Facial Emotion Recognition by PCA and LDA

Group 7

3/18/2020

Step 1: Some preparation

```
#getwd()
setwd("~/Desktop/Spring2020-Project3-ads-spring2020-project3-group7/doc/")
train_dir <- "../data/train_set/"
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="")

run.cv=FALSE # run cross-validation on the training set
K <- 5  # number of CV folds
run.feature.train=TRUE # process features for training set
```

Step 2: import data and train-test split

We splitted the data to 2000 (80%) for training and 500 (20%) for test.

```
#train-test split
set.seed(10)
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)
```

```
#function to read fiducial points
#input: index
#output: matrix of fiducial points corresponding to the index
n_files <- length(list.files(train_image_dir))
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")
```

```
load("../output/fiducial_pt_list.RData")
```

Step 3: baseline model

Step 3.1: feature construction

This step is converting 78 fiducial points to distances as 6006 features (3003 horizontal distances and 3003 vertical distances).

The time for training and test feature construction are as below (about 1.5s and 0.1s):

```
if(run.feature.train){

  source("../lib/feature.R")
  base.feature.construction.start = proc.time()
  tm_feature_train <- NA
  dat_train_base <- feature(fiducial_pt_list, train_idx)
  base.feature.construction.train.end = proc.time()

  tm_feature_test <- NA
  dat_test_base <- feature(fiducial_pt_list, test_idx)
  base.feature.construction.test.end = proc.time()

  #time for training feature construction
  print(base.feature.construction.train.end - base.feature.construction.start)
  #time for test feature construction
  print(base.feature.construction.test.end - base.feature.construction.train.end)

  save(dat_train_base, file="../output/feature_train_base.RData")
  save(dat_test_base, file="../output/feature_test_base.RData")

}
```

```
##    user  system elapsed
##   2.089   0.383   2.841
##    user  system elapsed
##   0.192   0.098   0.320
```

Step 3.2: load feature

```
load("../output/feature_train_base.Rdata")
load("../output/feature_test_base.Rdata")
```

Step 3.3: baseline model: gradient boosting machine

We use gradient boosting machine with stumps for our baseline model. The training dataset is 2000 × 6006 features and an emotion index list with length 2000 of 22 types as response. The time to train the baseline model is as below (about 307s):

```
#gbm classifier
base.train.model.start = proc.time()
baseline=gbm(emotion_idx~. ,data =dat_train_base ,distribution = "multinomial",n.trees = 100,
             shrinkage = 0.02,n.minobsinnode = 15,cv.folds = 5)
base.train.model.end = proc.time()
#time for training the baseline model
print(base.train.model.end - base.train.model.start)
```

```
##     user   system  elapsed
##  373.264    6.815 1101.670
```

This is our prediction part for gradient boosting model. The test dataset has the same variables as training data but with only 500 samples. It takes around 9.6s to predict and the test results are as below. The testing accuracy is 43%.

```
#predict on training data
baseline.pred.train = predict.gbm(object = baseline,
                   newdata = dat_train_base,
                   n.trees = 100,
                   type = "response")
#prediction result
baseline.labels.train = colnames(baseline.pred.train)[apply(baseline.pred.train, 1, which.max)]
baseline.cm.train = confusionMatrix(dat_train_base$emotion_idx, as.factor(as.numeric(baseline.labels.tra
print(baseline.cm.train$byClass[1])
```

```
## [1] 0.6979167
```

```
#predict on test data
base.test.start = proc.time()
baseline.pred = predict.gbm(object = baseline,
                   newdata = dat_test_base,
                   n.trees = 100,
                   type = "response")
base.test.end = proc.time()
#time for testing the baseline model
print(base.test.end - base.test.start)
```

```
##    user  system elapsed
##  13.608   0.430  14.244
```

```
#prediction result
baseline.labels = colnames(baseline.pred)[apply(baseline.pred, 1, which.max)]
baseline.cm = confusionMatrix(dat_test_base$emotion_idx, as.factor(as.numeric(baseline.labels)))
print(baseline.cm$byClass[1])
```

```
## [1] 0.5454545
```

```
print(baseline.cm$table)
```

```
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
##          1 12  0  5  1  0  0  0  0  0  3  0  0  0  0  0  1  0  0  0  0  0  0
##          2  0 17  0  0  0  0  0  1  5  0  0  0  0  0  0  0  0  0  0  1  0  0
##          3  0  0 10  1  0  0  0  0  0  1  0  0  1  0  0  0  1  0  3  0  0  0
##          4  3  0  2  9  0  2  1  0  0  1  0  1  3  0  0  0  0  0  0  0  0  1
##          5  0  0  0  0 10  0  3  1  0  0  0  0  0  7  2  0  1  0  1  1  0  0
##          6  1  0  4  0  0  5  0  0  2  0  0  1  2  0  0  0  0  0  0  0  1  0
##          7  0  0  0  0  1  2  7  0  0  0  0  0  0  0  1  1  6  1  4  2  2  0
##          8  0  4  0  0  0  0  0 15  6  0  0  0  0  0  0  0  2  0  0  1  0  0
##          9  0 10  0  0  0  1  0  3  4  1  0  1  0  0  1  0  0  0  0  4  1  1
##         10  1  0  6  2  0  1  0  0  1  9  0  0  2  0  0  0  0  0  0  0  0  0
##         11  1  0  6  1  0  0  0  0  2  3 10  3  1  1  0  1  0  0  0  1  0  0
##         12  2  0  2  2  0  1  0  0  0  2  0  2  8  0  0  2  0  0  0  0  0  1
##         13  2  0  1  2  0  0  0  0  0  0  2  1  4  0  1  2  0  0  0  0  1  0
##          14  0  0  0  0  5  0  0  0  0  0  0  0  0 11  4  0  0  0  0  2  3  0
##          15  0  0  0  0  2  0  0  0  0  0  0  0  0  3  5  0  0  0  3  1  0  1
##          16  0  0  8  2  0  0  0  1  1  0  0  0  2  0  0  4  6  1  0  0  1  0
##          17  0  0  0  0  1  0  0  3  0  0  0  0  0  0  1  0 10  3  1  2  0  0
##          18  0  0  0  0  1  0  1  0  0  0  0  0  0  1  1  0 10  6  3  1  0  1
##          19  0  0  0  0  0  0  1  2  0  1  0  0  0  1  2  0  3  0  6  0  2  1
##          20  0  1  0  0  0  0  0  0  1  0  0  0  0  2  0  0  4  1  2  4  5  0
##          21  0  0  0  0  2  1  1  0  0  0  0  0  1  6  1  0  2  0  1  3  6  0
##          22  0  1  4  1  0  1  1  0  1  1  0  1  2  2  0  0  0  0  5  3  2  0
```

Step 4: our improved model

Step 4.1: construct features and responses

Since people's faces areapproximately symmetric, we can choose only one side of each face to analyze.

- delete the duplicated right face;
- choose the points on the left side of each face and in the middle of each face;
- and obtain the features dataset of those chosen points.
- in the data dataset, calculate the mean of each feature in each emotion group, respectively, in this step, we get a 22 columns data frame, (i,j) means the mean value of feature i in emotion class j;
- then, if one feature changes very little in different emotions, we can say that this feature is not useful in distinguishing emotions. So, we can delete features that have small variation between emotions;
- choose certain variation level to delete features: <20% quantile;
- we delete 379 features.

```r
feature.construction.start = proc.time()
#delete the duplicated right face
#choose the points on one side of each face and in the middle of each face
leftmid_idx <- c(1:9,19:26,35:44,50:52,56:59,62,63,64:71)
fiducial_pt_list_lm <- lapply(fiducial_pt_list, function(mat){return(mat[leftmid_idx,])})



#let's see the features of these chosen points on every face provided.
data<-feature(fiducial_pt_list_lm,c(train_idx,test_idx))

#emotion is a vector containing all the unique emotions on the faces provided
emotion<-unique(data$emotion_idx)



#1st: in the data dataset, calculate the mean of each feature in each emotion group, respectively;
#2nd: heihei is a data frame containing the mean of each feature in each emotion.
heihei<-c()
library(dplyr)
for (i in 1:length(emotion)) {
  datadata<-data %>%
    filter(emotion_idx==emotion[i]) %>%
    dplyr::select(-emotion_idx) %>%
    colMeans()
  h<-t(as.matrix(datadata))
  h<-cbind(h,emotion_idx=emotion[i])
  heihei<-rbind(heihei,h)
  heihei<-as.data.frame(heihei)

}

#-------------------------------------------------------------------
#Then, according to heihei data frame, if one feature changes very little in different emotions,
#we can say that this feature is not useful in distinguishing emotions. So, we can delete features
#that have small variation between emotions.

feature.var<-map_dbl(heihei[,-ncol(heihei)],function(x) var(x))


#the following, I choose certain variation level to delete features: <20% quantile

q<-quantile(feature.var,0.2)
new_index_20<-which(feature.var<q) #delete these features

#load("../output/new_index(1).RData")
dup_horiz <- new_index_20
dup_horiz=as.numeric(dup_horiz)
#dup_horiz
```

```
if(run.feature.train){

  source("../lib/feature.R")
  tm_feature_train <- NA
  dat_train <- feature(fiducial_pt_list_lm, train_idx)
  feature.construction.train.end = proc.time()

  dat_train <- dat_train[,-dup_horiz]
  feature.construction.train.end = proc.time()

  tm_feature_test <- NA
  dat_test <- feature(fiducial_pt_list_lm, test_idx)
  feature.construction.test.end = proc.time()

  dat_test <- dat_test[,-dup_horiz]
  feature.construction.test.end = proc.time()

  #time for training feature construction
  print(feature.construction.train.end - feature.construction.start)
  #time for test feature construction
  print(feature.construction.test.end - feature.construction.train.end)

  save(dat_train, file="../output/feature_train.RData")
  save(dat_test, file="../output/feature_test.RData")

}
```

```
##    user  system elapsed
##   1.877   0.178   2.133
##    user  system elapsed
##   0.062   0.010   0.072
```

load features

```r
load("../output/feature_train.RData")
load("../output/feature_test.RData")
```

Step 4.2: PCA dimension reduction

Step 4.2.1: PCA+LDA

pca(linear method)

In this part, we choose principle component analysis(PCA) method to reduce dimension, thus to find the basic elements of the face image distribution, that is, the feature vector of the covariance matrix of the face image sample set, so as to approximate the face image.

```r
n.pca.list <- c(30,50,75,120,150,200)
dim(dat_train)
```

```
## [1] 2000 1514
```

```r
dim(dat_test)
```

```
## [1]  500 1514
```

```r
train.model.start = proc.time()

pca <- prcomp(dat_train[,-1514], cor=T)
#pca
train_pca <- data.frame(pca$x[,1:50])
#train_pca


pca2=predict(pca,dat_test[,-1514])
#pca2
test_pca=data.frame(pca2[, 1:50])
#test_pca


train_index<- dat_train[1514]
dat_train_pca <- cbind(train_pca, train_index)
#dat_train_pca

test_index<- dat_test[1514]
dat_test_pca <- cbind(test_pca, test_index)
#dat_test_pca

##training time

lda.model_pca <- lda(emotion_idx ~ ., data=dat_train_pca)
train.model.end = proc.time()
#time for training the model
print(train.model.end - train.model.start)
```

```
##    user  system elapsed
##  18.192   0.127  18.557
```

pca(linear method)

```r
n.pca.list <- c(30,50,80,100,150,200)
dim(dat_train)
```

```
## [1] 2000 1514
```

```r
dim(dat_test)
```

```
## [1]  500 1514
```

```r
train_time_pca=function(n.pca.list=n.pca.list){
for(i in 1:length(n.pca.list)){
  train.model.start = proc.time()
pca <- prcomp(dat_train[,-1514], cor=T)
train_pca <- data.frame(pca$x[,1:n.pca.list[i]])

pca2=predict(pca,dat_test[,-1514])
test_pca=data.frame(pca2[, 1:n.pca.list[i]])

train_index<- dat_train[1514]
dat_train_pca <- cbind(train_pca, train_index)

test_index<- dat_test[1514]
dat_test_pca <- cbind(test_pca, test_index)

##training time

lda.model_pca <- lda(emotion_idx ~ ., data=dat_train_pca)
#time for training the model
train.model.end = proc.time()

#time for testing the model
test.model.start = proc.time()
lda.test.pred_pca = predict(lda.model_pca, dat_test_pca[-dim(dat_test_pca)[2]])
test.model.end = proc.time()

#test accuracy
test_accuracy=confusionMatrix(lda.test.pred_pca$class, dat_test_pca$emotion_idx)$overall[1]

print(list(l1=train.model.end - train.model.start,
           l2=test.model.end - test.model.start,
           l3=test_accuracy))}
}
train_time_pca(n.pca.list)
```

```
## $l1
##    user  system elapsed
##  18.049   0.188  18.364
##
## $l2
##    user  system elapsed
##   0.006   0.001   0.008
##
## $l3
## Accuracy
##    0.494
##
## $l1
##    user  system elapsed
##  18.074   0.130  18.299
##
## $l2
##    user  system elapsed
##   0.004   0.001   0.004
##
## $l3
## Accuracy
##    0.508
##
## $l1
##    user  system elapsed
##  18.188   0.132  18.428
##
## $l2
##    user  system elapsed
##   0.005   0.001   0.006
##
## $l3
## Accuracy
##    0.492
##
## $l1
##    user  system elapsed
##  18.165   0.120  18.438
##
## $l2
##    user  system elapsed
##   0.006   0.000   0.007
##
## $l3
## Accuracy
##     0.48
##
## $l1
##    user  system elapsed
##  18.295   0.142  18.654
##
## $l2
##    user  system elapsed
##   0.008   0.000   0.009
##
## $l3
## Accuracy
##     0.46
##
## $l1
##    user  system elapsed
##  18.446   0.127  18.738
##
```

```
## $12
##    user  system elapsed
##   0.010   0.001   0.011
##
## $13
## Accuracy
##     0.45
```

Considering all the results including training time, testing time and accuracy, we choose 50 principle components.

```
train_pca_final <- data.frame(pca$x[,1:50])
dat_train_pca_final=cbind(train_pca_final, train_index)
pca2=predict(pca,dat_test[,-1514])
test_pca_final=data.frame(pca2[, 1:50])


dat_test_pca_final=cbind(test_pca_final, test_index)
dim(dat_test_pca_final)
```

```
## [1] 500  51
```

```
save(dat_train_pca_final, file="../output/feature_pca_train.RData")
save(dat_test_pca_final, file="../output/feature_pca_test.RData")
```

```
run.cv=TRUE # run cross-validation on the training set
K <- 5  # number of CV folds
run.feature.train=FALSE # process features for training set
run.test=TRUE # run evaluation on an independent test set
run.feature.test=TRUE # process features for test set
run.feature.test.test=FALSE # process features for test_test set
```

The best parameter with lda model

The original sample used a 1000×750 size picture to form a 750,000-dimensional feature vector, which contained a lot of redundant information and noise, which led to the inaccuracy of the LDA method. Therefore, PCA dimension reduction is generally used first: PCA dimension reduction is performed on the original sample image, and then LDA is used for classification training; when testing, PCA dimension reduction is also performed on the original image, and then LDA is used for recognition, which can effectively eliminate the interference of redundant information and noise, and the compressed information becomes insensitive to the position of the face.

```
################################################################################################
######################################---------LDA-------#######################################
################################################################################################
source("../lib/train_lda.R")
tm_train=NA
tm_train <- system.time(fit_train_baseline <- train(dat_train_pca_final, par = NULL))
save(fit_train_baseline, file="../output/fit_train_baseline_final.RData")

### Train Error
source("../lib/test_lda.R")
load("../output/fit_train_baseline_final.RData")

tm_test=NA
if(run.test){
```

```r
  tm_test <- system.time(pred_train <- test(fit_train_baseline, dat_train_pca_final))
}
accu <- mean(dat_train_pca_final$emotion_idx == pred_train$class)
accu
```

```
## [1] 0.578
```

```r
source("../lib/test_lda.R")
tm_test=NA
if(run.test){
  tm_test <- system.time(pred <- test(fit_train_baseline, dat_test_pca_final))
}

source("../lib/test_lda.R")
tm_test_test=NA
if(run.feature.test.test){
  load(file="../output/fit_train_baseline_final.RData")
  tm_test <- system.time(pred <- test(fit_train_baseline, dat_test_selected))
}

### Evaluation
accu <- mean(dat_test_pca_final$emotion_idx == pred$class)
cat("The accuracy of model:", "is", accu*100, "%.\n")
```

```
## The accuracy of model: is 50.8 %.
```

```r
library(caret)
#confusionMatrix(as.factor(labels), dat_test_pca_final$emotion_idx)
 ldatrain.model.start = proc.time()
lda.model <- lda(emotion_idx ~ ., data=dat_train_pca)
ldatrain.model.end = proc.time()
# #time for training the model
print(ldatrain.model.end - ldatrain.model.start)
```

```
##    user  system elapsed
##   0.041   0.004   0.044
```

```r
### 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= NA s
```

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

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

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

```
## Time for training model= 0.041 s
```

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

```
## Time for testing model= 0.005 s
```

Step 4.2.2: PCA+SVM

SVM takes into account both empirical risk and structural risk minimization, so it is stable. From a geometric point of view, the stability of the SVM is reflected in the requirement of the largest margin when constructing a hyperplane decision boundary, so there is ample space between the boundary boundaries to accommodate the test samples.

And we already know that SVM has been applied in pattern recognition problems in related fields, including portrait recognition, text classification, handwritten character recognition, etc.

Tune the SVM model with cross-validation:

```
###############################################################################################
#####################################---------SVM-------#####################################
###############################################################################################
tm_train=NA
tm_train <- system.time(tuned_parameters <- tune.svm(emotion_idx~.,
                                                     data = dat_train_pca_final,
                                                     gamma = 10^(-5:-1),
                                                     cost = c(30,35,40),
                                                     tunecontrol = tune.control(cross =12)
                                                     ))
summary(tuned_parameters)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 12-fold cross validation
##
## - best parameters:
##  gamma cost
##  1e-04   40
##
## - best performance: 0.5614915
##
## - Detailed performance results:
##    gamma cost     error dispersion
## 1  1e-05   30 0.9410095 0.01462613
## 2  1e-04   30 0.5725026 0.03886665
## 3  1e-03   30 0.5615065 0.03794215
## 4  1e-02   30 0.5640256 0.04088507
## 5  1e-01   30 0.8460278 0.03420935
## 6  1e-05   35 0.9380095 0.01794934
## 7  1e-04   35 0.5679905 0.04158850
## 8  1e-03   35 0.5695086 0.03177526
## 9  1e-02   35 0.5640256 0.04088507
## 10 1e-01   35 0.8460278 0.03420935
## 11 1e-05   40 0.9335125 0.01701958
## 12 1e-04   40 0.5614915 0.03818713
## 13 1e-03   40 0.5669955 0.03113381
## 14 1e-02   40 0.5640256 0.04088507
## 15 1e-01   40 0.8460278 0.03420935
```

Use cross validation to find the best number of PC under SVM model

```
###############################################################################################
#####################################---------svm-------########################################
###############################################################################################

train_time_svm=function(n.pca.list=n.pca.list){

for(i in 1:length(n.pca.list)){
  train.model.start = proc.time()
pca <- prcomp(dat_train[,-1514], cor=T)
train_pca <- data.frame(pca$x[,1:n.pca.list[i]])

pca2=predict(pca,dat_test[,-1514])
test_pca=data.frame(pca2[, 1:n.pca.list[i]])

train_index<- dat_train[1514]
dat_train_pca <- cbind(train_pca, train_index)

test_index<- dat_test[1514]
dat_test_pca <- cbind(test_pca, test_index)

##training time
svmtrain.model.start = proc.time()
svm.model <- svm(emotion_idx~., data = dat_train_pca, kernal = "radial", cost = 1)
svmtrain.model.end = proc.time()
#time for training the model

#time for testing the model
svmtest.model.start = proc.time()
test.svm.pred <- as.numeric(predict(svm.model, dat_test_pca[-dim(dat_test_pca)[2]]))
svmtest.model.end = proc.time()


#test accuracy
test_accuracy=mean(test.svm.pred == dat_test_pca$emotion_idx)

print(list(l1=svmtrain.model.end-svmtrain.model.start,
           l2=svmtest.model.end-svmtest.model.start,
           l3=test_accuracy))
}
}

train_time_svm(n.pca.list)
```

```
## $l1
##    user  system elapsed
##   0.971   0.012   0.994
##
## $l2
##    user  system elapsed
##   0.076   0.001   0.078
##
## $l3
## [1] 0.426
##
```

```
## $l1
##    user  system elapsed
##   1.458   0.016   1.485
##
## $l2
##    user  system elapsed
##   0.110   0.002   0.113
##
## $l3
## [1] 0.448
##
## $l1
##    user  system elapsed
##   2.227   0.016   2.263
##
## $l2
##    user  system elapsed
##   0.165   0.002   0.169
##
## $l3
## [1] 0.448
##
## $l1
##    user  system elapsed
##   3.028   0.074   3.154
##
## $l2
##    user  system elapsed
##   0.195   0.002   0.198
##
## $l3
## [1] 0.444
##
## $l1
##    user  system elapsed
##   3.709   0.026   3.753
##
## $l2
##    user  system elapsed
##   0.284   0.049   0.335
##
## $l3
## [1] 0.434
##
## $l1
##    user  system elapsed
##   4.973   0.070   5.119
##
## $l2
##    user  system elapsed
##   0.358   0.002   0.361
##
## $l3
## [1] 0.402
```

svm training time & svm testing time

```
source("../lib/train_svm.R")
par_best=NULL
fit_train_final_svm <- train(dat_train_pca_final, tuned_parameters$best.parameters)
save(fit_train_final_svm, file="../output/fit_train_final.RData")


### Train accurancy:
source("../lib/test_svm.R")
load("../output/fit_train_final.RData")

if(run.test){
  pred_train <- test(fit_train_final_svm, dat_train_pca_final)
}
accu.train <- mean(dat_train_pca_final$emotion_idx == pred_train)
accu.train
```

```
## [1] 0.59
```

```
# [1] 0.59

### SVM: Run test on test images
source("../lib/test_svm.R")
tm_test=NA
if(run.test){
  tm_test <- system.time(pred <- test(fit_train_final_svm, dat_test_pca_final))
}

### SVM: Run test_test on test images
source("../lib/test_svm.R")
tm_test=NA
if(run.test){
  load(file="../output/fit_train_final.RData")
  tm_test <- system.time(pred <- test(fit_train_final_svm, dat_test_pca_final))
}

### evaluation
accu <- mean(dat_test_pca_final$emotion_idx == pred)
cat("The accuracy of model:", "is", accu*100, "%.\n")
```

```
## The accuracy of model: is 47.2 %.
```

```
library(caret)
confusionMatrix(pred, dat_test_pca_final$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 22
##          1 18  0  1  1  0  2  0  0  1  1  1  0  0  0  1  0  1  2  0  0  1  0
##          2  0 20  0  0  0  0  0  6  7  0  0  0  0  0  0  0  0  0  0  0  0  0
##          3  0  0 11  0  0  1  1  0  0  2  4  2  2  0  0  9  1  0  1  1  2  6
##          4  1  0  2 11  0  1  0  0  1  2  2  1  0  0  0  0  0  0  1  0  1  1
##          5  0  0  0  0 17  0  0  1  0  0  0  0  0  0  1  0  1  3  0  1  0  0
##          6  0  0  0  0  0  4  0  0  1  0  0  3  0  0  0  1  0  0  0  0  0  0
##          7  2  1  0  2  1  0 13  2  1  0  0  0  0  1  0  0  0  2  3  2  1  0
##          8  0  2  0  0  1  0  0 16  2  0  0  0  0  0  0  0  2  1  0  0  0  0
##          9  0  1  0  0  0  0  0  1 10  0  0  0  0  0  0  0  0  0  0  0  0  0
##         10  0  0  1  1  0  1  0  0  1 15  4  0  4  0  0  0  0  0  0  0  0  3
##         11  0  0  0  1  0  2  0  0  0  1 10  3  0  0  0  0  0  0  0  1  0  1
##         12  0  0  0  0  0  3  0  0  0  0  4  8  0  1  0  0  0  0  0  0  0  1
##         13  0  0  0  5  0  0  0  0  1  0  3  3  9  1  0  2  0  0  1  1  1  1
##         14  0  0  0  0  3  1  0  0  0  0  0  1  0 16  4  0  1  2  1  1  2  0
##         15  0  0  0  0  1  0  0  0  0  0  0  0  0  4  8  0  0  0  0  0  1  0
##         16  0  0  0  1  0  0  3  0  0  0  0  0  1  0  0 14  0  0  0  1  0  1
##         17  0  0  0  0  0  0  1  2  0  0  0  0  0  0  0  0  6  9  2  2  1  0
##         18  0  0  0  0  3  0  1  0  0  0  0  0  0  0  0  0  5  6  0  0  0  0
##         19  0  0  2  0  0  1  3  0  0  1  0  0  0  0  0  0  2  0  5  1  3  5
##         20  0  0  0  0  0  0  2  0  2  0  1  0  0  0  1  0  2  0  3  8  2  2
##         21  0  0  0  0  0  0  3  0  0  0  0  0  0  2  0  0  0  0  1  1  9  2
##         22  1  0  0  1  0  0  0  0  0  0  1  1  0  0  0  0  0  0  1  0  0  2
##
## Overall Statistics
##
##                Accuracy : 0.472
##                  95% CI : (0.4275, 0.5168)
##     No Information Rate : 0.06
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.447
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.8182   0.8333   0.6471   0.4783   0.6538   0.2500
## Specificity            0.9749   0.9727   0.9337   0.9727   0.9852   0.9897
## Pos Pred Value         0.6000   0.6061   0.2558   0.4583   0.7083   0.4444
## Neg Pred Value         0.9915   0.9914   0.9869   0.9748   0.9811   0.9756
## Prevalence             0.0440   0.0480   0.0340   0.0460   0.0520   0.0320
## Detection Rate         0.0360   0.0400   0.0220   0.0220   0.0340   0.0080
## Detection Prevalence   0.0600   0.0660   0.0860   0.0480   0.0480   0.0180
## Balanced Accuracy      0.8965   0.9030   0.7904   0.7255   0.8195   0.6198
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.4815   0.5714   0.3704    0.6818    0.3333    0.3636
## Specificity            0.9619   0.9831   0.9958    0.9686    0.9809    0.9812
## Pos Pred Value         0.4194   0.6667   0.8333    0.5000    0.5263    0.4706
## Neg Pred Value         0.9701   0.9748   0.9652    0.9851    0.9584    0.9710
## Prevalence             0.0540   0.0560   0.0540    0.0440    0.0600    0.0440
## Detection Rate         0.0260   0.0320   0.0200    0.0300    0.0200    0.0160
## Detection Prevalence   0.0620   0.0480   0.0240    0.0600    0.0380    0.0340
## Balanced Accuracy      0.7217   0.7772   0.6831    0.8252    0.6571    0.6724
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity             0.5625    0.6400    0.5333    0.5385    0.2857
## Specificity             0.9607    0.9663    0.9876    0.9852    0.9645
## Pos Pred Value          0.3214    0.5000    0.5714    0.6667    0.2609
## Neg Pred Value          0.9852    0.9808    0.9856    0.9749    0.9686
## Prevalence              0.0320    0.0500    0.0300    0.0520    0.0420
## Detection Rate          0.0180    0.0320    0.0160    0.0280    0.0120
## Detection Prevalence    0.0560    0.0640    0.0280    0.0420    0.0460
## Balanced Accuracy       0.7616    0.8032    0.7605    0.7618    0.6251
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity             0.2400    0.2632    0.4000    0.3750    0.0800
## Specificity             0.9811    0.9626    0.9688    0.9811    0.9895
## Pos Pred Value          0.4000    0.2174    0.3478    0.5000    0.2857
## Neg Pred Value          0.9608    0.9706    0.9748    0.9689    0.9533
## Prevalence              0.0500    0.0380    0.0400    0.0480    0.0500
## Detection Rate          0.0120    0.0100    0.0160    0.0180    0.0040
## Detection Prevalence    0.0300    0.0460    0.0460    0.0360    0.0140
## Balanced Accuracy       0.6105    0.6129    0.6844    0.6780    0.5347
```

```r
### 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 training model=", tm_train[1], "s \n")
```

```
## Time for training model= 214.14 s
```

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

```
## Time for testing model= 0.109 s
```

You can see more methods and more details in doc/Additional Methods file and in Main.rmd file, including adaboost, xgboost, kpca, KSVM etc.