Main

Project3 Group11, Junyan Guo

Step 0 set work directories

```
set.seed(0)
p<-getwd()
setwd(p)
# here replace it with your own path or manually set it in RStudio to where t
his 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="")
```

Source important functions, including train, test, cross_validation and tune.The pca function is a plus, we put the pca file into the lib, however, it may decrease the predicting accuracy. Therefore, source it as long as you may be interested in it.

```
source("../lib/cross_validation.R")
source("../lib/train.R")
source("../lib/test.R")
source("../lib/tune.R")
source("../lib/feature.R")
source("../lib/pca.R")
source("../lib/kpca.R")
```

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 distance feature for training set (the default is pairwise feature)
- (T/F) conduct dimension reduction with pca method
- (T/F) conduct dimension reduction with kpca method
- (T/F) make prediction on an independent test set
- (T/F) do tuning for selected methods
- (T/F) do tuning on baseline model (the best parameters were plugged into train function as default)

- (T/F) decide whether to train baseline model and to predict
- (T/F) Change F to T if you want to train one of the models and predict

```
# overall control
run.cv = FALSE # run cross-validation on the training set
K = 5  # number of CV folds
run.feature = FALSE # process features (distance) for training set
run.pca = FALSE # dimension reduction with pca
run.kpca = FALSE # dimension reduction with kpca
run.test = TRUE # run evaluation on an independent test set
run.tune = FALSE #tune parameter

# baseline model control
run.baseline.tuning = FALSE
run.baseline = TRUE

# selected models (choose only one at a time)
run.gbm = FALSE   # baseline model
run.xgboost = FALSE
run.adaboost = FALSE
run.ksvm = FALSE
run.svm = TRUE
run.logistic = FALSE
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications.

```
# potential sets of parameters (for tuning)
shrinks_range <- c(0.01,0.05,0.1,0.15,0.2) #for gbm
trees_range  <- c(40,50,60,70,100) #for gbm
mfinal <- c(50, 75, 100, 125) # for adaboost
max_depth_values<-seq(3,9,2) #for xgboost
min_child_weight_values <- seq(1,6,2) #for xgboost
C <- c(1,5,10,20,50) # for ksvm
sigma <- c(0.0005,0.001,0.01,0.1,1) # for ksvm
cost <- c(0.00001,0.0001,0.001,0.01,0.1,1,5) # for svm
```

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

- We used pairwise distance between fiducial points as feature extraction method.

`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

```
tm_feature_train <- NA
tm_feature_test <- NA
if(run.feature == FALSE){
  tm_feature_train <- system.time(da_train <- feature(fiducial_pt_list, train
_idx))
  tm_feature_test <- system.time(da_test <- feature(fiducial_pt_list, test_id
x))
}else{
  source("../lib/feature_distance.R")
  tm_feature_train <- system.time(da_train <- feature_dist(fiducial_pt_list,
train_idx))
  tm_feature_test <- system.time(da_test <- feature_dist(fiducial_pt_list, te
st_idx))
}
```

```
save(da_train, file="../output/feature_train.RData")
save(da_test, file="../output/feature_test.RData")

dat_train <- da_train[,-ncol(da_train)]
dat_test <- da_test[,-ncol(da_test)]
label_train <- da_train[,ncol(da_train)]
label_test <- da_test[,ncol(da_test)]
```

Step 4: Train Baseline model

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

Tuning Parameters with cross-validation

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

```
if(run.baseline.tuning == T){
   par_best_baseline <- tune(dat_train ,label_train,
               run.gbm = T,
               run.xgboost = F,
               run.adaboost = F)
save(par_best_baseline,file = "par_best_baseline.RData")
print(par_best_baseline)
}

if(run.baseline == T){
  tm_train=NA
  tm_train <- system.time(fit_train_base <- train(dat_train, label_train, run.
gbm = T))
  tm_test = NA
  tm_test <- system.time(fit_test_base <- test(fit_train_base, dat_test, run.
gbm = T))
  error <- mean(fit_test_base != label_test)
  save(fit_train_base, file="../output/baseline_model.RData")
}
```

- evaluation

```
if(run.baseline == T){
  accu <- mean(label_test == fit_test_base)
  cat("The accuracy of model:", "is", accu*100, "%.\n")
}

## The accuracy of model: is 40.6 %.
```

Note that the accuracy is 43.4% after tuning wihle only 22% (before tuning) with default parameter settings in gbm function.

Step5: Potential Modified Models

tuning parameters for selected model

```
if(run.tune == T){
  par_best <- tune(dat_train,
                   label_train,
                   run.xgboost = F,
                   run.adaboost = F,
                   run.ksvm = F,
                   run.svm = T,
                   run.logistic = F,
                   verbose = FALSE)
  save(par_best,file = "par_best.RData")
  print(par_best)
}
```

train model on training set and make prediction

```
if(run.test == TRUE){
  tm_train=NA
  tm_train <- system.time(fit_train <- train(dat_train,
                                             label_train,
                                             run.gbm = F,
                                             run.xgboost = F,
                                             run.adaboost = F,
                                             run.ksvm = F,
                                             run.svm = T,
                                             run.logistic = F))
  tm_test = NA
  tm_test <- system.time(fit_test <- test(fit_train,
                                          dat_test,
                                          run.gbm = F,
                                          run.xgboost = F,
                                          run.adaboost = F,
                                          run.ksvm = F,
                                          run.svm = T,
                                          run.logistic = F))
  error <- mean(fit_test != label_test)
  save(fit_train, file="../output/model.RData")
  accu <- mean(fit_test == label_test)
```

```r
  cat("The accuracy of model:", "is", accu*100, "%.\n")
}
```

```
## The accuracy of model: is 52 %.
```

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")
```

```
## Time for constructing training features= 0.73 s
```

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

```
## Time for constructing testing features= 0.139 s
```

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

```
## Time for training model= 106.093 s
```

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

```
## Time for testing model= 10.358 s
```

Save all intersted infomation in to excel

```r
t1 <- tibble(True_label = label_test,
             Predict_label = fit_test)
t2 <- tibble(Type = c("Time for constructing training features",
                      "Time for constructing testing features",
                      "Time for training model",
                      "Time for testing model"),
             Time = c(tm_feature_train[1],
                      tm_feature_test[1],
                      tm_train[1],
                      tm_test[1]))
write_xlsx(list("Label"=t1,"Running_time"=t2),path = "../output/Result.xlsx")
```

###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.