Main_SVM_V2

Chengliang Tang, Yujie Wang, Tian Zheng, SVM Implemenation: Nichole Yao, Alexandra DeKinder, Yuting Gong, Adam Kravitz

This file is currently a template for running evaluation experiments. You should update it according to your codes but following precisely the same structure.

```
if (!requireNamespace("BiocManager", quietly = TRUE)){
   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("ggplot2")){
  install.packages("ggplot2")
}

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

library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(e1071)
```

Step 0 set work directories, extract paths, summarize

```
set.seed(0)
setwd("~/GitHub/Fall2019-proj3-sec1--proj3-sec1-grp9/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.

```
train_dir <- "../data/train_set/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*(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 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.train){
  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
- In this Starter Code, we use KNN to do classification.

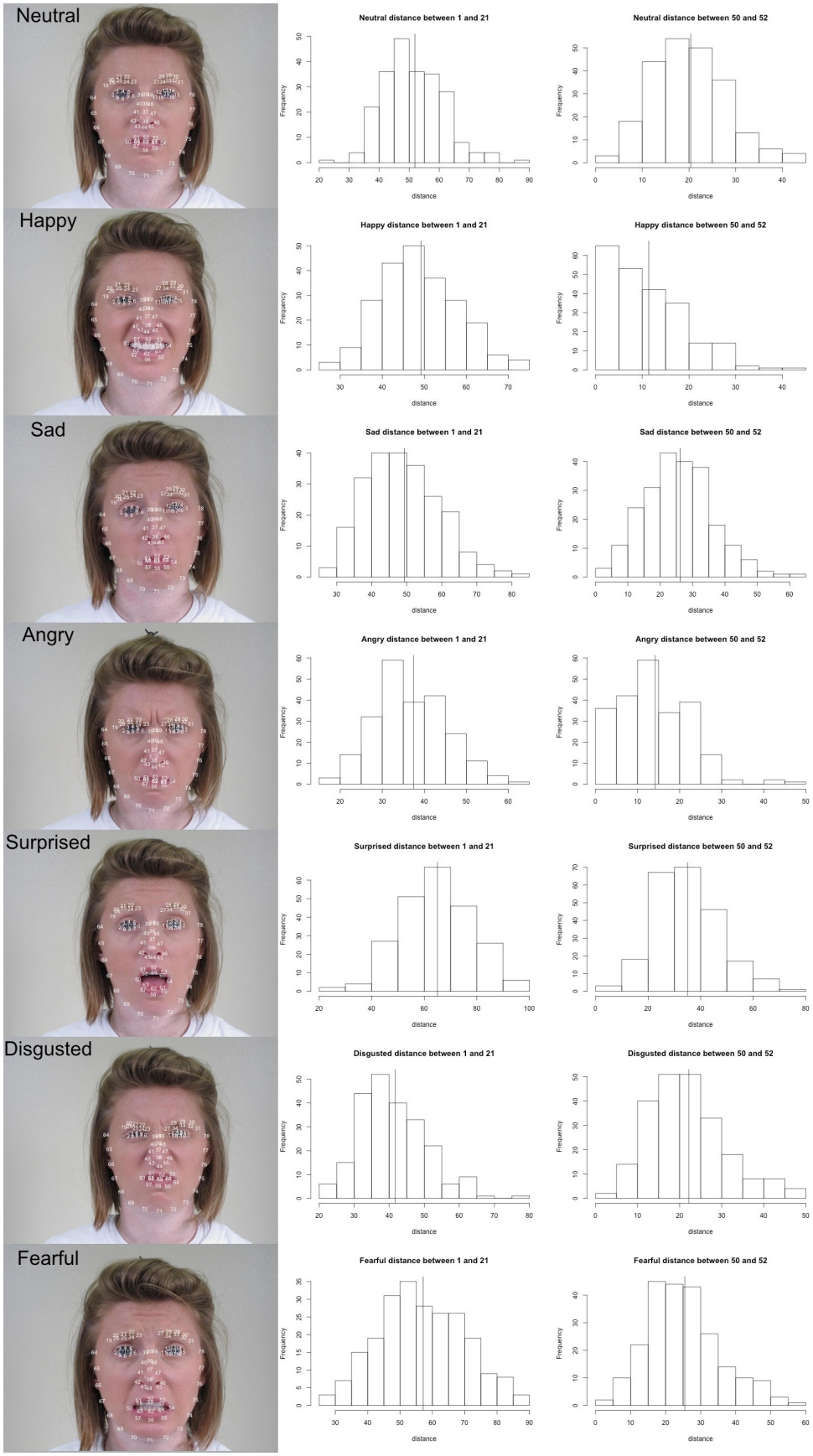


Figure 1: Figure1

```
#source("../lib/train.R") Since knn does not need to train, I comment this line.
source("../lib/test.R")
source("../lib/train.R")
```

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters.

For svm, adjust the model using gamma (scope of grouping) and cost (the complexity of the function)

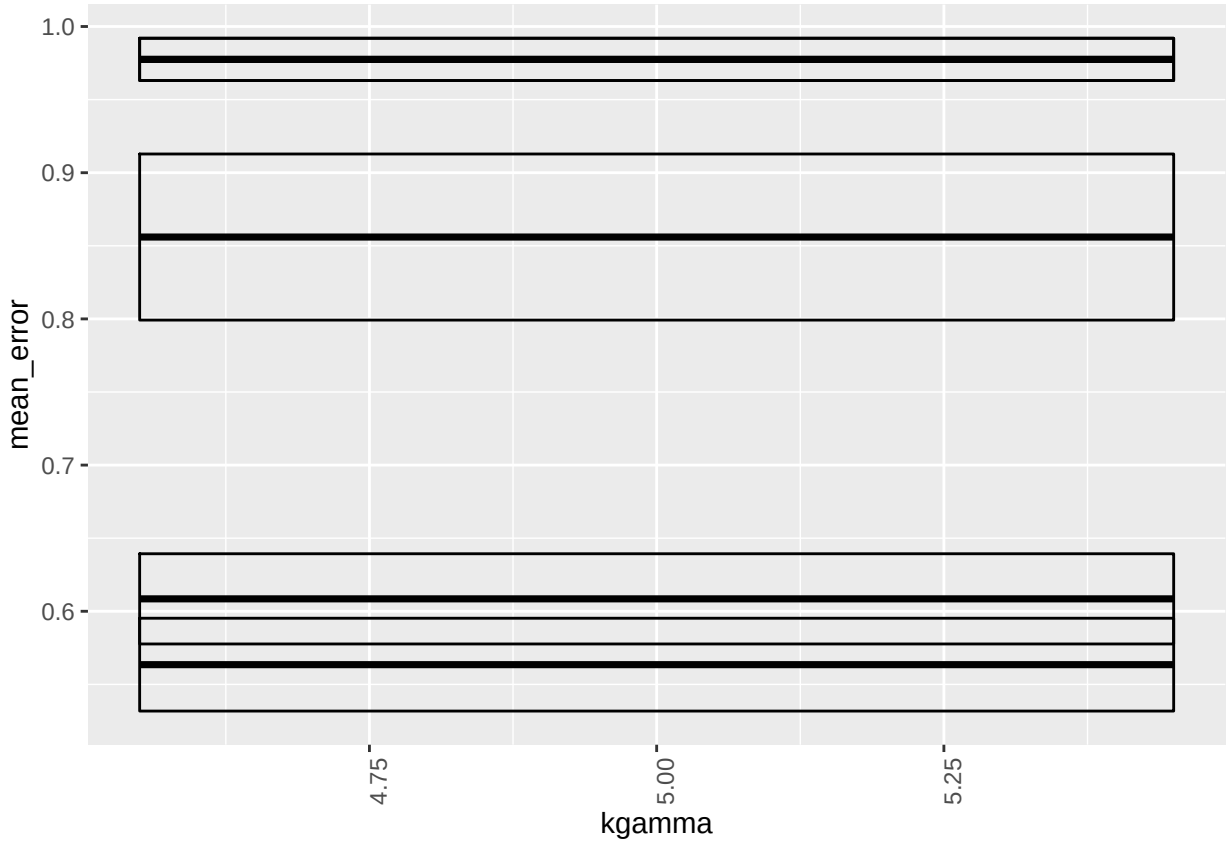
```
source("../lib/cross_validation.R")
k <- data.frame(gamma = 10^(-5:-1), cost = 10^(5:1))
if(run.cv){
  err_cv <- matrix(0, nrow = nrow(k), ncol = 2)
  for(j in 1:nrow(k)){
    err_cv[j,] <- cv.function(dat_train, K,k[j,1],k[j,2])
    save(err_cv, file="../output/err_cv.RData")
  }

}
```

Visualize cross-validation results.

```
if(run.cv){
  load("../output/err_cv.RData")
  err_cv <- as.data.frame(err_cv)
  colnames(err_cv) <- c("mean_error", "sd_error")

  kgamma <- k[[1]]
  err_cv %>%
    ggplot(aes(x = kgamma, y = mean_error,
                ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
}
```



- Choose the "best" parameter value

```
if(run.cv){

  k<-as.matrix(k)
  model_best <- k[which.min(err_cv[,1]),]
}
par_best = model_best
```

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

```
tm_train=NA
par_best <- as.data.frame(par_best)
tm_train <- system.time(fit_train <- train(dat_train, par_best$gamma, par_best$cost ))
save(fit_train, file="../output/fit_train.RData")
```

Step 5: Run test on test images

```
tm_test=NA
if(run.test){
  load(file="../output/fit_train.RData")
  tm_test <- system.time(pred <- test(fit_train, dat_test))
}
```

- evaluation

```r
accu <- mean(dat_test$emotion_idx == pred)
cat("The accuracy of model:", model_labels[which.min(err_cv[,1])], "is", accu*100, "%.\n")
```

```
## The accuracy of model: KNN with K = 5 is 4.2 %.
```

```r
library(caret)
confusionMatrix(pred, dat_test$emotion_idx)
```

```
## 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
##         1   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         2   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3  26 18 21 25 18 19 21 25 20 29 33 18 23 19 16 32 29 17 25 22 24
##         4   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         5   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         6   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         7   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         8   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         9   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         10  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         12  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         15  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         18  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         19  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         20  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         21  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         22  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##           Reference
## Prediction 22
##         1   0
##         2   0
##         3  20
##         4   0
##         5   0
##         6   0
##         7   0
##         8   0
##         9   0
##         10  0
##         11  0
##         12  0
##         13  0
##         14  0
##         15  0
##         16  0
##         17  0
##         18  0
```

```
##         19  0
##         20  0
##         21  0
##         22  0
##
## Overall Statistics
##
##                Accuracy : 0.042
##                  95% CI : (0.0262, 0.0635)
##     No Information Rate : 0.066
##     P-Value [Acc > NIR] : 0.9914
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity             0.000    0.000    1.000     0.00    0.000    0.000
## Specificity             1.000    1.000    0.000     1.00    1.000    1.000
## Pos Pred Value            NaN      NaN    0.042      NaN      NaN      NaN
## Neg Pred Value          0.948    0.964      NaN     0.95    0.964    0.962
## Prevalence              0.052    0.036    0.042     0.05    0.036    0.038
## Detection Rate          0.000    0.000    0.042     0.00    0.000    0.000
## Detection Prevalence    0.000    0.000    1.000     0.00    0.000    0.000
## Balanced Accuracy       0.500    0.500    0.500     0.50    0.500    0.500
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity             0.000     0.00     0.00     0.000     0.000
## Specificity             1.000     1.00     1.00     1.000     1.000
## Pos Pred Value            NaN      NaN      NaN       NaN       NaN
## Neg Pred Value          0.958     0.95     0.96     0.942     0.934
## Prevalence              0.042     0.05     0.04     0.058     0.066
## Detection Rate          0.000     0.00     0.00     0.000     0.000
## Detection Prevalence    0.000     0.00     0.00     0.000     0.000
## Balanced Accuracy       0.500     0.50     0.50     0.500     0.500
##                      Class: 12 Class: 13 Class: 14 Class: 15 Class: 16
## Sensitivity              0.000     0.000     0.000     0.000     0.000
## Specificity              1.000     1.000     1.000     1.000     1.000
## Pos Pred Value             NaN       NaN       NaN       NaN       NaN
## Neg Pred Value           0.964     0.954     0.962     0.968     0.936
## Prevalence               0.036     0.046     0.038     0.032     0.064
## Detection Rate           0.000     0.000     0.000     0.000     0.000
## Detection Prevalence     0.000     0.000     0.000     0.000     0.000
## Balanced Accuracy        0.500     0.500     0.500     0.500     0.500
##                      Class: 17 Class: 18 Class: 19 Class: 20 Class: 21
## Sensitivity              0.000     0.000      0.00     0.000     0.000
## Specificity              1.000     1.000      1.00     1.000     1.000
## Pos Pred Value             NaN       NaN       NaN       NaN       NaN
## Neg Pred Value           0.942     0.966      0.95     0.956     0.952
## Prevalence               0.058     0.034      0.05     0.044     0.048
## Detection Rate           0.000     0.000      0.00     0.000     0.000
## Detection Prevalence     0.000     0.000      0.00     0.000     0.000
## Balanced Accuracy        0.500     0.500      0.50     0.500     0.500
```

```
##                      Class: 22
## Sensitivity               0.00
## Specificity               1.00
## Pos Pred Value             NaN
## Neg Pred Value            0.96
## Prevalence                0.04
## Detection Rate            0.00
## Detection Prevalence      0.00
## Balanced Accuracy         0.50
```

Note that the accuracy is not high but is better than that of ramdom guess(4.5%).

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.

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

```
## Time for constructing training features= 0.69 s
```

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

```
## Time for constructing testing features= 0.17 s
```

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

```
## Time for testing model= 12.11 s
```

###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.