Train

NAN YANG

10/31/2019

Install and load packages we need

```
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")
}

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

Step 0 set work directories, extract paths, summarize

```
set.seed(0)
setwd("~/Desktop/proj3-sec2-group6/doc")
# here replace it with your own path or manually set it in RStudio to where this r
md 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 <- "~/Desktop/proj3-sec2-group6/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.

```
k = c(5,11,21,31,41,51)
model_labels = paste("KNN with 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*(0/5), 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"
))
}
```

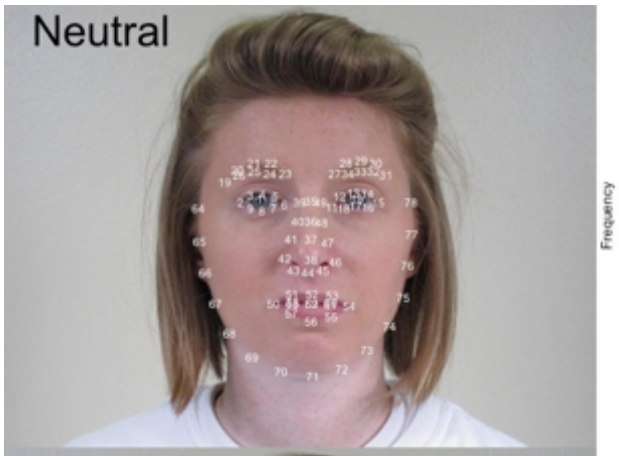
Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

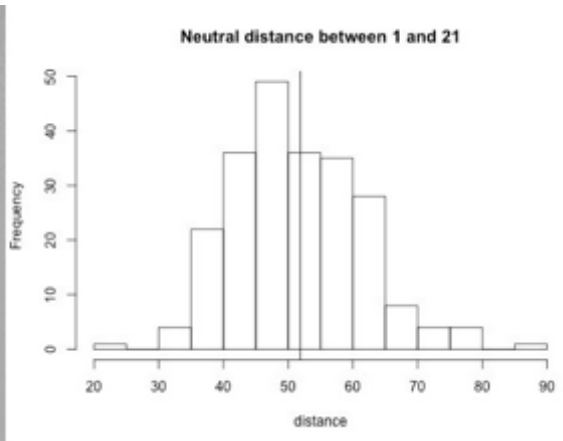
```
#function to read fiducial points
#input: index
#output: matrix of fiducial points corresponding to the index
readMat.matrix <- function(index){
     return(round(readMat(paste0(train_pt_dir, sprintf("%04d", index), ".mat"))[[1
]],0))
}

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

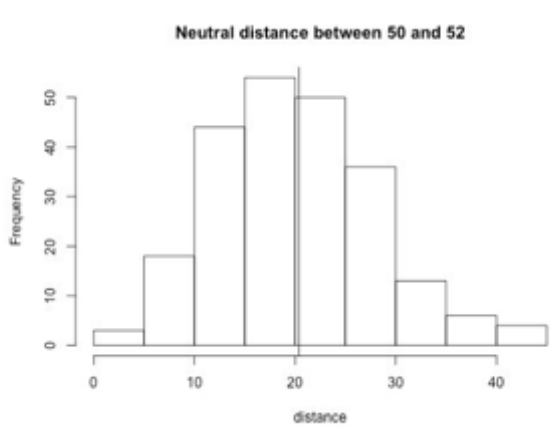
Step 3: construct features and responses

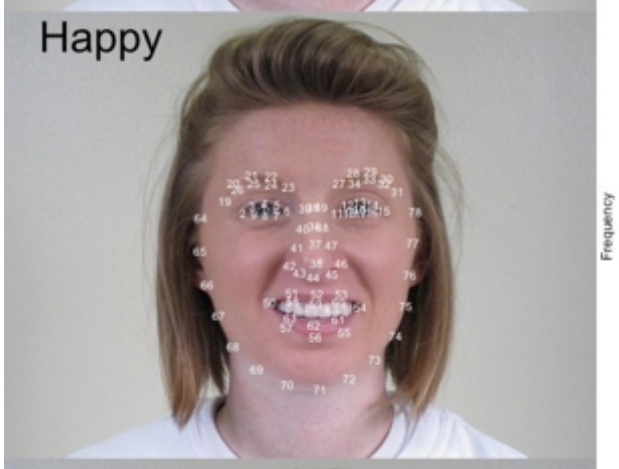
- The follow plots show how pairwise distance between fiducial points can work as feature for facial emotion recognition.

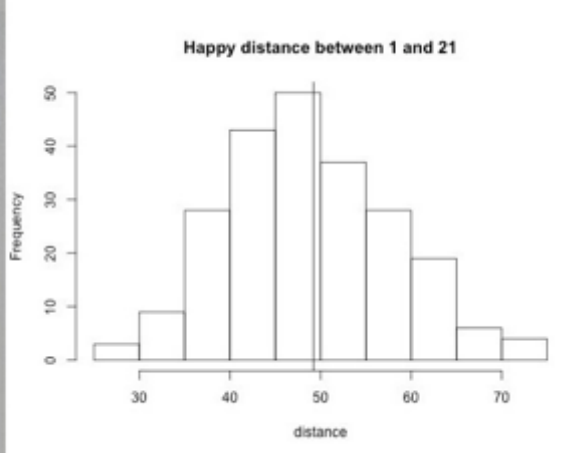
- In the first column, 78 fiducials points of each emotion are marked in order.
- In the second column distributions of vertical distance between right pupil(1) and right brow peak(21) are shown in histograms. For example, the distance of an angry face tends to be shorter than that of a surprised face.
- 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 face.

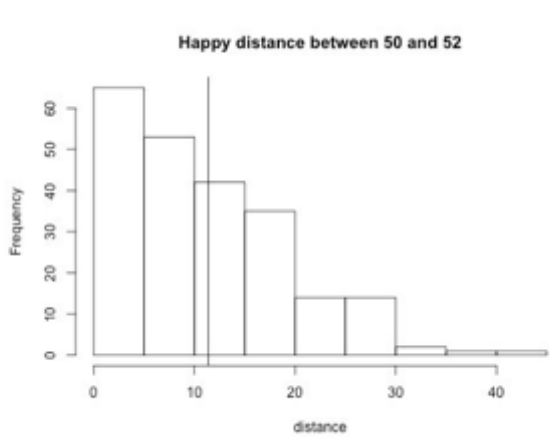


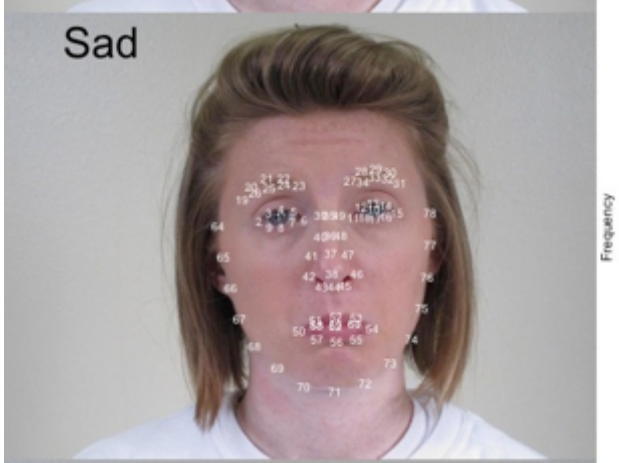


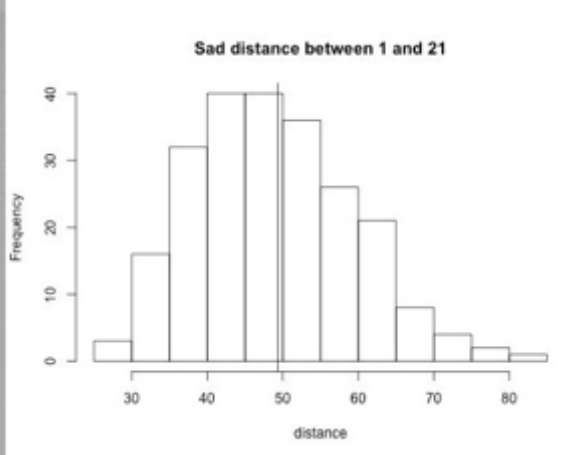


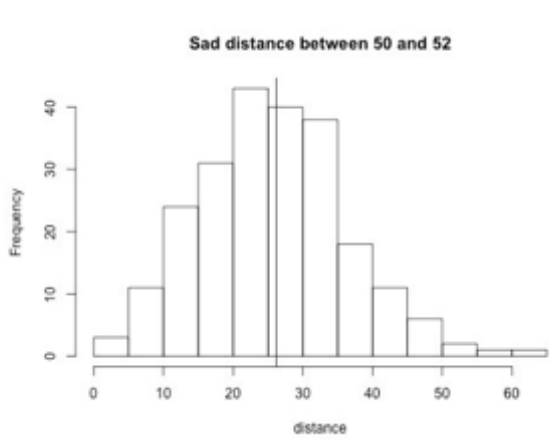












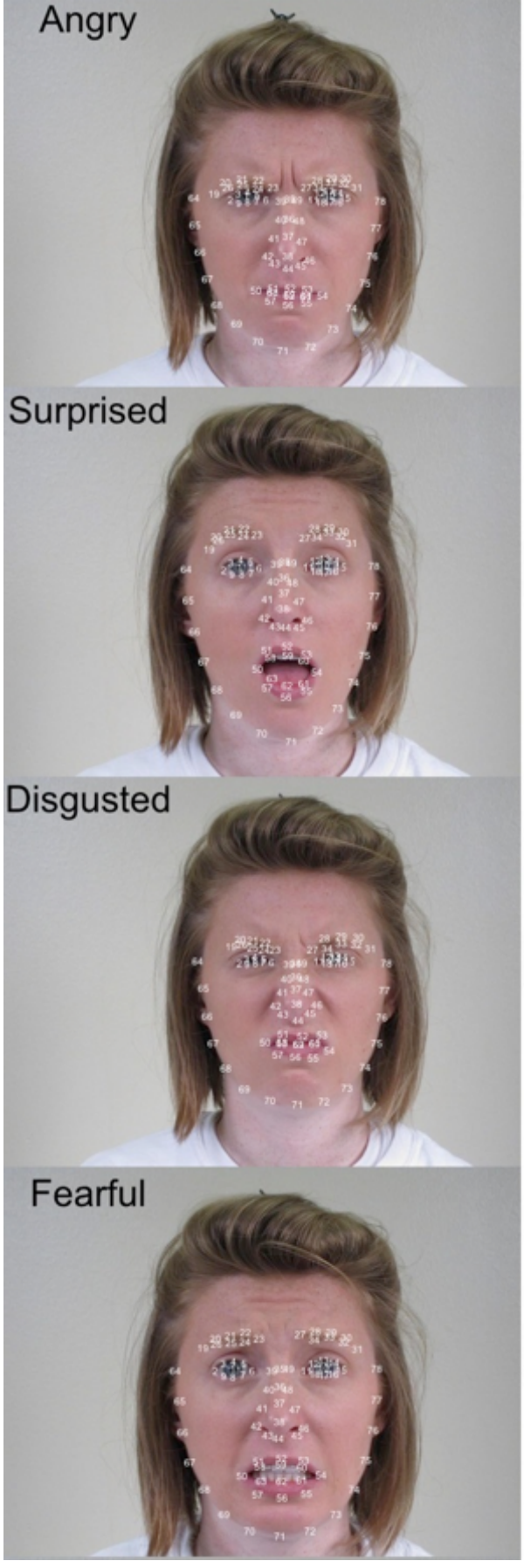


Figure1

`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

We need all the data to be test data.

```
source("../lib/feature.R")
#tm_feature_train <- NA
#if(run.feature.train){
  #tm_feature_train <- system.time(dat_train1 <- feature(fiducial_pt_list, train_i
dx))
#}

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

#save(dat_train1, file="../output/feature_train.RData")
save(dat_test, file="../output/feature_test.RData")
```

This is how we work out gbm model, it will take more than 3 hours. Thus I saved the model.

```
#fitControl <- trainControl(
  #method = "repeatedcv",
  #number = 5,
  #repeats = 1)

#gbmGrid <- expand.grid(interaction.depth = 2,
                        #n.trees = 300,
                        #shrinkage = 0.15,
                        #n.minobsinnode = 10)

#gbmFit_base <- train(emotion_idx~., data = dat_train2,
                  #method = "gbm",
                  #trControl = fitControl,
                  #verbose = TRUE,
                  #tuneGrid = gbmGrid)

#saveRDS(gbmFit_base,file = '~/Desktop/pca_ads_proj3/gbm_base1.rds')
```

Load the model.

```
gbm_base1 <- readRDS('~/Desktop/pca_ads_proj3/gbm_base1.rds')
```

Predict the data

```
pred1 <- predict(gbm_base1, newdata = dat_test)
accu <- mean(dat_test$emotion_idx == pred1)
cat("The accuracy of model: gredient boosting", "is", accu*100, "%.\n")
```

```
## The accuracy of model: gredient boosting is 100 %.
```

```
#confusionMatrix(pred1, dat_test$emotion_idx)
```

Since we used all 2500 data to be train_set to build model which were used to predict new 2500 test data. Thus the accuracy of the 2500 data will be high. However, the accuracy is 100%, showing that there is probably overfitting problem in this model.

main_advance

load package

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()

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")
}

library(BiocManager)
library(R.matlab)
library(readxl)
library(dplyr)
library(ggplot2)
library(caret)
library(OpenImageR)
```

set up the local path for data

```
local_dir <- "C:/Users/think/Desktop/ads/"
```

load training Rdata

```
load(paste(local_dir,'feature_train.RData',sep = ''))
```

normalize training set & PCA on training set

```
# normalize
df_train_X <- scale(dat_train[,!(names(dat_train) %in% 'emotion_idx')])
df_train_Y <- dat_train$emotion_idx

# PCA on training
source(paste(local_dir,'pca_feature.R',sep = ''))

pca <- pca_feature(df_train_X,threshhold=0.99)

# combine PCA X, Y after PCA
pca_train <- data.frame(pca$data_X_transformed,emotion_idx=df_train_Y)
```

build advance model on training

```
source(paste(local_dir,'train_advance.R',sep = ''))

run.advance.train=TRUE
tm_advance_train=NA

if(run.advance.train){
  tm_advance_train <- system.time(advance_model <- train_advance(pca_train))
}
```

load & scale on test & transform test into PC dimensions

```
load(paste(local_dir,'feature_test.RData',sep = ''))

dat_test <- dat_test2
```

```
## Error in eval(expr, envir, enclos): 找不到对象'dat_test2'
```

```
dat_test$emotion_idx <- c(1:2500)
```

```
## Error in `$<-.data.frame`(`*tmp*`, emotion_idx, value = 1:2500): replacement
has 2500 rows, data has 500
```

```
df_test_X <- scale(dat_test[,!(names(dat_test) %in% 'emotion_idx')])
df_test_Y <- dat_test$emotion_idx

data_test_X_transformed <-  df_test_X  %*% pca$trans_matrix

# combine test X, Y after PCA
pca_test <- data.frame(data_test_X_transformed,emotion_idx=df_test_Y)
```

predict on test - calculate accuracy on test

```
source(paste(local_dir,'test_advance.R',sep = ''))

run.advance.test=TRUE
tm_advance_test=NA

if(run.advance.test){
  tm_advance_test <- system.time(pred_test_advance <- test_advance(model=advanc
e_model,dat_pca_test=pca_test))
}


cat("The accuracy of advance model(lda2):",confusionMatrix(pred_test_advance,re
ference = pca_test$emotion_idx)$overall[1]*100, "%.\n")
```

```
## The accuracy of advance model(lda2): 54.4 %.
```

summarizing running time

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

```
## Time for training advance model= 22.95 s
```

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

```
## Time for testing advance model= 0.03 s
```

save predictions into csv

```
write.csv(pred_test_advance, file=paste(local_dir,"pred_test_advance.csv",sep
= ''))
```