Tree Classification

Yandong Xiong

3/16/2021

```r
if(!require("EBImage")){
  install.packages("BiocManager")
  BiocManager::install("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("caret")){
  install.packages("caret")
}

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

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

Step 0 set work directory

```r
set.seed(2020)
setwd("~/Documents/GitHub/Spring2021-Project3-group-1/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
```

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.

```r
run.feature.train <- TRUE # process features for training set
run.feature.test <- TRUE # process features for test set
run_tree <- TRUE # process tree classification for training set
```

Step 2: import data and train-test split

```r
#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) #get the index in info$Index but different from 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.

```r
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.

```r
#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.

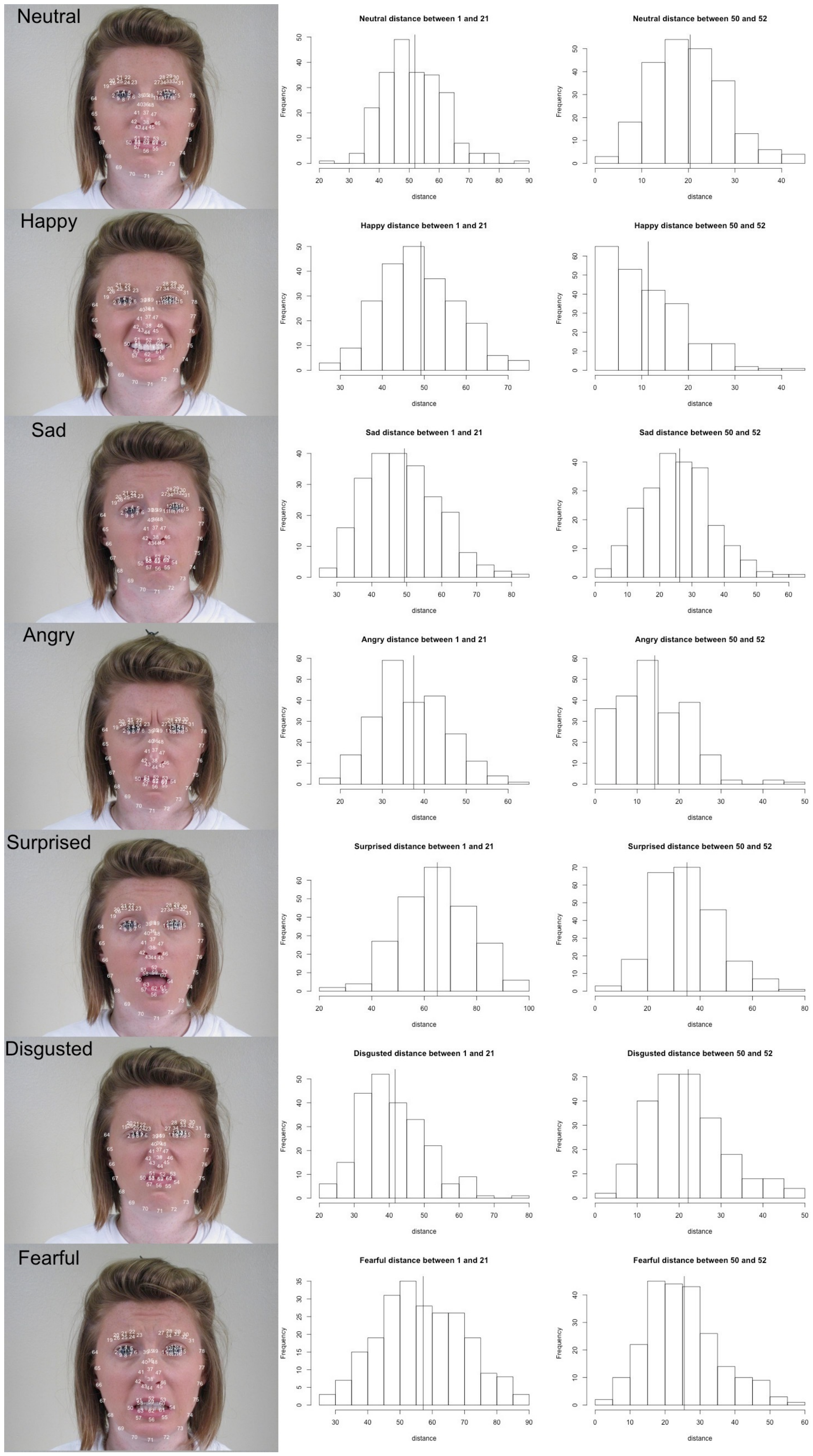

Figure 1: Figure1

`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

```r
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))
  save(dat_train, file="../output/feature_train.RData")
}else{
  load(file="../output/feature_train.RData")
}

tm_feature_test <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature(fiducial_pt_list, test_idx))
  save(dat_test, file="../output/feature_test.RData")
}else{
  load(file="../output/feature_test.RData")
}
```

step 4: load balanced train dataset

```r
source("../lib/Trees_Classification.R")
feature_train_smote = read.csv("../output/feature_train_smote.csv")
label_train_smote = read.csv("../output/label_train_smote.csv")
dat_train_smote = data.frame(cbind(feature_train_smote[,-1],label = label_train_smote[,-1]-1))
```

step 5: Train a decision tree model with imbalanced and balanced datasets

```r
tree_list = Tree(dat_train, dat_test, run_tree)
```

```
## Loading required package: rpart

## Loading required package: pROC

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```r
tree_list.smote = Tree(dat_train_smote, dat_test, run_tree)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

step 6: summry from the training and testing result

```r
cat("Training and testing on imbalanced dataset:\n", "  The training accuracy of the decision tree with
```

```
## Training and testing on imbalanced dataset:
##    The training accuracy of the decision tree with the minimal cost complexity: 0.0191 is 80.3333 %.
##    The testing accuracy of the decision tree with the minimal cost complexity: 0.0191 is 79 %.
##    The AUC of the model is 0.5 .
##    Time for training model= 44.437 s.
##    Time for testing model= 0.995 s.
```

```r
cat("Training and testing on balanced dataset:\n", "  The training accuracy of the decision tree with th
```

```
## Training and testing on balanced dataset:
##    The training accuracy of the decision tree with the minimal cost complexity: 0.01 is 81.3809 %.
##    The testing accuracy of the decision tree with the minimal cost complexity: 0.01 is 77.5 %.
##    The AUC of the model is 0.4384 .
##    Time for training model= 55.831 s.
##    Time for testing model= 0.993 s.
```