# Project 3: Facial Emotion Recognition

Group 9

3/30/2020

**Abstract**

| | baseline model | improved model |
|---|---|---|
| test accuracy | 46.6% | 54.6% |
| training time | 1816.61s | 39s |
| testing time | 13.199s | 0.264s |

(There might be some slight difference of time and accuracy on different computers, so the data above just give you a sense of our model. )

The goal of this project is to predict facial emotion. After trying multiple machine learning models such as Gradient Boosting Machine (GBM), Linear discriminant analysis (LDA), Support Vector Machine (SVM), Random Forest (RF), XGBoost (XGB), ensemble model of SVM and XGB, and Neural Network (NN), we found that using ensemble model has better effect. The comparison bewteen baseline model (GBM) and improved model (NN) is shown as below.

### Step 1: set up work directory

Before reproducing the project, please set your work directory.

```
set.seed(5)
seed <- .Random.seed
setwd("/Users/rachel/Documents/GitHub/Spring2020-Project3-ads-spring2020-project3-group9/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.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 data split

We split 80% of data into training and 20% into testing. For baseline model - GBM, and other models such as SVM, XGBoost, RF, and NN, we only used the fiducial points extracted from the images.

```
### set 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)
```

```
### read fiducial points
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: Feature Extraction

Based on the fiducial points, we calculated the pairwise distances between fiducial points and utilized each distance as a feature. We didn't construct or remove any feature in this step as the fiducial points are extracted features from the original images. That said, the information we can obtain from the fiducial points is less than that from the images. If we continue removing some points or information, there is less data for models to train. But we conducted the principal component analysis(PCA) to reduce the dimensions in further steps.

```
### calculate the pairwise distances between fiducial points
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.test){
  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")

#load(file="../output/feature_train.RData")
#load(file="../output/feature_test.RData")
```

### Step 4: Baseline Model - Gradient Boosting Machine (GBM)

#### Step 4.1: Build model and fit on train data

To keep this main file simple, we removed the cross validation and parameters tuning step to reduce the reproduction time. After conduction cross validation, the hyperparameters we used for GBM model are listed as followed.

- n.trees = 300 – number of iterations/trees
- shrinkage = 0.15 – learning rate
- interaction.depth = 2 – number of splits on a tree
- n.minobsinnode = 10 – minimum number of observations in trees' terminal nodes

```
### load models built
source("../lib/train_gbm.R")     ### train model
```

```
### fit train data
tm_train=NA
tm_train_gbm <- system.time(fit_train_gbm <- train_gbm(feature_df = dat_train))
save(fit_train_gbm, file="../output/gbm_train.RData")
```

Please see the details of model as below.

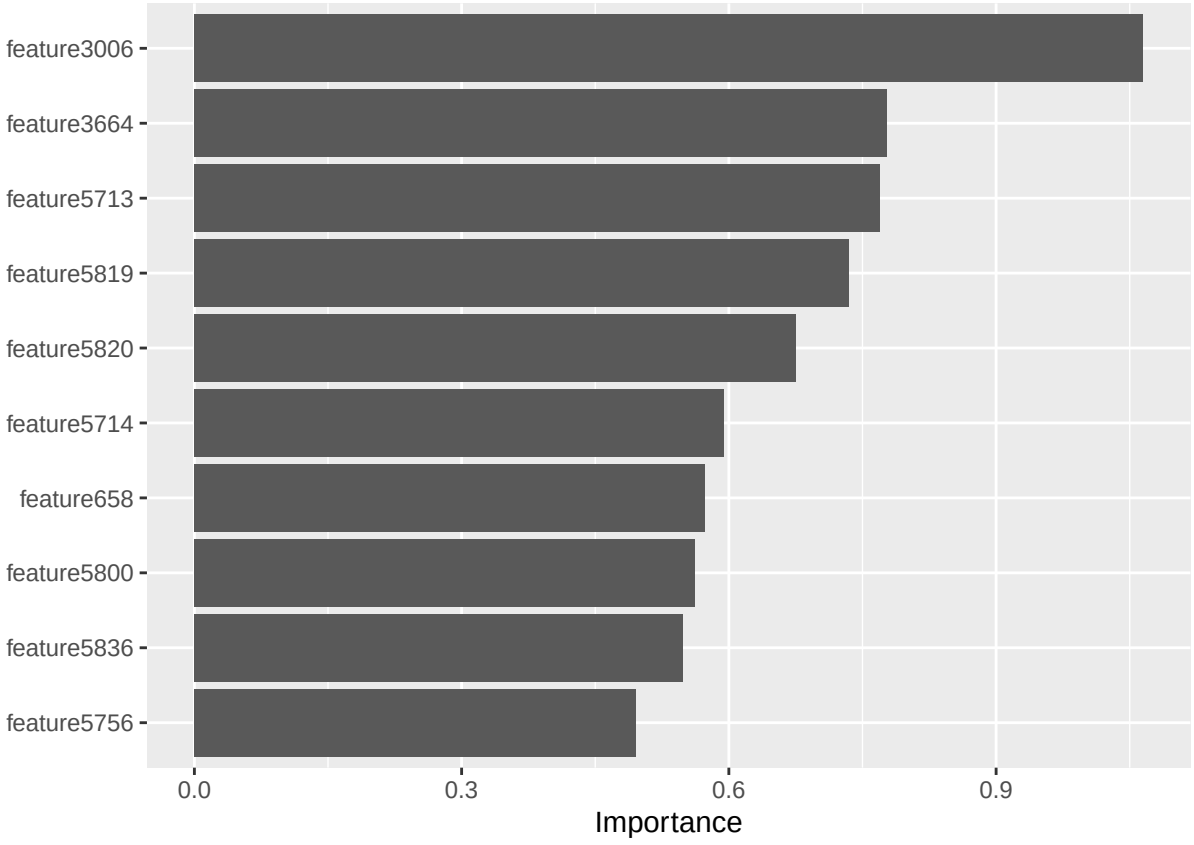
```
fit_train_gbm
```

```
## gbm(formula = emotion_idx ~ ., distribution = "multinomial",
##     data = feature_df, n.trees = 300, interaction.depth = 2,
##     n.minobsinnode = 10, shrinkage = 0.15)
## A gradient boosted model with multinomial loss function.
## 300 iterations were performed.
## There were 6006 predictors of which 3573 had non-zero influence.
```

### Step 4.2: Visualize variables importances of train model

After running our baseline model, we wanted to understand the variables that influence the emotion prediction largely. So, we plotted the most influential variables. Here, we utilized the default method - relative influence to compute the variables importance, which shows the average improvement obtained by each variable across all trees that use the variable.

```
vip::vip(fit_train_gbm)
```




### Step 4.3: Predict test data and evaluate

Calculate the training error of GBM model

```
source("../lib/test_gbm.R")
pred_train_gbm <- test_gbm(fit_train_gbm, dat_train)
accu_train_gbm <- mean(pred_train_gbm == dat_train$emotion_idx)
cat("The trainig accuracy of model: GBM", "is", accu_train_gbm*100, "%.\n")
```

```
## The trainig accuracy of model: GBM is 100 %.
```

Then, we predicted the test data and evaluated the performance of model.

```
### load models built
source("../lib/test_gbm.R")     ### test model

### predict test data
tm_test_gbm=NA
if(run.test){
  load(file="../output/gbm_train.RData")
  tm_test_gbm <- system.time(pred <- test_gbm(model_best = fit_train_gbm, feature_test = dat_test))
}
```

Evaluation on GBM model

```
pred_gbm = test_gbm(fit_train_gbm, dat_test)
accu_gbm <- mean(dat_test$emotion_idx == pred_gbm)
cat("The accuracy of model: gradient boosting machine", "is", accu_gbm*100, "%.\n")
```

```
## The accuracy of model: gradient boosting machine is 42 %.
```

```
pred_gbm = as.factor(pred_gbm)
confusionMatrix(pred_gbm, 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 22
##         1  12  0  3  0  0  0  0  0  0  1  0  0  0  0  0  2  0  0  0  0  0  1
##         2   0 13  0  0  0  0  0  2  6  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3   6  0 14  2  0  0  0  0  0  7  0  0  0  0  0  2  0  0  0  0  0  1
##         4   1  0  1 11  0  1  0  0  0  1  3  1  2  0  0  0  0  0  0  0  0  1
##         5   0  0  0  0 15  0  1  0  0  0  0  0  0  0  3  0  3  1  1  0  0  0
##         6   0  0  0  2  0  8  1  0  0  1  3  2  1  0  0  0  0  0  0  0  0  1
##         7   0  0  0  0  2  0 15  2  2  0  0  0  0  0  1  0  0  0  0  0  0  0
##         8   0  0  0  0  0  0  0 19  1  0  0  0  0  0  1  0  1  0  0  1  1  0
##         9   0  5  0  0  0  0  0  0  9  0  0  0  0  0  0  0  0  0  0  0  1  0
##         10  1  0  4  6  0  1  1  0  0 10  2  2  4  0  0  1  0  0  0  0  0  2
##         11  0  0  0  2  0  1  0  0  1  1  6  2  1  0  0  1  0  0  0  0  1  0
##         12  0  0  1  1  0  1  0  0  0  0  3  2  5  0  0  2  0  0  0  0  0  0
##         13  2  0  2  2  0  2  1  0  0  2  6  6  3  0  0  0  0  0  0  0  2  1
##         14  0  0  0  1  0  0  1  1  0  0  1  0  1 13  2  0  0  0  3  0  2  0
##         15  0  0  0  0  3  0  0  0  0  1  0  1  0  5  5  0  0  0  0  0  0  0
##         16  0  0  1  0  0  2  0  0  0  0  0  0  0  0  0 13  0  0  0  1  0  0
##         17  0  0  0  0  2  0  2  0  0  0  0  0  0  1  0  1 11 10  0  2  0  1
##         18  1  0  0  0  5  0  1  0  0  0  0  0  0  0  0  0  1  6  0  1  0  0
##         19  0  0  0  0  0  0  5  1  0  0  0  0  0  2  3  1  3  2  3  4  1  3
##         20  0  2  0  0  0  0  0  1  3  0  0  0  0  0  2  0  0  0  0  9  2  3
##         21  0  0  0  0  0  4  0  0  3  0  0  2  1  3  2  0  3  0  6  4  9  2
##         22  0  0  1  0  0  0  0  0  0  0  1  2  0  0  1  1  0  0  3  3  1  4
##
```

```
## Overall Statistics
##
##                Accuracy : 0.42
##                  95% CI : (0.3763, 0.4646)
##     No Information Rate : 0.056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3924
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.5217   0.6500   0.5185   0.4074   0.5556   0.4000
## Specificity            0.9853   0.9833   0.9619   0.9767   0.9810   0.9771
## Pos Pred Value         0.6316   0.6190   0.4375   0.5000   0.6250   0.4211
## Neg Pred Value         0.9771   0.9854   0.9722   0.9665   0.9748   0.9751
## Prevalence             0.0460   0.0400   0.0540   0.0540   0.0540   0.0400
## Detection Rate         0.0240   0.0260   0.0280   0.0220   0.0300   0.0160
## Detection Prevalence   0.0380   0.0420   0.0640   0.0440   0.0480   0.0380
## Balanced Accuracy      0.7535   0.8167   0.7402   0.6921   0.7683   0.6885
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.5357   0.7308   0.3600    0.4167    0.2400    0.1000
## Specificity            0.9852   0.9895   0.9874    0.9496    0.9789    0.9729
## Pos Pred Value         0.6818   0.7917   0.6000    0.2941    0.3750    0.1333
## Neg Pred Value         0.9728   0.9853   0.9670    0.9700    0.9607    0.9629
## Prevalence             0.0560   0.0520   0.0500    0.0480    0.0500    0.0400
## Detection Rate         0.0300   0.0380   0.0180    0.0200    0.0120    0.0040
## Detection Prevalence   0.0440   0.0480   0.0300    0.0680    0.0320    0.0300
## Balanced Accuracy      0.7604   0.8601   0.6737    0.6831    0.6095    0.5365
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity             0.1667    0.5417    0.2500    0.5417    0.5000
## Specificity             0.9461    0.9748    0.9792    0.9916    0.9603
## Pos Pred Value          0.1034    0.5200    0.3333    0.7647    0.3667
## Neg Pred Value          0.9682    0.9768    0.9691    0.9772    0.9766
## Prevalence              0.0360    0.0480    0.0400    0.0480    0.0440
## Detection Rate          0.0060    0.0260    0.0100    0.0260    0.0220
## Detection Prevalence    0.0580    0.0500    0.0300    0.0340    0.0600
## Balanced Accuracy       0.5564    0.7582    0.6146    0.7666    0.7301
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity             0.3158    0.1875    0.3600    0.4500    0.2000
## Specificity             0.9813    0.9483    0.9726    0.9375    0.9729
## Pos Pred Value          0.4000    0.1071    0.4091    0.2308    0.2353
## Neg Pred Value          0.9732    0.9725    0.9665    0.9761    0.9669
## Prevalence              0.0380    0.0320    0.0500    0.0400    0.0400
## Detection Rate          0.0120    0.0060    0.0180    0.0180    0.0080
## Detection Prevalence    0.0300    0.0560    0.0440    0.0780    0.0340
## Balanced Accuracy       0.6485    0.5679    0.6663    0.6937    0.5865
```

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

```
## Time for training model GBM =  1811.846 s
```

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

```
## Time for testing model GBM =  13.356 s
```

### Step 4.4: Summary

Overall, the predictive accuracy of GBM is not bad. Also, it doesn't need data pre-processing and handles multiple categorical value well. However, it is computational expensive to build many trees and tune multiple parameters based on cross validation. The result of model is less interpretable and easily understood as we built models on fiducial points distances. To improve the accuracy and eliminate other disadvantages of GBM, we also tried other models such as LDA, SVM, RF, and XGBoost.

## Step 5: Improved Models - Linear Discriminant Analysis (LDA), Support Vector Machine (SVM), Random Forest, Extreme Gradient Boosting (XGBoost), ensemble model of SVM and XGB, Neural Network (NN)

### Step 5.1: Linear Discriminant Analysis (LDA) with Principal Component Analysis (PCA)

#### PCA

Extract first principal component from feature data

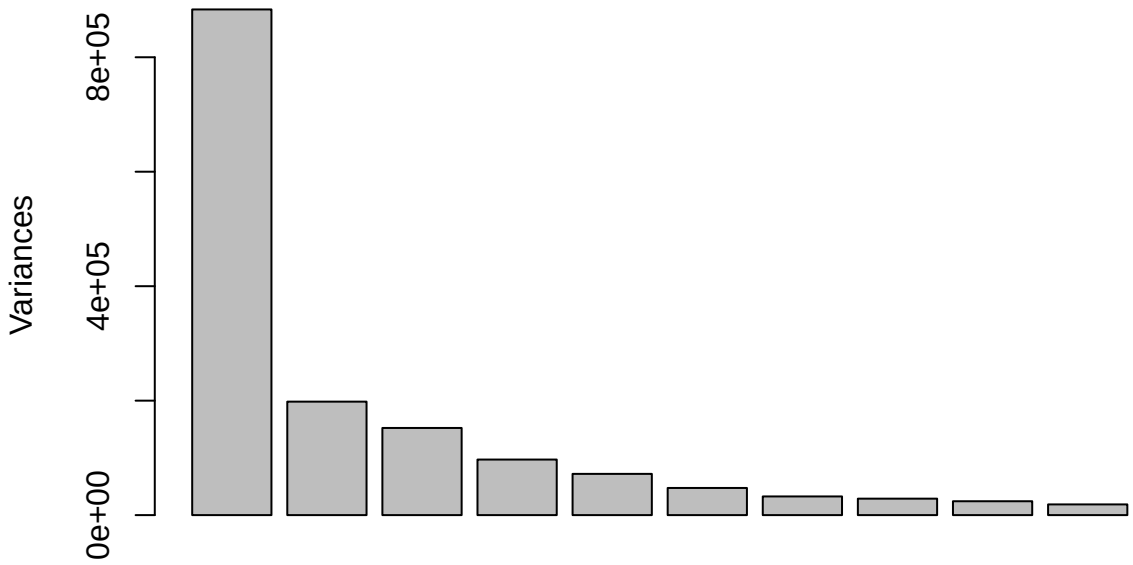
```
#load(file="../output/feature_train.RData")
#load(file="../output/feature_test.RData")

## Convert dat_train to numeric data.frame
dat_train.new <- matrix(0, ncol = ncol(dat_train) - 1, nrow = nrow(dat_train))
for (i in 1:(ncol(dat_train) - 1)) {
  dat_train.new[,i] <- as.numeric(dat_train[[i]])
}
dat_train.new <- as.data.frame(dat_train.new)

## PCA for training features
source("../lib/train_pca.R")
tm_train_pca <- NA
tm_train_pca <- system.time(fit_train_pca <- train_pca(dat_train.new))
save(fit_train_pca, file="../output/pca_train.RData")
```

Visualization of proportion of variance to choose number of principle components

```
screeplot(fit_train_pca)
```




```
## The proportion of variance for first 500 PCs
sum((fit_train_pca$sdev[1:500])^2) / sum((fit_train_pca$sdev)^2)
```

```
## [1] 0.9997597
```

```
## Combine pc_features with the emption index
dat_train_pca <- data.frame(fit_train_pca$x[,1:500], emotion_idx = dat_train[,6007])
```

Apply PC model to test data

```
## Extract PC from test data
source("../lib/test_pca.R")
dat_test.new <- dat_test
colnames(dat_test.new) <- c(colnames(dat_train.new), "emotion_idx")
tm_test_pcs <- NA
tm_test_pca <- system.time(dat_test.new <- test_pca(fit_train_pca, dat_test.new))

## Combine pc_features with the emption index
dat_test_pca <- data.frame(dat_test.new[,1:500], emotion_idx = dat_test[,6007])


save(dat_train_pca, file="../output/feature_train_pca.RData")
save(dat_test_pca, file="../output/feature_test_pca.RData")

#load("../output/feature_train_pca.RData")
#load("../output/feature_test_pca.RData")
```

Evaluation on PCA model

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

```
## Time for training model PCA =  87.942 s
```

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

```
## Time for testing model PCA =  4.267 s
```

### LDA

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

```
source("../lib/train_lda.R")
tm_train_lda = NA
tm_train_lda <- system.time(fit_train_lda <- train_lda(dat_train_pca, par_best))
save(fit_train_lda, file="../output/lda_train.RData")
```

Calculate the training error of LDA model

```
source("../lib/test_lda.R")
pred_train_lda <- test_lda(fit_train_lda, dat_train_pca)
accu_train_lda <- mean(pred_train_lda == dat_train_pca$emotion_idx)
cat("The trainig accuracy of model: LDA", "is", accu_train_lda*100, "%.\n")
```

```
## The trainig accuracy of model: LDA is 90.8 %.
```

Predicted the test data using LDA model

```
tm_test_lda = NA
if(run.test){
  load(file="../output/lda_train.RData")
  tm_test_lda <- system.time(pred_lda <- test_lda(fit_train_lda, dat_test_pca))
}
```

Evaluation on LDA model

```
accu_lda <- mean(dat_test_pca$emotion_idx == pred_lda)
cat("The accuracy of model: LDA", "is", accu_lda*100, "%.\n")
```

```
## The accuracy of model: LDA is 46.4 %.
```

```
confusionMatrix(pred_lda, dat_test_pca$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  17  0  2  3  1  0  0  0  0  1  0  1  0  1  0  2  0  0  0  0  0  0
##         2   0 13  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3   0  0 15  0  0  0  0  0  0  2  2  2  0  0  0  1  0  0  0  0  1  1
##         4   1  0  2  9  0  2  0  0  0  0  3  0  5  0  0  0  0  0  0  0  1  0
##         5   0  0  0  0 18  0  2  1  0  1  0  0  0  2  0  0  1  0  0  0  0  0
##         6   0  0  0  1  0  7  0  0  2  1  2  1  1  0  0  1  0  0  0  1  0  1
##         7   0  1  1  0  0  0 16  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0
##         8   0  2  0  0  0  0  0 22  1  0  0  0  0  0  0  0  1  0  0  1  0  0
##         9   0  2  0  0  0  0  0  1 15  0  0  0  0  0  0  0  0  0  0  0  1  1
##         10  1  0  2  3  0  1  0  0  0 11  0  1  1  1  0  0  0  0  0  0  1  0
##         11  0  0  0  1  0  0  0  0  0  0  9  1  1  0  0  0  0  0  0  0  0  0
##         12  0  0  2  3  0  3  0  0  0  2  5  6  2  0  0  1  0  0  0  0  1  1
##         13  0  0  0  5  0  1  0  0  0  2  4  4  6  0  1  0  0  0  1  0  3  2
##         14  0  0  0  0  0  0  1  1  0  0  0  1  0 13  4  0  0  1  1  0  0  0
##         15  1  0  0  0  2  0  0  0  0  0  0  0  0  2  8  0  1  1  2  0  0  0
##         16  1  0  1  0  0  1  1  0  0  0  0  1  0  0  0 13  1  0  0  3  0  1
##         17  0  0  0  0  1  0  1  0  0  0  0  0  0  1  0  0  8  7  0  4  0  0
##         18  0  0  0  0  5  0  1  1  0  0  0  0  0  0  0  2  2  6  0  3  0  0
##         19  2  0  1  1  0  2  4  0  0  0  0  0  0  1  3  0  1  2  4  7  2  3
##         20  0  1  0  0  0  1  1  0  2  0  0  0  0  1  2  0  2  0  0  3  1  1
```

```
##         21  0  1  0  0  0  1  1  0  2  2  0  0  1  2  2  0  4  1  6  2  9  5
##         22  0  0  1  1  0  1  0  0  2  2  0  2  1  0  0  4  0  0  2  1  0  4
##
## Overall Statistics
##
##                Accuracy : 0.464
##                  95% CI : (0.4196, 0.5088)
##     No Information Rate : 0.056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4388
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.7391   0.6500   0.5556   0.3333   0.6667   0.3500
## Specificity            0.9769   0.9979   0.9810   0.9704   0.9852   0.9771
## Pos Pred Value         0.6071   0.9286   0.6250   0.3913   0.7200   0.3889
## Neg Pred Value         0.9873   0.9856   0.9748   0.9623   0.9811   0.9730
## Prevalence             0.0460   0.0400   0.0540   0.0540   0.0540   0.0400
## Detection Rate         0.0340   0.0260   0.0300   0.0180   0.0360   0.0140
## Detection Prevalence   0.0560   0.0280   0.0480   0.0460   0.0500   0.0360
## Balanced Accuracy      0.8580   0.8240   0.7683   0.6519   0.8259   0.6635
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.5714   0.8462   0.6000    0.4583    0.3600    0.3000
## Specificity            0.9915   0.9895   0.9895    0.9769    0.9937    0.9583
## Pos Pred Value         0.8000   0.8148   0.7500    0.5000    0.7500    0.2308
## Neg Pred Value         0.9750   0.9915   0.9792    0.9728    0.9672    0.9705
## Prevalence             0.0560   0.0520   0.0500    0.0480    0.0500    0.0400
## Detection Rate         0.0320   0.0440   0.0300    0.0220    0.0180    0.0120
## Detection Prevalence   0.0400   0.0540   0.0400    0.0440    0.0240    0.0520
## Balanced Accuracy      0.7815   0.9178   0.7947    0.7176    0.6768    0.6292
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity             0.3333    0.5417    0.4000    0.5417    0.3636
## Specificity             0.9523    0.9811    0.9812    0.9790    0.9707
## Pos Pred Value          0.2069    0.5909    0.4706    0.5652    0.3636
## Neg Pred Value          0.9745    0.9770    0.9752    0.9769    0.9707
## Prevalence              0.0360    0.0480    0.0400    0.0480    0.0440
## Detection Rate          0.0120    0.0260    0.0160    0.0260    0.0160
## Detection Prevalence    0.0580    0.0440    0.0340    0.0460    0.0440
## Balanced Accuracy       0.6428    0.7614    0.6906    0.7603    0.6672
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity             0.3158    0.2500    0.1200    0.4500    0.2000
## Specificity             0.9709    0.9401    0.9747    0.9375    0.9646
## Pos Pred Value          0.3000    0.1212    0.2000    0.2308    0.1905
## Neg Pred Value          0.9729    0.9743    0.9546    0.9761    0.9666
## Prevalence              0.0380    0.0320    0.0500    0.0400    0.0400
## Detection Rate          0.0120    0.0080    0.0060    0.0180    0.0080
## Detection Prevalence    0.0400    0.0660    0.0300    0.0780    0.0420
## Balanced Accuracy       0.6433    0.5950    0.5474    0.6937    0.5823
```

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

```
## Time for training model LDA =  2.576 s
```

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

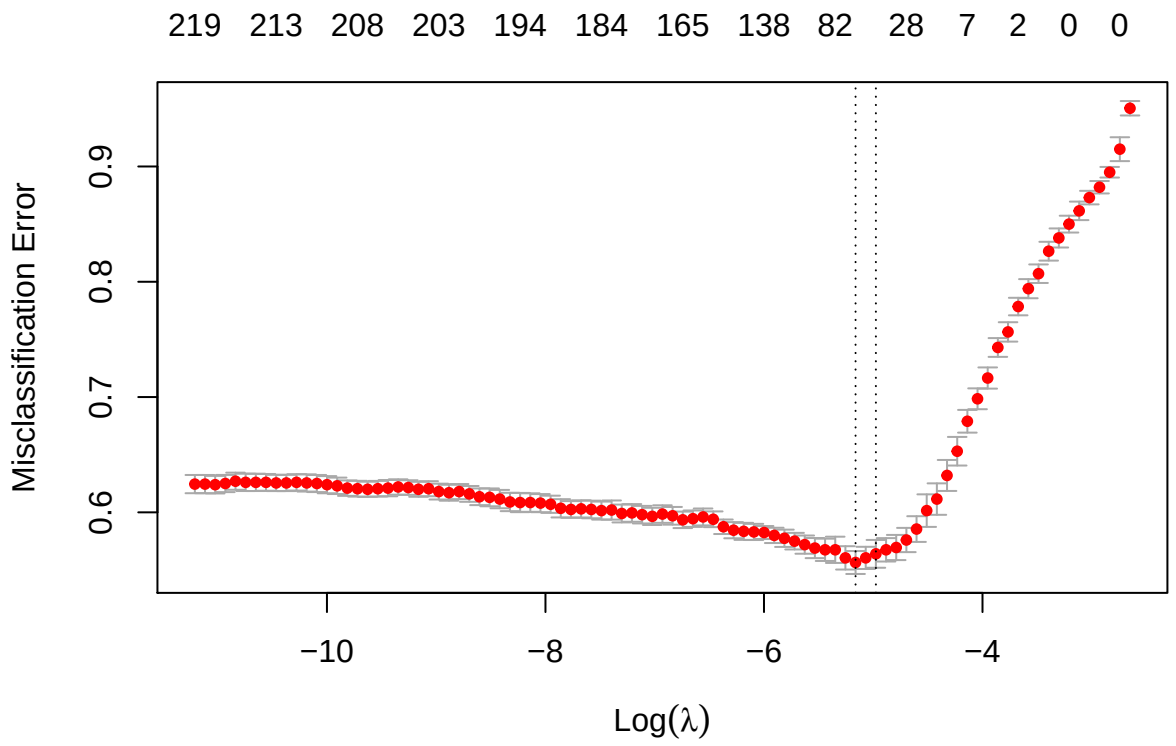
```
## Time for testing model LDA =  0.033 s
```

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

### Step 5.2: Logistic Regression Model (LR) with Principal Component Analysis (PCA)

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

```
source("../lib/train_lr.R")
tm_train_lr=NA
tm_train_lr <- system.time(cv_fit_train_lr <- train_lr(feature_df = dat_train_pca))
plot(cv_fit_train_lr)
```




```
opt_lambda <- cv_fit_train_lr$lambda.1se

fit_train_lr <- glmnet(as.matrix(dat_train_pca[, 1:500]),
                       factor(dat_train_pca$emotion_idx),
                       family = "multinomial",
                alpha = 0,
                lambda = opt_lambda)
save(fit_train_lr, file="../output/lr_train.RData")
```

Calculate the training error of LR model

```
source("../lib/test_lr.R")
pred_train_lr <- test_lr(fit_train_lr,opt_lambda, dat_train_pca)
```

```
accu_train_lr <- mean(pred_train_lr == dat_train$emotion_idx)
cat("The trainig accuracy of model: LR", "is", accu_train_lr*100, "%.\n")
```

```
## The trainig accuracy of model: LR is 4.6 %.
```

Predicted the test data using LR model

```
tm_test_lr = NA
if(run.test){
  load(file="../output/lr_train.RData")
  tm_test_lr <- system.time(pred_lr <- test_lr(fit_train_lr,opt_lambda, dat_test_pca))
}
```

Evaluation on LR model

```
accu_lr <- mean(dat_test_pca$emotion_idx == pred_lr)
cat("The accuracy of model: LR", "is", accu_lr*100, "%.\n")
```

```
## The accuracy of model: LR is 4 %.
```

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

```
## Time for training model LR =  422.817 s
```

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

```
## Time for testing model LR =  0.085 s
```

### Step 5.3: Support Vector Machine (SVM)

Then, we tried Support Vector Machine Model. An SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

The global argument we need to consider is stated below:

- kernel = "linear" – the kernel used in training and predicting
- cost = 0.02 – cost of constraints violation(default: 1)

We tried linear, polynomial and radial basis kernels seperately , expolred different parameters(degree for polynomial kernel and gamma for radial basis kernel) for this model and found the linear kernel for SVM model fit the data best.

```
source("../lib/train_svm.R")
tm_train_svm <- NA
tm_train_svm <- system.time(fit_train_svm <- train_svm(dat_train, probability = TRUE))
save(fit_train_svm, file="../output/svm_train.RData")
```

Calculate the training error of SVM model

```
source("../lib/test_svm.R")
pred_train_svm <- test_svm(fit_train_svm, dat_train)
accu_train_svm <- mean(pred_train_svm == dat_train$emotion_idx)
cat("The trainig accuracy of model: SVM", "is", accu_train_svm*100, "%.\n")
```

```
## The trainig accuracy of model: SVM is 100 %.
```

Predicted the test data using SVM model

```
tm_test_svm <- NA
if(run.test){
  load(file="../output/svm_train.RData")
```

```
  tm_test_svm <- system.time(pred_svm <- test_svm(fit_train_svm, dat_test, probability = TRUE))
}
```

Evaluation on SVM model

```
accu_svm <- mean(dat_test$emotion_idx == pred_svm)
cat("The accuracy of model: SVM", "is", accu_svm*100, "%.\n")
```

```
## The accuracy of model: SVM is 49.8 %.
```

```
confusionMatrix(pred_svm, 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 22
##         1  18  0  0  2  0  0  0  0  0  2  0  0  0  0  0  1  0  0  0  0  0  0
##         2   0 12  0  0  0  1  0  2  1  0  0  0  0  0  0  0  0  0  0  1  0  0
##         3   2  0 15  1  0  0  0  0  0  5  0  0  0  0  0  1  0  0  1  0  0  1
##         4   1  0  2 15  0  1  0  0  0  1  3  1  2  0  0  0  0  0  0  0  1  1
##         5   0  0  0  0 22  0  1  0  0  0  0  0  0  0  4  0  0  4  0  0  0  0
##         6   0  0  0  0  0  8  0  0  2  2  1  1  1  0  0  0  0  0  0  0  0  1
##         7   0  0  1  0  3  0 16  1  1  0  0  0  0  0  0  2  1  0  0  1  0  0
##         8   0  0  0  0  0  0  1 21  1  0  0  0  0  0  0  0  1  0  0  1  0  0
##         9   0  6  0  0  0  0  0  1 16  0  0  0  0  0  0  0  0  0  0  0  1  0
##         10  1  0  2  3  0  2  0  0  0 11  1  1  2  1  0  0  0  0  0  0  1  2
##         11  0  0  1  2  0  0  0  0  0  1  9  3  1  1  0  1  0  0  0  0  0  0
##         12  0  0  3  1  0  4  0  0  0  1  5  5  2  0  0  1  0  0  0  1  0  1
##         13  0  0  0  2  0  0  0  0  0  0  6  4  6  0  0  0  0  0  0  0  1  1
##         14  0  0  0  0  0  0  1  0  0  0  0  1  2 17  4  0  0  0  1  0  2  0
##         15  0  0  0  0  0  0  1  0  0  0  0  0  0  2  7  0  0  0  2  0  1  0
##         16  0  0  1  1  0  2  0  0  0  0  0  1  0  0  0 14  1  0  0  1  0  0
##         17  0  0  0  0  1  0  0  1  0  0  0  0  0  2  1  0 12  7  0  2  0  0
##         18  1  0  1  0  1  0  1  0  0  0  0  0  0  0  0  0  3  6  0  4  0  0
##         19  0  0  0  0  0  0  2  0  0  0  0  0  0  0  3  0  0  2  4  3  2  4
##         20  0  1  0  0  0  0  3  0  3  0  0  0  0  1  0  0  1  0  0  6  2  2
##         21  0  1  0  0  0  1  2  0  0  0  0  1  1  0  1  0  3  0  7  5  7  5
##         22  0  0  1  0  0  1  0  0  1  1  0  2  1  0  0  4  0  0  1  0  2  2
##
## Overall Statistics
##
##                Accuracy : 0.498
##                  95% CI : (0.4533, 0.5427)
##     No Information Rate : 0.056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4737
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.7826   0.6000   0.5556   0.5556   0.8148   0.4000
## Specificity            0.9895   0.9896   0.9767   0.9725   0.9810   0.9833
```

```
## Pos Pred Value         0.7826   0.7059   0.5769   0.5357   0.7097   0.5000
## Neg Pred Value         0.9895   0.9834   0.9747   0.9746   0.9893   0.9752
## Prevalence             0.0460   0.0400   0.0540   0.0540   0.0540   0.0400
## Detection Rate         0.0360   0.0240   0.0300   0.0300   0.0440   0.0160
## Detection Prevalence   0.0460   0.0340   0.0520   0.0560   0.0620   0.0320
## Balanced Accuracy      0.8861   0.7948   0.7661   0.7640   0.8979   0.6917
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.5714   0.8077   0.6400    0.4583    0.3600    0.2500
## Specificity            0.9788   0.9916   0.9832    0.9664    0.9789    0.9604
## Pos Pred Value         0.6154   0.8400   0.6667    0.4074    0.4737    0.2083
## Neg Pred Value         0.9747   0.9895   0.9811    0.9725    0.9667    0.9685
## Prevalence             0.0560   0.0520   0.0500    0.0480    0.0500    0.0400
## Detection Rate         0.0320   0.0420   0.0320    0.0220    0.0180    0.0100
## Detection Prevalence   0.0520   0.0500   0.0480    0.0540    0.0380    0.0480
## Balanced Accuracy      0.7751   0.8996   0.8116    0.7124    0.6695    0.6052
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity             0.3333    0.7083    0.3500    0.5833    0.5455
## Specificity             0.9710    0.9769    0.9875    0.9853    0.9707
## Pos Pred Value          0.3000    0.6071    0.5385    0.6667    0.4615
## Neg Pred Value          0.9750    0.9852    0.9733    0.9791    0.9789
## Prevalence              0.0360    0.0480    0.0400    0.0480    0.0440
## Detection Rate          0.0120    0.0340    0.0140    0.0280    0.0240
## Detection Prevalence    0.0400    0.0560    0.0260    0.0420    0.0520
## Balanced Accuracy       0.6521    0.8426    0.6687    0.7843    0.7581
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity             0.3158    0.2500    0.2400    0.3500    0.1000
## Specificity             0.9771    0.9669    0.9726    0.9437    0.9708
## Pos Pred Value          0.3529    0.2000    0.3158    0.2059    0.1250
## Neg Pred Value          0.9731    0.9750    0.9605    0.9721    0.9628
## Prevalence              0.0380    0.0320    0.0500    0.0400    0.0400
## Detection Rate          0.0120    0.0080    0.0120    0.0140    0.0040
## Detection Prevalence    0.0340    0.0400    0.0380    0.0680    0.0320
## Balanced Accuracy       0.6465    0.6085    0.6063    0.6469    0.5354
```

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

```
## Time for training model SVM =  204.214 s
```

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

```
## Time for testing model SVM =  10.126 s
```

### Step 5.4: Random Forest

Random Forest is another approach that we consider as a candidate of our advanced model. It is a non-parametric model that would give us more flexibility but along with the risk of overfitting. Random forest is a popular algorithm for classification in a long time, and it dominant a lot of other approaches in application.

However, tuning a random forest model is not easy, the global argument we need to consider is stated below:

- ntrees = 500 – the number of trees in the forest
- mtry = 77 – the number of features to consider when looking for the best split
- max_depth – the maximum depth of each tree(by default)

To tune these three parameters using grid search and cross validation is super time costing, since we are dealing with a multi-dimensional data. So we decide only tune the mtry parameter which we consider as the most important parameter that affect the model performance.

Thus, after tuning the mtry parameter, we get the optimal parameter as mtry = 77. We then plug in this optimal parameter into our training process.

```
## Fit the train model and record the running time
source("../lib/train_rf.R")
tm_train_rf <- NA
tm_train_rf <- system.time(fit_train_rf <- train_rf(dat_train))
save(fit_train_rf, file="../output/rf_train.RData")
```

Calculate the training error of RF model

```
### load models built
source("../lib/test_rf.R")     ### test model

pred_train_rf <- test_rf(fit_train_rf, dat_train)
accu_train_rf <- mean(pred_train_rf == dat_train$emotion_idx)
cat("The trainig accuracy of model: RF", "is", accu_train_rf*100, "%.\n")
```

```
## The trainig accuracy of model: RF is 100 %.
```

The training process is already encapsulated in the *train_rf.R* file and the training time is recorded as tm_train_rf.

```
tm_test_rf <- NA
if(run.test){
  load(file="../output/rf_train.RData")
  tm_test_rf <- system.time(pred_rf <- test_rf(fit_train_rf, dat_test = dat_test))
}
```

Evaluation on RF model

```
accu_rf <- mean(dat_test$emotion_idx == pred_rf)
cat("The accuracy of model: RF", "is", accu_rf*100, "%.\n")
```

```
## The accuracy of model: RF is 42.8 %.
```

```
confusionMatrix(pred_rf, 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 22
##         1  14  0  2  2  0  0  1  0  0  2  0  0  0  0  0  4  0  1  0  0  0  1
##         2   0 15  0  0  0  1  0  0  9  0  0  0  0  0  0  0  0  0  0  2  0  0
##         3   5  0 15  1  0  1  1  0  0  5  2  2  0  0  1  2  0  0  1  0  0  4
##         4   1  0  3 11  0  1  0  0  0  1  0  4  6  0  0  0  0  0  0  0  1  0
##         5   0  0  0  0 21  0  1  0  0  0  0  0  0  0  4  0  0  2  0  0  0  0
##         6   0  0  2  1  0  5  0  0  0  0  3  0  1  0  0  0  0  0  0  0  0  0
##         7   0  0  0  0  1  0  8  0  0  0  0  0  0  1  1  0  0  0  0  1  0  0
##         8   0  0  0  0  0  0  4 22  3  0  0  0  0  0  0  0  5  0  0  3  0  0
##         9   0  5  0  0  0  1  0  1 11  0  0  0  0  0  0  0  0  0  0  0  2  1
##         10  0  0  2  4  0  0  0  0  0  9  3  2  0  0  0  1  0  0  0  0  0  1
##         11  0  0  1  1  0  2  0  0  0  0  5  2  1  0  0  0  0  0  0  0  0  1
##         12  0  0  0  3  0  4  0  0  0  1  9  5  4  0  0  2  0  0  0  0  0  1
##         13  1  0  1  4  0  0  1  0  0  5  3  2  3  0  1  1  0  0  0  0  1  0
##         14  0  0  0  0  0  0  1  0  0  0  0  2  2 19  4  0  0  0  2  2  2  1
##         15  0  0  0  0  1  0  1  0  0  0  0  0  0  1  4  0  2  0  1  1  0  0
##         16  1  0  0  0  0  3  1  0  0  0  0  0  0  0  0 14  0  0  0  0  0  1
```

```
##         17  0  0  0  0  2  0  3  1  0  0  0  0  0  1  0  0  8  9  0  4  0  1
##         18  0  0  0  0  2  0  2  0  0  0  0  0  0  0  2  0  3  6  0  2  0  0
##         19  1  0  1  0  0  0  3  0  0  0  0  0  0  0  3  0  1  0  4  3  2  4
##         20  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0  2  0  0
##         21  0  0  0  0  0  2  1  1  2  1  0  0  1  2  0  0  3  0  6  4 12  3
##         22  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  2  1  0  1
##
## Overall Statistics
##
##                Accuracy : 0.428
##                  95% CI : (0.3842, 0.4727)
##     No Information Rate : 0.056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4005
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.6087   0.7500   0.5556   0.4074   0.7778   0.2500
## Specificity            0.9727   0.9750   0.9471   0.9641   0.9852   0.9854
## Pos Pred Value         0.5185   0.5556   0.3750   0.3929   0.7500   0.4167
## Neg Pred Value         0.9810   0.9894   0.9739   0.9661   0.9873   0.9693
## Prevalence             0.0460   0.0400   0.0540   0.0540   0.0540   0.0400
## Detection Rate         0.0280   0.0300   0.0300   0.0220   0.0420   0.0100
## Detection Prevalence   0.0540   0.0540   0.0800   0.0560   0.0560   0.0240
## Balanced Accuracy      0.7907   0.8625   0.7514   0.6857   0.8815   0.6177
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.2857   0.8462   0.4400    0.3750    0.2000    0.2500
## Specificity            0.9915   0.9684   0.9789    0.9727    0.9832    0.9500
## Pos Pred Value         0.6667   0.5946   0.5238    0.4091    0.3846    0.1724
## Neg Pred Value         0.9590   0.9914   0.9708    0.9686    0.9589    0.9682
## Prevalence             0.0560   0.0520   0.0500    0.0480    0.0500    0.0400
## Detection Rate         0.0160   0.0440   0.0220    0.0180    0.0100    0.0100
## Detection Prevalence   0.0240   0.0740   0.0420    0.0440    0.0260    0.0580
## Balanced Accuracy      0.6386   0.9073   0.7095    0.6738    0.5916    0.6000
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity             0.1667    0.7917    0.2000    0.5833    0.3636
## Specificity             0.9585    0.9664    0.9854    0.9874    0.9561
## Pos Pred Value          0.1304    0.5429    0.3636    0.7000    0.2759
## Neg Pred Value          0.9686    0.9892    0.9673    0.9792    0.9703
## Prevalence              0.0360    0.0480    0.0400    0.0480    0.0440
## Detection Rate          0.0060    0.0380    0.0080    0.0280    0.0160
## Detection Prevalence    0.0460    0.0700    0.0220    0.0400    0.0580
## Balanced Accuracy       0.5626    0.8790    0.5927    0.7854    0.6599
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity             0.3158    0.2500    0.0800    0.6000    0.0500
## Specificity             0.9771    0.9628    0.9958    0.9458    0.9917
## Pos Pred Value          0.3529    0.1818    0.5000    0.3158    0.2000
## Neg Pred Value          0.9731    0.9749    0.9536    0.9827    0.9616
## Prevalence              0.0380    0.0320    0.0500    0.0400    0.0400
## Detection Rate          0.0120    0.0080    0.0040    0.0240    0.0020
```

```
## Detection Prevalence    0.0340    0.0440    0.0080    0.0760    0.0100
## Balanced Accuracy       0.6465    0.6064    0.5379    0.7729    0.5208
```

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

```
## Time for training model RF =  435.702 s
```

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

```
## Time for testing model RF =  0.603 s
```

### Step 5.5: Extreme Gradient Boosting (XGBoost)

XGBoost stands for "Extreme Gradient Boosting", where the term "Gradient Boosting" originates from the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman.

Because the constructing XGBoost models take quite a long time, we used random search to tune the parameters. With the result of cross validation on the training set, the hyperparameters we used for XGBoost model are listed as followed.

- booster = "gblinear" – the booster to use, can be gbtree or gblinear, gblinear has better performance and efficiency
- objective = "multi:softmax" – set xgboost to do multiclass classification using the softmax objective, the prediction outputs the probability of each class
- eval_metric = "mlogloss" – evaluation metrics for validation data
- lambda = 1.46 – L2 regularization term on weights
- lambda_bias = 0.234 – L2 regularization term on bias
- alpha = 0.0198 – L1 regularization term on weights

```
### load models built
source("../lib/train_xgb.R")    ### train model

### use the optimal parameters
xgb_param_list <- list(
  xgb_para = list(alpha = 0.0198, lambda = 1.46, lambda_bias = 0.234),
  nround = 100
)

### fit train data

tm_train_xgb = NA
tm_train_xgb <- system.time(fit_train_xgb <- train_xgb(feature_df = dat_train, par = xgb_param_list))
save(fit_train_xgb, file="../output/xgb_train.RData")
```

Please see the details of model as below.

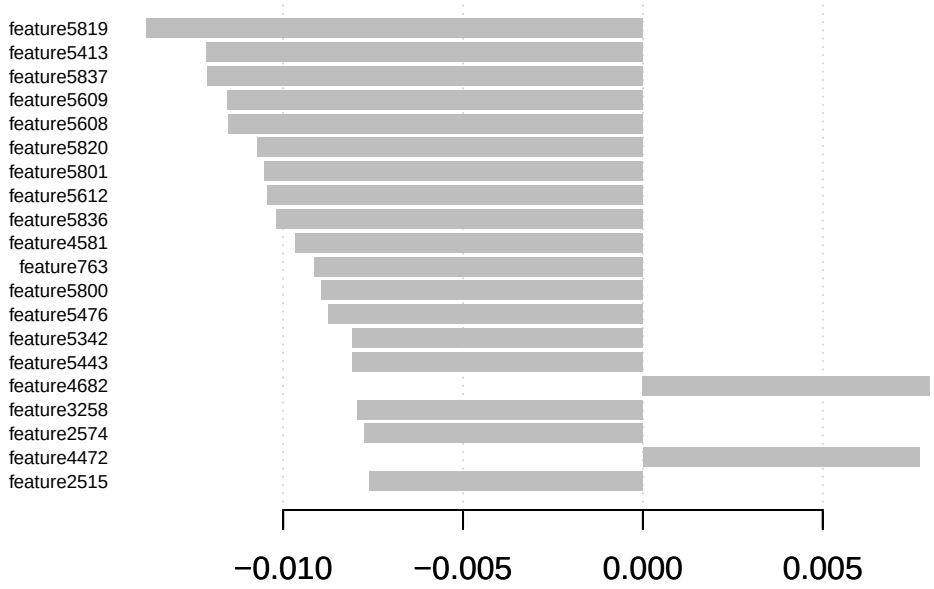
```
fit_train_xgb
```

```
## ##### xgb.Booster
## raw: 516.7 Kb
## call:
##   xgb.train(params = par$xgb_para, data = dtrain, nrounds = par$nround,
##     nthread = 6)
## params (as set within xgb.train):
##   alpha = "0.0198", lambda = "1.46", lambda_bias = "0.234", booster = "gblinear", objective = "multi
## xgb.attributes:
##   niter
## callbacks:
```

```
##   cb.print.evaluation(period = print_every_n)
## # of features: 6006
## niter: 100
## nfeatures : 6006
```

After running our baseline model, we wanted to visualize the contribution of each feature.

```
xgb_importance_mat <- xgb.importance(
  feature_names = colnames(dat_train[,-which(names(dat_train) == 'emotion_idx')]),
  model = fit_train_xgb)
xgb.plot.importance (importance_matrix = xgb_importance_mat[1:20])
```




Calculate the training error of XGB model

```
### load models built
source("../lib/test_xgb.R")     ### test model

pred_train_xgb <- test_xgb(fit_train_xgb, dat_train)
accu_train_xgb <- mean(pred_train_xgb$max_prob == dat_train$emotion_idx)
cat("The trainig accuracy of model: XGB", "is", accu_train_xgb*100, "%.\n")
```

```
## The trainig accuracy of model: XGB is 75.75 %.
```

Then, we predicted the test data and evaluated the performance of model.

```
### predict test data
tm_test_xgb=NA
if(run.test){
  load(file="../output/xgb_train.RData")
  tm_test_xgb <- system.time(pred_xgb <- test_xgb(xgb_model = fit_train_xgb, dat_test = dat_test))
}
```

Evaluation on XGB model

```
accu_xgb <- mean(pred_xgb$max_prob == dat_test$emotion_idx)
cat("The accuracy of model: XGB", "is", accu_xgb*100, "%.\n")
```

```
## The accuracy of model: XGB is 53.2 %.
```

```r
confusionMatrix(factor(pred_xgb$max_prob),factor(dat_test$emotion_idx),mode = "everything")
```

```
## 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  15  0  3  1  0  0  0  0  0  2  0  1  0  0  0  2  0  1  0  0  0  0
##         2   0 14  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3   4  0 16  1  0  1  0  0  0  5  0  0  0  0  0  0  0  0  1  0  0  1
##         4   1  0  2 17  0  0  1  0  0  2  2  1  2  0  0  0  0  0  0  0  0  1
##         5   0  0  0  0 19  0  0  0  0  0  0  0  0  0  4  0  0  2  0  0  0  0
##         6   0  0  0  1  0  9  0  0  1  0  0  1  2  1  0  0  0  0  0  0  0  1
##         7   0  0  0  0  2  0 14  0  1  0  0  0  0  0  0  1  1  0  0  0  0  0
##         8   0  2  0  0  1  0  3 23  1  0  0  0  0  0  0  0  2  0  0  2  0  0
##         9   0  3  0  0  0  0  0  1 18  0  0  0  0  0  0  0  0  0  0  0  1  1
##         10  1  0  2  3  0  1  0  0  0 11  2  2  2  0  0  0  0  0  0  0  1  1
##         11  0  0  0  0  0  1  0  0  0  1 11  2  1  1  0  0  0  0  0  0  0  1
##         12  0  0  2  0  0  2  0  0  0  1  7  7  3  0  0  2  0  0  0  0  0  0
##         13  1  0  0  2  0  0  0  0  0  1  3  3  5  0  0  0  0  0  0  0  1  1
##         14  0  0  0  0  0  0  1  0  0  0  0  1  1 17  4  0  0  0  2  0  0  0
##         15  0  0  0  1  1  0  1  0  0  0  0  0  0  1  8  0  0  0  1  0  1  0
##         16  0  0  0  0  1  2  0  0  0  0  0  1  0  0  0 15  1  0  0  3  0  1
##         17  0  0  0  0  1  0  0  1  0  0  0  0  0  1  0  0  8  6  0  4  0  0
##         18  1  0  1  0  2  0  4  0  0  0  0  0  0  0  0  0  1  5  8  0  1  0  0
##         19  0  0  0  0  0  1  2  0  1  0  0  0  1  1  3  0  0  2  6  4  2  3
##         20  0  0  0  0  0  1  1  1  2  0  0  0  0  0  0  0  1  0  0  9  1  2
##         21  0  1  0  0  0  2  1  0  0  0  0  0  1  2  1  0  4  0  4  2 12  3
##         22  0  0  1  1  0  0  0  0  0  1  0  1  0  0  0  3  0  0  2  0  1  4
##
## Overall Statistics
##
##                Accuracy : 0.532
##                  95% CI : (0.4872, 0.5764)
##     No Information Rate : 0.056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5094
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.6522   0.7000   0.5926   0.6296   0.7037   0.4500
## Specificity            0.9790   0.9979   0.9725   0.9746   0.9873   0.9854
## Pos Pred Value         0.6000   0.9333   0.5517   0.5862   0.7600   0.5625
## Neg Pred Value         0.9832   0.9876   0.9766   0.9788   0.9832   0.9773
## Precision              0.6000   0.9333   0.5517   0.5862   0.7600   0.5625
## Recall                 0.6522   0.7000   0.5926   0.6296   0.7037   0.4500
## F1                     0.6250   0.8000   0.5714   0.6071   0.7308   0.5000
## Prevalence             0.0460   0.0400   0.0540   0.0540   0.0540   0.0400
## Detection Rate         0.0300   0.0280   0.0320   0.0340   0.0380   0.0180
## Detection Prevalence   0.0500   0.0300   0.0580   0.0580   0.0500   0.0320
## Balanced Accuracy      0.8156   0.8490   0.7826   0.8021   0.8455   0.7177
```

```
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.5000   0.8846   0.7200    0.4583    0.4400    0.3500
## Specificity            0.9894   0.9768   0.9874    0.9685    0.9853    0.9646
## Pos Pred Value         0.7368   0.6765   0.7500    0.4231    0.6111    0.2917
## Neg Pred Value         0.9709   0.9936   0.9853    0.9726    0.9710    0.9727
## Precision              0.7368   0.6765   0.7500    0.4231    0.6111    0.2917
## Recall                 0.5000   0.8846   0.7200    0.4583    0.4400    0.3500
## F1                     0.5957   0.7667   0.7347    0.4400    0.5116    0.3182
## Prevalence             0.0560   0.0520   0.0500    0.0480    0.0500    0.0400
## Detection Rate         0.0280   0.0460   0.0360    0.0220    0.0220    0.0140
## Detection Prevalence   0.0380   0.0680   0.0480    0.0520    0.0360    0.0480
## Balanced Accuracy      0.7447   0.9307   0.8537    0.7134    0.7126    0.6573
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity             0.2778    0.7083    0.4000    0.6250    0.3636
## Specificity             0.9751    0.9811    0.9875    0.9811    0.9728
## Pos Pred Value          0.2941    0.6538    0.5714    0.6250    0.3810
## Neg Pred Value          0.9731    0.9852    0.9753    0.9811    0.9708
## Precision               0.2941    0.6538    0.5714    0.6250    0.3810
## Recall                  0.2778    0.7083    0.4000    0.6250    0.3636
## F1                      0.2857    0.6800    0.4706    0.6250    0.3721
## Prevalence              0.0360    0.0480    0.0400    0.0480    0.0440
## Detection Rate          0.0100    0.0340    0.0160    0.0300    0.0160
## Detection Prevalence    0.0340    0.0520    0.0280    0.0480    0.0420
## Balanced Accuracy       0.6264    0.8447    0.6938    0.8030    0.6682
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity             0.4211    0.3750    0.3600    0.6000    0.2000
## Specificity             0.9688    0.9587    0.9811    0.9563    0.9792
## Pos Pred Value          0.3478    0.2308    0.5000    0.3636    0.2857
## Neg Pred Value          0.9769    0.9789    0.9668    0.9829    0.9671
## Precision               0.3478    0.2308    0.5000    0.3636    0.2857
## Recall                  0.4211    0.3750    0.3600    0.6000    0.2000
## F1                      0.3810    0.2857    0.4186    0.4528    0.2353
## Prevalence              0.0380    0.0320    0.0500    0.0400    0.0400
## Detection Rate          0.0160    0.0120    0.0180    0.0240    0.0080
## Detection Prevalence    0.0460    0.0520    0.0360    0.0660    0.0280
## Balanced Accuracy       0.6949    0.6668    0.6705    0.7781    0.5896
```

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

```
## Time for training model XGB =  539.534 s
```

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

```
## Time for testing model XGB =  0.587 s
```

Overall, the predictive accuracy of XGBoost is beyond 50%. Also, the accuracy of test set is close to the accuracy of the cross validation on training set. And with the linear booster, the model performs better and more efficient than tree booster. However, it is computational expensive to tune multiple parameters based on cross validation. The result of model is less interpretable and easily understood as we built models on fiducial points distances.

### Step 5.6: Ensemble Extreme Gradient Boosting (XGBoost) and Support Vector Machine (SVM)

Among these improved models, XGBoost and SVM (With some seed, SVM has the accuracy above 50%) have better performance than others. So we think maybe build a simple ensemble model based on XGBoost result

and SVM result, calculating a weighted average of the probability matrix then deciding the final prediction, will increase the accuracy. We conducted the experiments of different weights on the training set to see the performance of the simple ensemble by cross validation.

```
source("../lib/simple_ensemble.R")
mat1 <- pred_xgb[,-23]
mat2 <- pred_svm %>% attr("probabilities") %>% as.data.frame()
mat2 <- mat2[, order(as.integer(colnames(mat2)))]
colnames(mat2) = colnames(mat1)

### predict test data
tm_test_ensemble <- system.time(pred_ensemble <- simple_ensemble(mat1,mat2,0.51))
```

```
accu_ensemble <- mean(pred_ensemble$max_prob == dat_test$emotion_idx)
cat("The accuracy of ensemble model: ", "is", accu_ensemble*100, "%.\n")
```

```
## The accuracy of ensemble model:  is 54.2 %.
```

```
confusionMatrix(factor(pred_ensemble$max_prob),factor(dat_test$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 22
##         1  17  0  3  1  0  0  0  0  0  2  0  1  0  0  0  1  0  0  0  0  0  0
##         2   0 14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
##         3   3  0 15  1  0  0  0  0  0  5  0  0  0  0  0  0  0  0  1  0  0  1
##         4   1  0  2 17  0  0  1  0  0  2  2  1  2  0  0  0  0  0  0  1  0
##         5   0  0  0  0 21  0  1  0  0  0  0  0  0  0  4  0  0  4  0  0  0  0
##         6   0  0  0  1  0  9  0  0  1  0  0  1  2  1  0  0  0  0  0  0  0  1
##         7   0  0  0  0  2  0 15  1  1  0  0  0  0  0  0  1  1  0  0  0  0  0
##         8   0  1  0  0  0  0  2 23  1  0  0  0  0  0  0  0  1  0  0  1  0  0
##         9   0  4  0  0  0  0  0  1 20  0  0  0  0  0  0  0  0  0  0  0  1  1
##         10  1  0  2  3  0  2  0  0  0 11  2  2  2  0  0  0  0  0  0  0  1  2
##         11  0  0  0  1  0  0  0  0  0  2 11  2  1  1  0  1  0  0  0  0  0  1
##         12  0  0  2  0  0  4  0  0  0  1  7  7  3  0  0  1  0  0  0  0  0  0
##         13  0  0  0  1  0  0  0  0  0  0  3  2  4  0  0  0  0  0  0  0  1  2
##         14  0  0  0  0  0  0  1  0  0  0  0  1  2 17  5  0  0  0  2  0  0  0
##         15  0  0  0  0  0  0  1  0  0  0  0  0  0  2  7  0  0  0  1  0  1  0
##         16  0  0  0  1  1  2  0  0  0  0  0  1  0  0  0 16  1  0  0  3  0  0
##         17  0  0  0  0  1  0  0  0  0  0  0  0  0  1  0  0 11  5  0  3  0  0
##         18  1  0  1  0  2  0  2  0  0  0  0  0  0  0  0  1  3  8  0  3  0  0
##         19  0  0  0  0  0  0  2  0  1  0  0  0  0  0  3  0  0  2  6  4  2  4
##         20  0  0  0  0  0  1  2  1  1  0  0  0  0  1  0  0  1  0  0  8  2  2
##         21  0  1  0  0  0  2  1  0  0  0  0  0  1  1  1  0  4  0  4  2 11  3
##         22  0  0  2  1  0  0  0  0  0  1  0  2  1  0  0  3  0  0  2  0  0  3
##
## Overall Statistics
##
##                Accuracy : 0.542
##                  95% CI : (0.4972, 0.5863)
##     No Information Rate : 0.056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5198
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity            0.7391   0.7000   0.5556   0.6296   0.7778   0.4500
## Specificity            0.9832   0.9979   0.9767   0.9746   0.9810   0.9854
## Pos Pred Value         0.6800   0.9333   0.5769   0.5862   0.7000   0.5625
## Neg Pred Value         0.9874   0.9876   0.9747   0.9788   0.9872   0.9773
## Prevalence             0.0460   0.0400   0.0540   0.0540   0.0540   0.0400
## Detection Rate         0.0340   0.0280   0.0300   0.0340   0.0420   0.0180
## Detection Prevalence   0.0500   0.0300   0.0520   0.0580   0.0600   0.0320
## Balanced Accuracy      0.8612   0.8490   0.7661   0.8021   0.8794   0.7177
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity            0.5357   0.8846   0.8000    0.4583    0.4400    0.3500
## Specificity            0.9873   0.9873   0.9853    0.9643    0.9811    0.9625
## Pos Pred Value         0.7143   0.7931   0.7407    0.3929    0.5500    0.2800
## Neg Pred Value         0.9729   0.9936   0.9894    0.9725    0.9708    0.9726
## Prevalence             0.0560   0.0520   0.0500    0.0480    0.0500    0.0400
## Detection Rate         0.0300   0.0460   0.0400    0.0220    0.0220    0.0140
## Detection Prevalence   0.0420   0.0580   0.0540    0.0560    0.0400    0.0500
## Balanced Accuracy      0.7615   0.9360   0.8926    0.7113    0.7105    0.6562
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity             0.2222    0.7083    0.3500    0.6667    0.5000
## Specificity             0.9813    0.9769    0.9896    0.9811    0.9791
## Pos Pred Value          0.3077    0.6071    0.5833    0.6400    0.5238
## Neg Pred Value          0.9713    0.9852    0.9734    0.9832    0.9770
## Prevalence              0.0360    0.0480    0.0400    0.0480    0.0440
## Detection Rate          0.0080    0.0340    0.0140    0.0320    0.0220
## Detection Prevalence    0.0260    0.0560    0.0240    0.0500    0.0420
## Balanced Accuracy       0.6018    0.8426    0.6698    0.8239    0.7395
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity             0.4211    0.3750    0.3200    0.5500    0.1500
## Specificity             0.9730    0.9628    0.9768    0.9583    0.9750
## Pos Pred Value          0.3810    0.2500    0.4211    0.3548    0.2000
## Neg Pred Value          0.9770    0.9790    0.9647    0.9808    0.9649
## Prevalence              0.0380    0.0320    0.0500    0.0400    0.0400
## Detection Rate          0.0160    0.0120    0.0160    0.0220    0.0060
## Detection Prevalence    0.0420    0.0480    0.0380    0.0620    0.0300
## Balanced Accuracy       0.6970    0.6689    0.6484    0.7542    0.5625
```

```
cat("Time for training ensemble model = ", tm_train_xgb[1] + tm_train_svm[1], "s \n")
```

```
## Time for training ensemble model =  743.748 s
```

```
cat("Time for testing ensemble model = ",tm_test_xgb[1] + tm_test_svm[1] + tm_test_ensemble[1], "s \n")
```

```
## Time for testing ensemble model =  10.718 s
```

With the experiments, we found that the weighted average can improve the accuracy by about 1.3% averagely and 2.7% most. Although the accuracy of ensemble model didn't increase that much, the ensemble model balance the error and give us a more stable model.

### Step 5.7: Neutral Network (NN)

We developed NN model by Python, so please see another pdf file in doc folder on Github. Plus, the data

used for NN model is dat_train.csv and data_test.csv processed from this R markdown.

**Step 6: Comparison and Summary**

From the data above, we can choose NN model as our advanced model and there are some advantages of this model:

- High efficiency in both training and testing process;
- Higher prediction accuracy compared to baseline model;
- Robustness.