PCA-LDA

Zhihang Xia

```
if(!require("EBImage")){
  install.packages("BiocManager")
  BiocManager::install("EBImage")
}
if(!require("R.matlab")){
  install.packages("R.matlab")
}
```

```
## Warning: package 'R.matlab' was built under R version 4.0.4
```

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

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

```
## Warning: package 'dplyr' was built under R version 4.0.4
```

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

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

```
## Warning: package 'ggplot2' was built under R version 4.0.4
```

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

```
## Warning: package 'caret' was built under R version 4.0.4
```

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

```
## Warning: package 'glmnet' was built under R version 4.0.4
```

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

```
## Warning: package 'WeightedROC' was built under R version 4.0.4
```

```
library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(glmnet)
library(WeightedROC)
library(MASS)
library(vtreat)
```

```
## Warning: package 'vtreat' was built under R version 4.0.4
```

```
## Warning: package 'wrapr' was built under R version 4.0.4
```

Step 0 set work directories

```
set.seed(2020)
#setwd("../../Spring2021-Project3-group-1/doc")
```

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="")
```

Step 1: set up controls for evaluation experiments.

```
run.cv <- TRUE # run cross-validation on the training set
sample.reweight <- TRUE # run sample reweighting in model training
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
```

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)
```

```
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

`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))
  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: Train PCA-lDA model with training features and responses

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

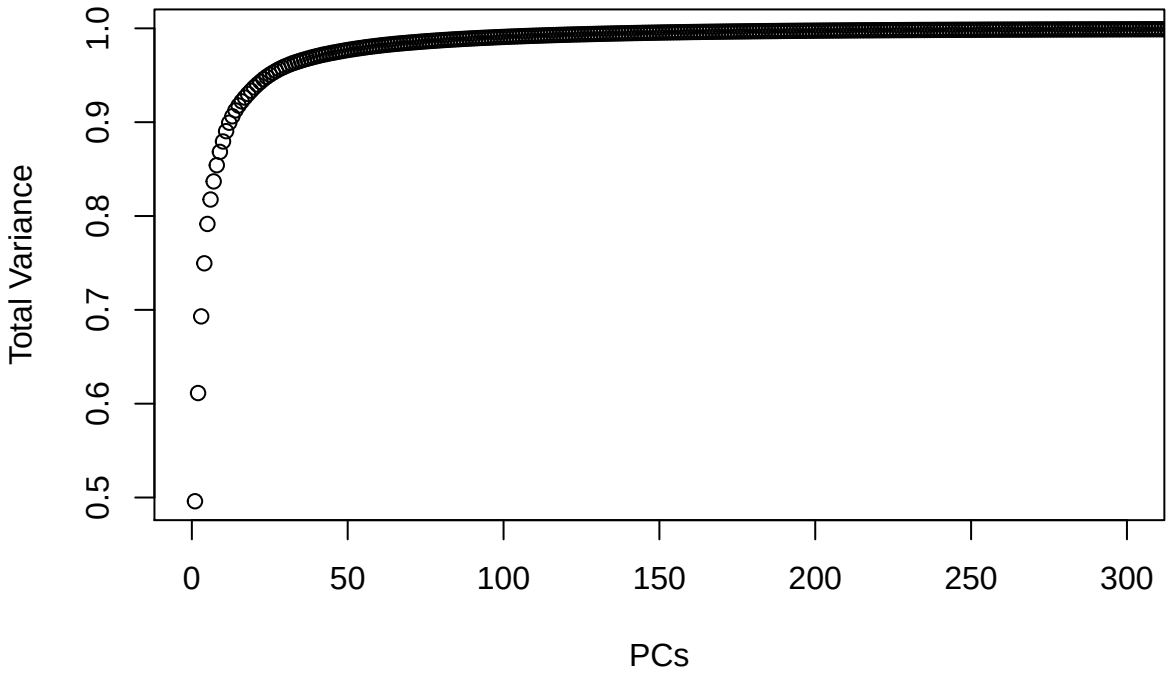
```
source("../lib/cross_validation_lda-pca.R")

run.pca_lda.cv = FALSE
if (run.pca_lda.cv){
  acc_means <- pca_lda.cv(dat_train, K=K)
  save(acc_means, file="../output/pca_lda_cv.RData")
}else{
  load("../output/pca_lda_cv.RData")
}
```

- Choose the "best" parameter value

```
c <- which.max(acc_means)
c
```

```
## [1] 168
```

```
# Train
t1 <- proc.time()
feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)
pca_train <- prcomp(feature_train)

CPVE <- (cumsum((pca_train$sdev)^2)/sum((pca_train$sdev)^2))
plot(1:length(CPVE), CPVE, xlab="PCs", ylab="Total Variance", xlim=c(0,300))
```




```
pca_train_c <- pca_train$x[,1:c]
dat_train_c <- data.frame(pca_train_c, label_train)
lda <- lda(label_train ~ ., data=dat_train_c)
label_predict_train <- predict(lda, data.frame(pca_train_c))

t2 <- proc.time()
train_time <- t2 - t1
accuracy_train <- mean(label_predict_train$class == label_train)

save(lda, file="../output/lda.RData")

#Test
t3 <- proc.time()
feature_test = as.matrix(dat_test[, -6007])
label_test = as.integer(dat_test$label)
pca_test <- predict(pca_train, feature_test)

label_predict_test <- predict(lda, data.frame(pca_test))
t4 <- proc.time()

test_time <- t4 - t3
accuracy_test <- mean(label_predict_test$class == label_test)



cat("Accuracy for training model=", accuracy_train, "\n")
```

```
## Accuracy for training model= 0.8525
```

```
cat("Accuracy for testing model=", accuracy_test, "\n")
```

```
## Accuracy for testing model= 0.8483333
```

```
cat("Time for training model=", train_time[3], "s \n")
```

```
## Time for training model= 125.82 s
```

```
cat("Time for testing model=", test_time[3], "s \n")
```

```
## Time for testing model= 6.19 s
```

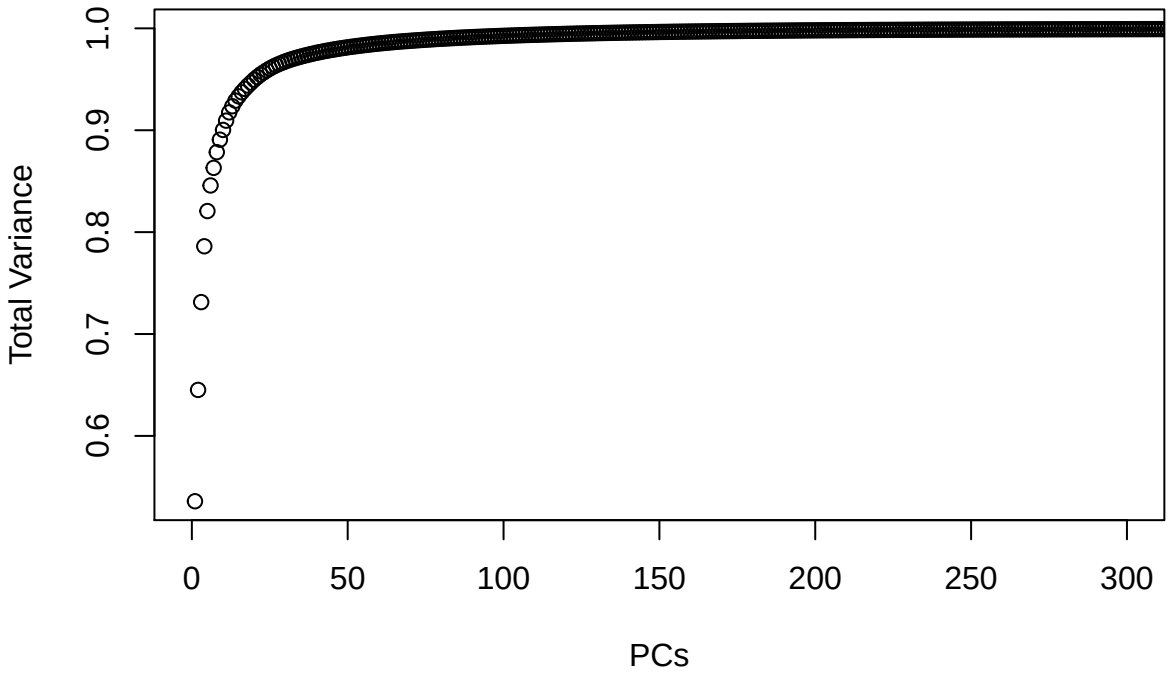
```
feature_train_new <- as.matrix(read.csv("../data/feature_train_smote.csv"))[, -1]
label_train_new <- read.csv("../data/label_train_smote.csv")[, -1]
dat_train_new <- data.frame(feature_train_new, label_train_new)
```

```
source("../lib/cross_validation_lda-pca.R")

run.pca_lda.cv_new <- FALSE
if (run.pca_lda.cv_new){
  acc_means_new <- pca_lda.cv(dat_train_new, K=K)
  save(acc_means_new, file="../output/pca_lda_cv_new.RData")
}else{
  load("../output/pca_lda_cv_new.RData")
}
```

```
c_new <- which.max(acc_means_new)
c_new
```

```
## [1] 246
```

```
# Train
t1_new <- proc.time()
pca_train_new <- prcomp(feature_train_new)

CPVE_new <- (cumsum((pca_train_new$sdev)^2)/sum((pca_train_new$sdev)^2))
plot(1:length(CPVE_new), CPVE_new, xlab="PCs", ylab="Total Variance", xlim=c(0,300))
```




```
pca_train_c_new <- pca_train_new$x[,1:c_new]
dat_train_c_new <- data.frame(pca_train_c_new, label_train_new)
lda_new <- lda(label_train_new ~ ., data=dat_train_c_new)
label_predict_train_new <- predict(lda_new, data.frame(pca_train_c_new))

t2_new <- proc.time()
train_time_new <- t2_new - t1_new
accuracy_train_new <- mean(label_predict_train_new$class == label_train_new)

save(lda_new, file="../output/lda_new.RData")

#Test
t3_new <- proc.time()
feature_test = as.matrix(dat_test[, -6007])
label_test = as.integer(dat_test$label)
pca_test_new <- predict(pca_train_new, feature_test)

label_predict_test_new <- predict(lda_new, data.frame(pca_test_new))
t4_new <- proc.time()

test_time_new <- t4_new - t3_new
accuracy_test_new <- mean(label_predict_test_new$class == label_test)

cat("Accuracy for training new model=", accuracy_train_new, "\n")
```

```
## Accuracy for training new model= 0.8642358
```

```r
cat("Accuracy for testing new model=", accuracy_test_new, "\n")
```

```
## Accuracy for testing new model= 0.805
```

```r
cat("Time for training new model=", train_time_new[3], "s \n")
```

```
## Time for training new model= 311.25 s
```

```r
cat("Time for testing new model=", test_time_new[3], "s \n")
```

```
## Time for testing new model= 9.58 s
```