Facial Emotion Recognition by Linear Discriminant Analysis

Group 8

10/22/2019

Abstract

In this project, we created a classification engine for facial emotion recognition. After testing a variety of different models, we verified that using LDA with a PCA-driven dimensional reduction offered the best improvement over the baseline model of gradient boosting machines.

	baseline model	improved model
test accuracy	43.3%	51.2%
training feature construction	1.550s	0.230s
testing feature construction	0.132s	0.057s
training time	307.863s	10.953s
testing time	9.674s	0.004s

Step 1: set up controls for evaluation experiments

```
#setwd("/Users/mengyang/Documents/GitHub/fall2019-proj3-sec2--grp8/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 50 (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
##   1.545   0.257   1.809
##    user  system elapsed
##   0.132   0.064   0.196
```

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
## 309.336   4.075 627.069
```

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 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
##   9.986   0.237  10.239
```

```
#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] 0.4333333
```

```
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
##         1  13  0  2  1  0  0  0  0  0  5  0  0  1  0  0  1  0  1  0  0  0
##         2   0 12  0  0  0  0  0  3  5  0  0  0  0  0  0  0  0  0  0  3  0
##         3   3  0 14  1  0  1  0  1  0  3  0  0  0  0  0  3  0  0  1  0  0
##         4   2  0  2  8  0  3  2  0  0  5  0  1  4  0  0  0  0  0  0  0  0
##         5   0  0  0  0  8  0  2  0  0  0  0  0  0 10  3  0  1  3  0  0  0
##         6   2  0  2  1  0  8  1  0  0  0  0  2  2  0  0  0  0  0  0  0  0
##         7   0  0  0  0  1  0  4  0  0  0  0  0  0  1  0  1  3  0  3  0  4
##         8   0  4  0  0  0  0  0 12  2  0  0  0  0  0  0  0  4  0  0  1  0
##         9   0  2  0  0  1  1  0  5  4  1  0  0  0  0  0  0  0  0  0  3  1
##         10  1  0  4  4  0  0  0  0  0  4  0  1  3  0  0  0  0  0  0  0  1
##         11  2  0  3  1  0  6  0  0  0  0  3  4  1  0  0  0  0  0  0  1  0
##         12  3  0  6  0  0  4  0  0  1  2  2  4  4  0  0  2  0  0  1  0  0
##         13  1  0  2  2  0  0  0  0  1  1  2  1  1  0  0  0  0  0  0  1  1
```

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

Step 4: our improved model

Step 4.1: construct features and responses

In this part, we did feature selection by deleting the duplicated fiducial points on the right face. We did this because:

- 1. We can recognize someone's emotion only through his/ her left face and the middle part.
- 2. If one's face is not symmetric, then we believe the imbalance in the face is an error term in the model, and deleting the duplicates will not change the model.
- 3. If one's face is symmetric, then for the features construction, we only need the left face and the middle points. Because for any distances on the right, they have corresponding points on the left which give the same distances.
- 4. For any distance between a left point A and a right point B, its horizontal distance can be measured by (distance (A, middle) + distance (B's corresponding point on the left, middle)), and its vertical distances can be measured by (distance(A, B's corresponding point on the left))

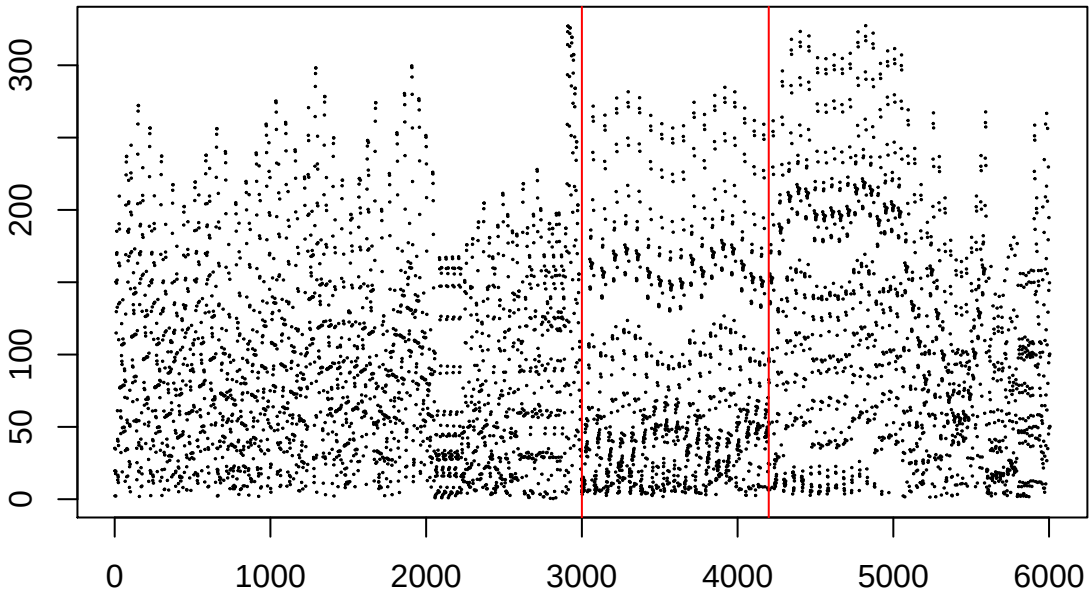
After deleting the 34 fiducial points on the right, we still have 44 fiducial points (34 on the left and 10 in the middle) for each image, which will result to $44 \times 43 = 1892$ features. 1892 is still too much for PCA, our

next step for dimensional reduction. Because when tuning the parameter (number of principle components selected) by 5-fold cross validation, we only have 1600 data (80% training dataset) to train the model, which are not enough (with p=1892 larger than n=1600) to construct a stable covariance matrix in PCA.

So in the next step, we deleted the duplicated 306 (34×9) features, because for each point among the 34 points on the left, the horizontal distances between it and the 10 points in the middle are all the same. We just kept one of them for each of the 34 left points to reduce the dimension.

We can also take a look at the following picture. It is the average 6006 features of emotion 1, and we can see some duplicates between the red lines, so it's reasonable to delete those duplicates.

```
plot(colMeans(dat_train_base[dat_train_base$emotion_idx=="1",-6007]),
     cex=.1, main="", ylab="", xlab="")
abline(v = c(3000, 4200), col=2)
```



The time for training features construction is around 0.23s, and it's around 0.06s for test features construction.

```
feature.construction.start = proc.time()
#delete the duplicated right 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,])})

#306 duplicates index
n.m <- 44
middle <- c(18:21,27,30,31,34,35,44)
d <- rep(1, 946)
m=matrix(rep(0, n.m^2),n.m,n.m,byrow=T)
k=1
for(i in 1:(n.m-1))for(j in (i+1):n.m){
  m[i,j] <- m[j,i] <- d[k]
  k=k+1
}
m[middle,] <- -1
m[,middle] <- -1
m[middle,middle] <- -2
ind <- c()
k=1
```

```
for(i in 1:(n.m-1)) for(j in (i+1):n.m){
  if(m[j,i] == -1) ind <- c(ind, k)
  k=k+1
}
dup_horiz <- ind[!(ind %in% cumsum(43:1)[-middle])]
```

```
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
##   0.261   0.052   0.313
##    user  system elapsed
##   0.049   0.006   0.056
```

Step 4.2: load features

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

Step 4.3: cross validation for number of principle components

We tuned the first 30, 50, 75, 120, 150, 200 principle components with proportions of variance of about 50%, 60%, 70%, 80% ,85% and 90% respectively.

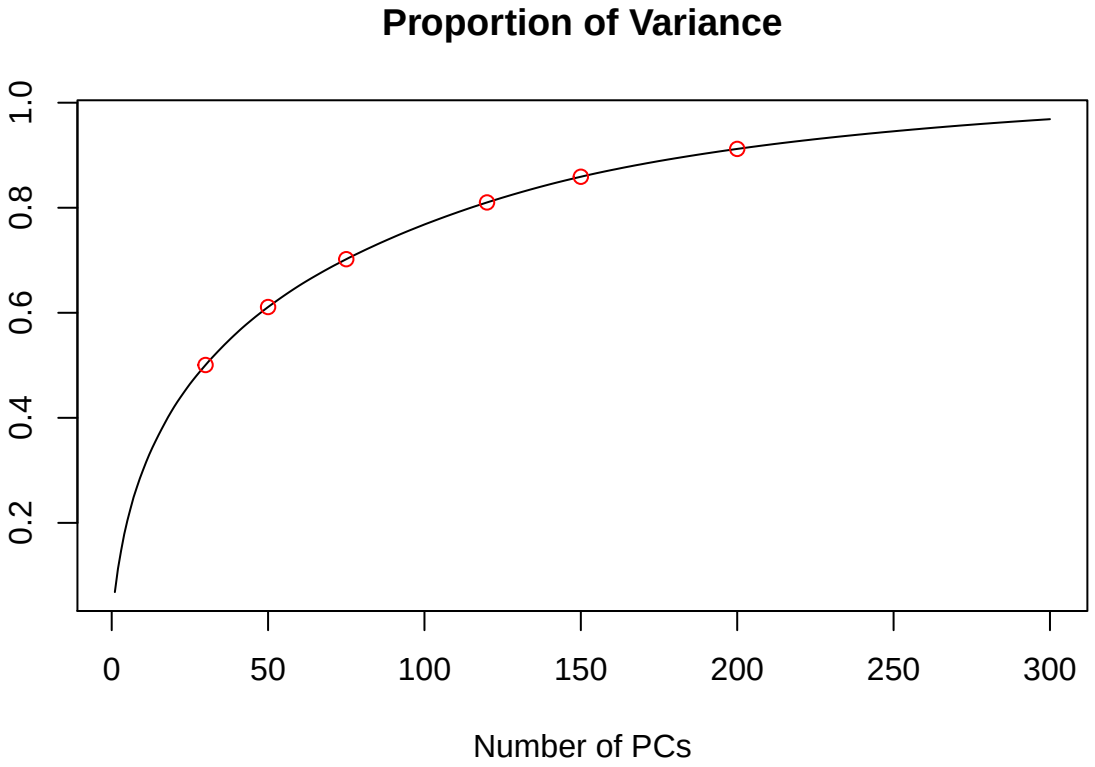
To choose the best number of principal components among these 6 numbers, we used 5-folder cross validation.

We got the highest cross validation accuracy with 50 PCs, and the cross validation accuracy is as below.

number of PCs	proportion of variance	cross validation accuracy
30	50.1%	44.3%
50	61.1%	46.6%
75	70.2%	45.9%
120	81.0%	45.6%
150	85.9%	45.6%
200	91.2%	43.3%

```
pca <- princomp(dat_train[,-1587], cor=T)
```

```
plot(cumsum(pca$sdev)[1:300] / sum(pca$sdev), type = "l",
     main = "Proportion of Variance", ylab = "", xlab = "Number of PCs")
n.pc.list <- c(30,50,75,120,150,200)
for (n.pc in n.pc.list) points(n.pc, (sum(pca$sdev[1:n.pc]) / sum(pca$sdev)), col="red")
```




```
if(run.cv){

set.seed(10)
cv.k <- createFolds(1:2000, k = K)
n.pc.list <- c(30,50,75,120,150,200)
cv.accuracy <- c()

for (f in cv.k){
  dat_train_cv <- dat_train[-f,]
  dat_validation_cv <- dat_train[f,]
  cv.tuning <- c()
  for (n.pc in n.pc.list){
    #pca
    pca <- princomp(dat_train_cv[,-1587], cor=T)
```

```
    train_x_pca <- data.frame(pca$scores[,1:n.pc])
    train_y <- dat_train_cv[1587]
    test_y <- dat_validation_cv[1587]
    colnames(train_x_pca) <- c(1:n.pc)
    train_pca <- cbind(train_x_pca, train_y)
    test_x_pca <- data.frame(predict(pca, dat_validation_cv[,-1587])[,1:n.pc])
    colnames(test_x_pca) <- c(1:n.pc)
    test_pca <- cbind(test_x_pca, test_y)
    train <- train_pca
    test <- test_pca
    #lda
    lda.model <- lda(emotion_idx ~ ., data=train)
    lda.test.pred = predict(lda.model, test[-dim(test)[2]])
    cv.tuning <- c(cv.tuning,
                   confusionMatrix(lda.test.pred$class, test$emotion_idx)$overall[1])
  }
  cv.accuracy <- cbind(cv.accuracy, cv.tuning)
}
colnames(cv.accuracy) <- 1:K
rownames(cv.accuracy) <- n.pc.list
cv.accuracy <- as.data.frame(cv.accuracy)
cv.accuracy$mean <- rowMeans(cv.accuracy)
cv.accuracy

}
```

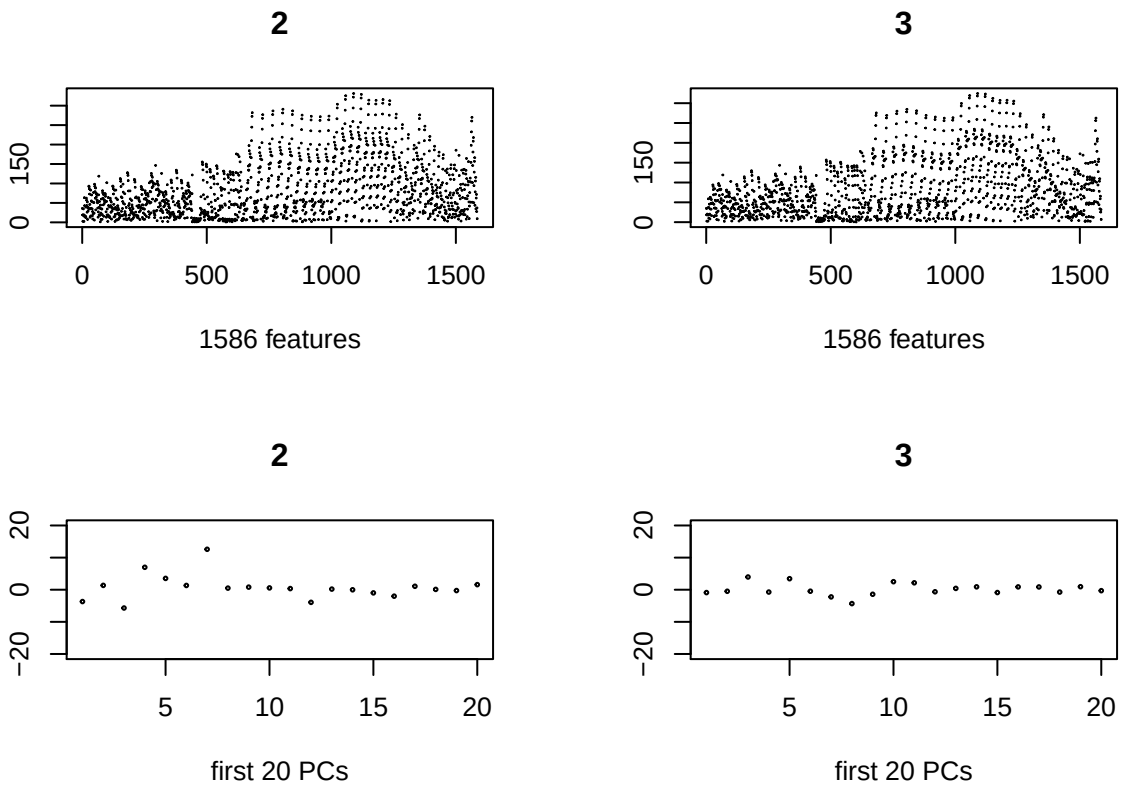
Step 4.4: principle components

The following pictures show us the features and first 20 principles components of emotion 2 and 3. In the feature plot, the difference of 1586 features between emotion 2 and 3 is a mess and it's not significant, but it becomes easier to detect if we use principle components as our features.

```
train.model.start = proc.time()
pca <- princomp(dat_train[,-1587], cor=T)
```

```
n.pc <- 50
train_x_pca <- data.frame(pca$scores[,1:n.pc])
train_y <- dat_train[1587]
test_y <- dat_test[1587]
colnames(train_x_pca) <- c(1:n.pc)
train_pca <- cbind(train_x_pca, train_y)
test_x_pca <- data.frame(predict(pca, dat_test[,-1587])[,1:n.pc])
colnames(test_x_pca) <- c(1:n.pc)
test_pca <- cbind(test_x_pca, test_y)
train <- train_pca
test <- test_pca
```

```
par(mfrow=c(2, 2))
for(i in 2:3) plot(colMeans(dat_train[dat_train$emotion_idx==as.character(i),-1587]),
                   cex=.07, main=i,ylab="",xlab="1586 features")
for(i in 2:3) plot(colMeans(train[train$emotion_idx==as.character(i),-51])[1:20],
                    cex=.3, main=i,ylab="",xlab="first 20 PCs",ylim=c(-20,20))
```



Step 4.5: linear discriminant analysis

The training time for our improved model is as below (around 11s).

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

```
##    user  system elapsed
##  11.321   0.131  11.480
```

The testing process took less than 0.01s, and the test accuracy is 51.2%, which is much higher than that of baseline model.

```
test.model.start = proc.time()
lda.test.pred = predict(lda.model, test[-dim(test)[2]])
test.model.end = proc.time()
#time for testing the model
print(test.model.end - test.model.start)
```

```
##    user  system elapsed
##   0.006   0.001   0.006
```

```
#test accuracy
confusionMatrix(lda.test.pred$class, test$emotion_idx)$overall[1]
```

```
## Accuracy
##    0.512
```

```r
confusionMatrix(lda.test.pred$class, test$emotion_idx)$table
```

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

Step 5: comparison and summary

Step 5.1: advantages of our improved model

- 1. High efficiency in both training and test step

It needs only 11s to train the model, and less than 0.01s to predict. The feature construction for training and test data is only 1.5s and 0.1s. These are much faster than almost any kind of boosting methods.

- 2. Much higher prediction accuracy compared to baseline model

Our model's prediction accuracy is 51.2%, but the baseline model is only 43.3%

- 3. Robustness

Step 5.2: comparison with quadratic discriminant analysis

LDA's training accuracy and prediciton accuracy are 57.2% and 51.2%, while QDA's are 99.5% and 20+%. It means QDA overfits the data.

It makes sense because in QDA model, every one of the 22 groups has a different covariance matrix, but in LDA, every group shares the same covariance matrix. So in QDA model, these covariance matrixs can be very different because some groups contain outliers and these outliers will result in a odd covariance matrix.

Acctually, the difference within each group should be the same, because the difference within a group comes from the difference of people's appearances in that group. However, the difference of people's appearances within each group should be similar. There's no reason that people in group 1 look similar, but people in group 2 look different.

Step 5.3: comparison with boosting

It needs only 11s to train the model, and less than 0.01s to predict. The feature construction for training and test data is only 1.5s and 0.1s. These are much faster than almost any kind of boosting methods.

What is more, boosting is sensitive to outliers, which means boosting methods are not as robust. In most boosting methods, the difference in training and test accuracy is larger than 20%, but in our model, the difference is only 6%. Such difference may not show how good a model is, but it may show the robustness in some way.

Step 6: possible improvements

We are aware that in the real world a prediction accuracy of 50% can result in rather conterproductive outcomes. Selling this final product to an actual company could prove to be difficult because companies cannot afford to be wrong 50% of the time. Therefore, a potential way to further improve our model and make it competitive on the market would involve using a two-layer model. In fact, we have noticed that emotions 19, 20, 21, 22 are the most prone to misclassification in our current model. Let's say we use SVM as the second layer to reclassify misclassified data given by LDA. If SVM outperforms LDA in some typical classification, then the second layer will improve the accuracy.