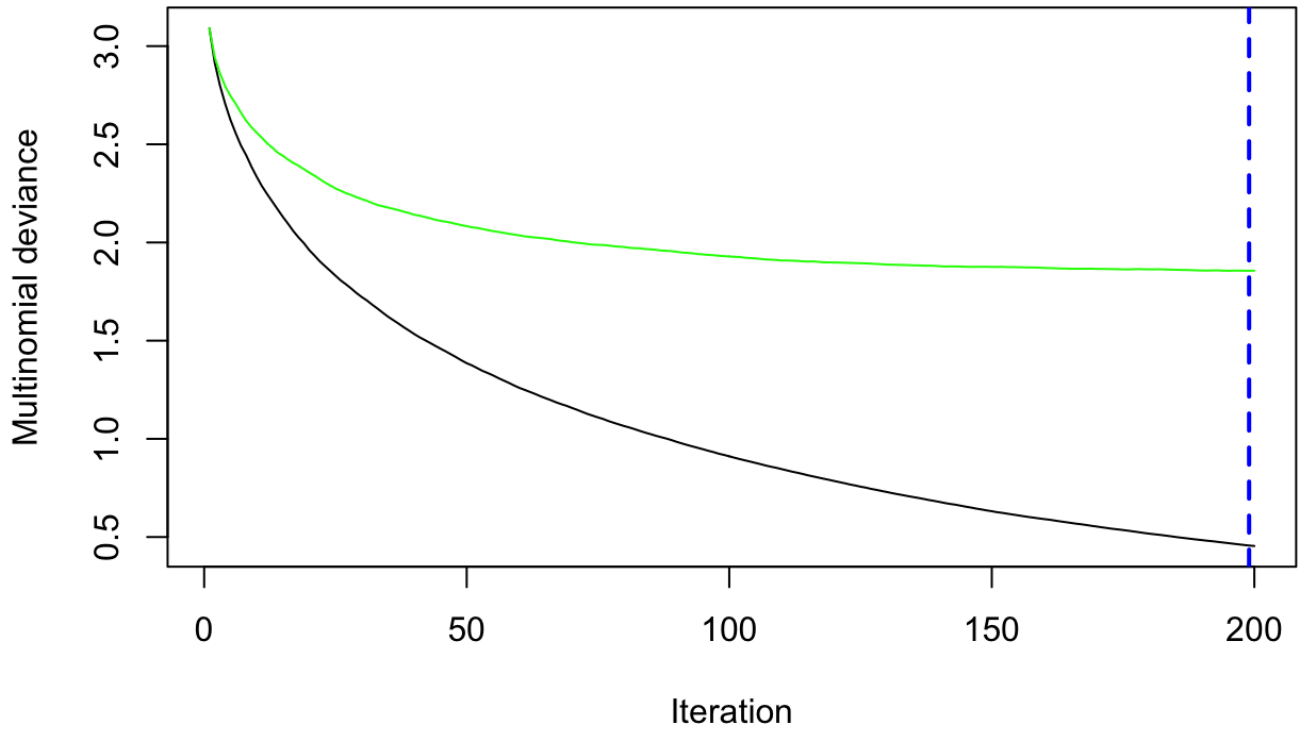
```
source("../lib/train_gbm.R")
###Traing
#gbm.fit<-gbm_train(dat_train)
#Save model
#saveRDS(gbm.fit, "../output/gbm.RDS")

#Google Drive link: https://drive.google.com/file/d/16ZQ-hkR1sJURZNX_NIpXcOsRSNsXgwyC/view?usp=sharing
#load model
gbm.fit<-readRDS("../output/gbm.RDS")
```

Step 5: Run test on test images

Hide

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



Evaluation

Hide

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

Overall Statistics

               Accuracy : 0.424
                 95% CI : (0.3802, 0.4687)
    No Information Rate : 0.07
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3963

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7
Sensitivity            0.6667   0.6923   0.5143   0.5263   0.6471   0.5789   0.5714
Specificity            0.9875   0.9852   0.9742   0.9584   0.9731   0.9792   0.9767
Pos Pred Value         0.7000   0.7200   0.6000   0.3333   0.4583   0.5238   0.5926
Neg Pred Value         0.9854   0.9832   0.9638   0.9809   0.9874   0.9833   0.9746
Prevalence             0.0420   0.0520   0.0700   0.0380   0.0340   0.0380   0.0560
Detection Rate         0.0280   0.0360   0.0360   0.0200   0.0220   0.0220   0.0320
Detection Prevalence   0.0400   0.0500   0.0600   0.0600   0.0480   0.0420   0.0540
Balanced Accuracy      0.8271   0.8388   0.7442   0.7424   0.8101   0.7791   0.7741
                     Class: 8 Class: 9 Class: 10 Class: 11 Class: 12 Class: 13
Sensitivity            0.7200   0.7000    0.3333    0.2000    0.2143    0.1364
Specificity            0.9832   0.9812    0.9687    0.9663    0.9767    0.9665
Pos Pred Value         0.6923   0.6087    0.3182    0.2381    0.3529    0.1579
Neg Pred Value         0.9852   0.9874    0.9707    0.9582    0.9545    0.9605
Prevalence             0.0500   0.0400    0.0420    0.0500    0.0560    0.0440
Detection Rate         0.0360   0.0280    0.0140    0.0100    0.0120    0.0060
Detection Prevalence   0.0520   0.0460    0.0440    0.0420    0.0340    0.0380
```

```
Balanced Accuracy      0.8516   0.8406    0.6510    0.5832    0.5955    0.5514
                     Class: 14 Class: 15 Class: 16 Class: 17 Class: 18 Class: 19
Sensitivity             0.5000    0.2941    0.6842    0.2609    0.2727    0.2222
Specificity             0.9807    0.9627    0.9854    0.9665    0.9770    0.9710
Pos Pred Value          0.6538    0.2174    0.6500    0.2727    0.3529    0.2222
Neg Pred Value          0.9641    0.9748    0.9875    0.9644    0.9669    0.9710
Prevalence              0.0680    0.0340    0.0380    0.0460    0.0440    0.0360
Detection Rate          0.0340    0.0100    0.0260    0.0120    0.0120    0.0080
Detection Prevalence    0.0520    0.0460    0.0400    0.0440    0.0340    0.0360
Balanced Accuracy       0.7403    0.6284    0.8348    0.6137    0.6249    0.5966
                     Class: 20 Class: 21 Class: 22
Sensitivity             0.2353    0.2000   0.08333
Specificity             0.9607    0.9521   0.96429
Pos Pred Value          0.1739    0.1481   0.10526
Neg Pred Value          0.9727    0.9662   0.95426
Prevalence              0.0340    0.0400   0.04800
Detection Rate          0.0080    0.0080   0.00400
Detection Prevalence    0.0460    0.0540   0.03800
Balanced Accuracy       0.5980    0.5760   0.52381
```

Hide

```
cat("The accuracy for gbm model is", mean(dat_test$emotion_idx==pred.class)*100,
"%.\n")
```

```
The accuracy for gbm model is 42.4 %.
```

Summarize Running Time

Hide

```
tm_test_gbm<-system.time(pred_gbm)
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
```

```
Time for constructing training features= 0.891 s
```

Hide

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

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

Hide

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

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

Hide

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

```
Time for testing model= 0 s
```