Can you recognize the emotion from an image of a face?

Jiancong Shen, Vikki Sui, Jinxu Xiang, Ruiqi Xie, Wenjie Xie

Abstract

This document mainly describes an improved way of emotion recognition algorithm. This method reduces the calculation time and improves the classification accuracy by pre-transforming data, extracting efficient features (base on points and images), and improving the algorithm. The goal is to select the best model by comparing the classification results of different classifiers under different features.

Computer configuration

System: Microsoft Windows 10 x64

CPU: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz(3600 MHz)

GPU: NVIDIA GeForce RTX 2070 SUPER (8192 MB)

RAM: 16.00 GB (2400 MHz)

Final result

	test accuracy	model training time	all training time	test prediction time	all prediction time
gbm_baseline	44.7%	1960s	1961s	13.3s	13.4s
gbm_improved	47.7%	42.9s	67.2s	0.03s	5.92s
gbm_colored	48.8%	45.2s	188s	0.04s	38.6s
svm_improved	57.1%	2.39s	26.7s	0.20s	6.09s
svm_colored	60.5%	2.38s	146s	0.20s	38.8s
xgb_improved	52.9%	45.2s	69.5s	0.06s	5.95s
xgb_colored	56.3%	31.7s	175s	0.06s	38.6s

Based on the fact that the total training time is the sum of model training time, the feature extraction time, and the data processing time, we have the formula below:

All training time = Training data preprocessing time + Feature extraction time + Model training time

All test time = Testing data preprocessing time + Feature extraction time + Test prediction time

The accuracy and time shown in the table above are from a certain measurement and may not exactly match the results shown in the PDF document.

The improved feature contains 129 features extracted from 78 points' position. The color feature contains all improved feature and 10 other features which are extracted from points and images. Therefore, the total dimension of color feature is 139.

We extract color feature and use the svm classifier as our improved model. Since there is no other results affect our judgment of in-class tests. We predict the final accuracy of in-class test to be 58%.

Difference between Rmd and Html

Since the baseline model is large, we can't upload it to github, so we set the run.baseline to false after knit. If you need to run Rmd, the result will be slightly different, it will not contain baseline model, but the rest part is same as Html file. If you want to see the result of baseline, you have to set run.baseline and run.train.gbm to true, it will take more than half an hour to calculate. Moreover, before running Rmd, you should change setwd() below to your path.

```r
if(!require("EBImage")){
  source("https://bioconductor.org/biocLite.R")
  biocLite("EBImage")
}
if(!require("R.matlab")){
  install.packages("R.matlab")
}
if(!require("readxl")){
  install.packages("readxl")
}
if(!require("dplyr")){
  install.packages("dplyr")
}
if(!require("readxl")){
  install.packages("readxl")
}
if(!require("ggplot2")){
  install.packages("ggplot2")
}
if(!require("caret")){
  install.packages("caret")
}
```

```
## Warning: package 'lattice' was built under R version 3.6.3
```

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

```
## Warning: package 'gbm' was built under R version 3.6.3
```

```r
if(!require("e1071")){
  install.packages("e1071")
}
if(!require("geometry")){
  install.packages("geometry")
}
```

```
## Warning: package 'geometry' was built under R version 3.6.3
```

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

```
## Warning: package 'mlogit' was built under R version 3.6.3
```

```
## Warning: package 'lmtest' was built under R version 3.6.3
```

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

```
## Warning: package 'forcats' was built under R version 3.6.3
```

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

```
## Warning: package 'xgboost' was built under R version 3.6.3
```

```r
library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
library(gbm)
library(e1071)
library(geometry)
library(mlogit)
library(tidyverse)
library(xgboost)
```

Step 0 set work directories

```r
set.seed(1)
setwd("C:/course/5243 Applied Data Science/Project/Spring2020-Project3-group8/doc")
# here replace it with your own path or manually set it in RStudio to where this rmd file is located.
# use relative path for reproducibility
```

Provide directories for training images. Training images and training fiducial points will be in different subfolders.

```r
train_dir <- "../data/train_set/" # This will be modified for different data sets.
train_image_dir <- paste(train_dir, "images/", sep="")
train_pt_dir <- paste(train_dir,  "points/", sep="")
train_label_path <- paste(train_dir, "label.csv", sep="")
```

Step 1: set up controls for evaluation experiments.

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) process for baseline
- (T/F) process for data set
- (T/F) cross-validation on the training set
- (number) K, the number of CV folds
- (T/F) process features for training and test set
- (T/F) process model of training data

```r
run.baseline = TRUE # run baseline feature, training slow!

run.dataprocess.fiducial_pt_list = FALSE # save fiducial_pt_list
run.dataprocess.all_points = FALSE # save all_points, new data after rotate zoom and move
run.dataprocess.ave_points = FALSE # save ave_points, average data of 22 emotions
run.dataprocess.diff_points = FALSE # save diff, distance of each point to its mean

run.cv = FALSE # run cross-validation on the training set
K = 5  # number of CV folds
```

```r
run.feature.train = FALSE # process features for training set
run.feature.test = FALSE # process features for test set

run.train.gbm = FALSE # run gbm training model
run.train.svm = FALSE # run svm training model
run.train.xgb = FALSE # run xgb training model
```

The cross-validation method takes too much time, and the final model we choose does not need it. Although it works well, we will not run cross-validation process in this document.

Step 2: import data and train-test split

This time we did not use a random 20% photos as the test set, but instead randomized all the pictures of 20% of the people as the test set. Because when we deal with the problem of facial expression recognition, there is a high probability of encountering strangers. So the expectation of this selection method is closer to the real classification accuracy.

However, we only have 2500 pictures and 230 people, which is not enough for accurate training. When we randomly select 20% of the people, it is likely that there is too little training data for a certain expression in the training set. This may increase the variance of classification accuracy.

This effect is limited to the tests in this document. The tests we need to do will use all 2,500 training sets, the expectations and variances mentioned above have nothing to do with it.

```r
#train-test split
info <- read.csv(train_label_path)
n <- length(info$identity %>% unique())

#take 80% of the observations as the train set
n_train <- round(n*(4/5), 0)
train_identity = sample(info$identity %>% unique(), n_train, replace = F)
train_idx <- which(info$identity %in% train_identity)
test_idx <- setdiff(info$Index,train_idx)
train_idx <- sample(train_idx, length(train_idx), replace = F)
```

Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

```r
#function to read fiducial points
#input: index
#output: matrix of fiducial points corresponding to the index
if(run.dataprocess.fiducial_pt_list){

  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")
}
```

```r
#function to rotate, translate, zoom the fiducial points to standardize all the points
#take the middle height of two eyes and the height of nose center to standardize this distance to be 17

#input: indices, train_dir, fiducial_pt_list, single, double
#output: the fiducial points after all the trasformation
```

```
if(run.dataprocess.all_points){

  source("../lib/change_images.R")
  load("../output/fiducial_pt_list.RData")

  single = c(35,36,37,38,44,52,56,59,62)  #points without a match, i.e, points on the axis of symmetry
  double = data.frame(
    x1 = c(1,2,3,4,5,6,7,8,9,19,20,21,22,23,24,25,26,39,40,41,42,43,50,51,57,58,63),
    x2 = c(10,15,14,13,12,11,18,17,16,31,30,29,28,27,34,33,32,49,48,47,46,45,54,53,55,60,61))
  #points that matches each other on both sides of the face

  indices = 1:2500

  tm.run.all_points = system.time(all_points <- change_points(indices, train_dir, fiducial_pt_list, sin

  all_points = map(all_points, as.matrix)

  save(all_points, file = "../output/all_points.RData")
  save(tm.run.all_points, file = "../output/tm.run.all_points.RData")
}
```

```
#data manipulation to find the average points of each emotion, in order to find more detailed differenc
#input: all_points
#output: ave_points
if(run.dataprocess.ave_points){

  load("../output/all_points.RData")

  emo_idx = map(1:22, ~info$emotion_idx == .x)
  group_points = map(emo_idx, ~all_points[.x])
  ave_points = NULL
  for (i in 1:length(group_points)){
    mean = matrix(rep(0,78*2), nc = 2)
    l = length(group_points[[i]])
    for(j in 1:l){
      group_points[[i]][[j]]
      mean = mean + group_points[[i]][[j]]/l
    }
    ave_points = c(ave_points, list(mean))
  }
  save(ave_points, file="../output/ave_points.Rdata")
}
```

```
if(run.dataprocess.diff_points){

  load("../output/all_points.RData")
  load("../output/ave_points.RData")

  #functions to find the difference on x-aixs and y-axis of each point to corresponding points of the 2
  #input: emo_index, all_points
  #output: diff
  get_diff <- function(emo_index, train= all_points){
    group <- info %>% filter(emotion_idx == emo_index)
    idx <- group$Index
    points <- train[idx]
    dfmean <- ave_points[[emo_index]]
    diff <- map(points, function(x){x-dfmean})
    return (diff)
  }
  diff_points <- list()

  for (i in 1:22){
    diff_points <- c(diff_points, get_diff(i, all_points))
  }

  #find the the distance of each point to the corresponding points of the 22 average points of differen
  #output: distance
  distance <- map(diff_points, function(x){x[,1]^2+x[,2]^2})
  distance = map(1:78, function(y) map(distance, ~.x[y]) %>% unlist)

  save(diff_points, file="../output/diff_points.Rdata")
  save(distance, file="../output/distance.Rdata")
}
load("../output/diff_points.Rdata")
load("../output/distance.Rdata")
```

Step 3: construct features and responses

- feature_baseline: function that represent the original features given by the distance of 78 points between each other
- feature: function that takes the updated feature that were selected by ourselves, which has 129 dimensions if we exclude the color feature and 139 dimension if we include the color feature
  - feature function has a boolean input which indicate if we want to take color features or not
  - color feature includes the wrinkle between eyes and the nasolabial folds
  - with the color feature, the running time will be longer but the predicting accuracy will increase
  - without the color feature, the running time will be shorter but the predicting acuuracy will decrease
- we will also the features into the output folder

```
source("../lib/feature_baseline.R")
source("../lib/feature.R")
load("../output/all_points.RData")

#get the time required for taking the features of train set and test set using the original feature fun
if(run.baseline){
  tm_feature_train_baseline <- NA
  if(run.feature.train){
    tm_feature_train_baseline <- system.time(dat_train_baseline <- feature_baseline(fiducial_pt_list, t
    save(dat_train_baseline, file="../output/feature_train_baseline.RData")
    save(tm_feature_train_baseline, file="../output/tm_feature_train_baseline.RData")
  }

  tm_feature_test_baseline <- NA
  if(run.feature.test){
    tm_feature_test_baseline <- system.time(dat_test_baseline <- feature_baseline(fiducial_pt_list, tes
    save(dat_test_baseline, file="../output/feature_test_baseline.RData")
    save(tm_feature_test_baseline, file="../output/tm_feature_test_baseline.RData")
  }
}

#time required for taking the features of train set using the updated feature function with and without
tm_feature_train <- NA
tm_feature_train_color <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(fiducial_pt_list, train_idx, image_file = "../da
  save(dat_train, file="../output/feature_train.RData")
  save(tm_feature_train, file="../output/tm_feature_train.RData")

  tm_feature_train_color <- system.time(dat_train_color <- feature(fiducial_pt_list, train_idx, image_f
  save(dat_train_color, file="../output/feature_train_color.RData")
  save(tm_feature_train_color, file="../output/tm_feature_train_color.RData")
}

#time required for taking the features of test set using the updated feature function with and without
tm_feature_test <- NA
tm_feature_test_color <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature(fiducial_pt_list, test_idx, image_file = "../data/
  save(dat_test, file="../output/feature_test.RData")
  save(tm_feature_test, file="../output/tm_feature_test.RData")

  tm_feature_test_color <- system.time(dat_test_color <- feature(fiducial_pt_list, test_idx, image_file
  save(dat_test_color, file="../output/feature_test_color.RData")
  save(tm_feature_test_color, file="../output/tm_feature_test_color.RData")
}
```

Step 4: Train a classification model with training features and responses

In this step, we will train the model using GBM, SVM, and XGB, respectively using baseline features, improved features and improved features with color features to get the training model, then save these model to the output folder.

1.Baseline model with dist feature in GBM

```
#use the baseline features to train the gbm model, get the model called gbm_model_baseline and the time
#save the output model into the output
if(run.baseline){
  load("../output/feature_train_baseline.RData")
  if(run.train.gbm){
    source("../lib/train_gbm.R")
    gbm_result_train_baseline = gbm_train(dat_train_baseline, n.trees = 200, bag.fraction = 0.8, shrink
    gbm_model_baseline = gbm_result_train_baseline[[1]]
    tm.gbm.train_baseline = gbm_result_train_baseline[[2]]
    save(gbm_model_baseline, file="../output/gbm_model_baseline.RData")
    save(tm.gbm.train_baseline, file="../output/tm.gbm.train_baseline.RData")
  }
}
```

2.Improved feature in GBM

```
#use the improved features to train the gbm model, get the model called gbm_model and the time used cal
#save the output model into the output
if(run.train.gbm){
  source("../lib/train_gbm.R")
  load("../output/feature_train.RData")

  gbm_result_train = gbm_train(dat_train, n.trees = 200, bag.fraction = 0.8, shrinkage = 0.1, cv.folds
  gbm_model = gbm_result_train[[1]]
  tm.gbm.train = gbm_result_train[[2]]
  save(gbm_model, file="../output/gbm_model.RData")
  save(tm.gbm.train, file="../output/tm.gbm.train.RData")
}
```

3.Improved feature in SVM

```
#use the improved features to train the svm model, get the model called svm_model and the timeused call
#save the output model into the output
if(run.train.svm){
  source("../lib/train_SVM.R")
  load("../output/feature_train.RData")

  svm_result_train = svm_train(dat_train, kernel = 'poly', degree = 1, gamma = 0.008)
  svm_model = svm_result_train[[1]]
  tm.svm.train = svm_result_train[[2]]
  save(svm_model, file="../output/svm_model.RData")
  save(tm.svm.train, file="../output/tm.svm.train.RData")
}
```

4.Improved feature in XGB

```
#use cross validation to find the best parameter for the xgb model
if(run.cv & run.train.xgb){
  source("../lib/xgb_tune.R")
  load("../output/feature_train.RData")

  depth = c(5,10,15)
  child = c(3,5,10)
  xgb_result_cv = xgb_tune(dat_train, depth, child, K)
  xgb_err = xgb_result_cv[[1]]
  tm.xgb.cv = xgb_result_cv[[2]]
  xgb_err_tune = xgb_err[[1]] %>% as.data.frame()
  xgb_best_par = xgb_err[[2]] %>% as.data.frame()
}
```

```
if(run.cv & run.train.xgb){
  colnames(xgb_err_tune) = c(3,5,10)
  xgb_err_tune = gather(xgb_err_tune, key = "min_child")
  xgb_err_tune$depth = rep(c(5,10,15),3)
  xgb_err_tune
  ggplot(xgb_err_tune, mapping = aes(x=min_child, y=depth, fill=value))+
    geom_tile()
}
```

```r
#fit the xgb model with the best parameter if we use cross validation and with default if we do not use
if(run.train.xgb){
  source("../lib/xgb_train.R")
  load("../output/feature_train.RData")

  if(run.cv)
    xgb_result = xgb_train(dat_train, par = best_par_xgb)
  else
    xgb_result = xgb_train(dat_train)
  xgb_model = xgb_result[[1]]
  tm.xgb.train = xgb_result[[2]]
  save(xgb_model, file="../output/xgb_model.RData")
  save(tm.xgb.train, file="../output/tm.xgb.train.RData")
}
```

5.Improved feature with color in GBM

```r
#use the features with colors to train the gbm model called gbm_model_color, and save the model into ou
if(run.train.gbm){
  source("../lib/train_gbm.R")
  load("../output/feature_train_color.RData")

  gbm_result_train_color = gbm_train(dat_train_color, n.trees = 200, bag.fraction = 0.8, shrinkage = 0.
  gbm_model_color = gbm_result_train_color[[1]]
  tm.gbm.train_color = gbm_result_train_color[[2]]
  save(gbm_model_color, file="../output/gbm_model_color.RData")
  save(tm.gbm.train_color, file="../output/tm.gbm.train_color.RData")
}
```

6.Improved feature with color in SVM

```r
#use the features with colors to train the svm model called svm_model_color and save the model into out
if(run.train.svm){
  source("../lib/train_SVM.R")
  load("../output/feature_train_color.RData")

  svm_result_train_color = svm_train(dat_train_color, kernel = 'poly', degree = 1, gamma = 0.008)
  svm_model_color = svm_result_train_color[[1]]
  tm.svm.train_color = svm_result_train_color[[2]]
  save(svm_model_color, file="../output/svm_model_color.RData")
  save(tm.svm.train_color, file="../output/tm.svm.train_color.RData")
}
```

7.Improved feature with color in XGB

```r
#We can choose to use cross validation or not. If yes, we will need to run the two chunks below and get
if(run.cv & run.train.xgb){
  source("../lib/xgb_tune.R")
  load("../output/feature_train_color.RData")

  depth = c(5,10,15)
  child = c(3,5,10)
  xgb_result_cv_color = xgb_tune(dat_train_color, depth, child, K)
  xgb_err_color = xgb_result_cv_color[[1]]
  tm.xgb.cv_color = xgb_result_cv_color[[2]]
  xgb_err_tune_color = xgb_err_color[[1]] %>% as.data.frame()
  xgb_best_par_color = xgb_err_color[[2]] %>% as.data.frame()
}
```

```
if(run.cv & run.train.xgb){
  colnames(xgb_err_tune_color) = c(3,5,10)
  xgb_err_tune_color = gather(xgb_err_tune_color, key = "min_child")
  xgb_err_tune_color$depth = rep(c(5,10,15),3)
  xgb_err_tune_color
  ggplot(xgb_err_tune_color, mapping = aes(x=min_child, y=depth, fill=value))+
    geom_tile()
}
```

```
#use the features with colors to train the xgb model called xgb_model_color and save the model into the
if(run.train.xgb){
  source("../lib/xgb_train.R")
  load("../output/feature_train_color.RData")

  if(run.cv)
    xgb_result_color = xgb_train(dat_train_color, par = best_par_xgb)
  else
    xgb_result_color = xgb_train(dat_train_color)
  xgb_model_color = xgb_result_color[[1]]
  tm.xgb.train_color = xgb_result_color[[2]]
  save(xgb_model_color, file="../output/xgb_model_color.RData")
  save(tm.xgb.train_color, file="../output/tm.xgb.train_color.RData")
}
```

Step 5: Run test on test images

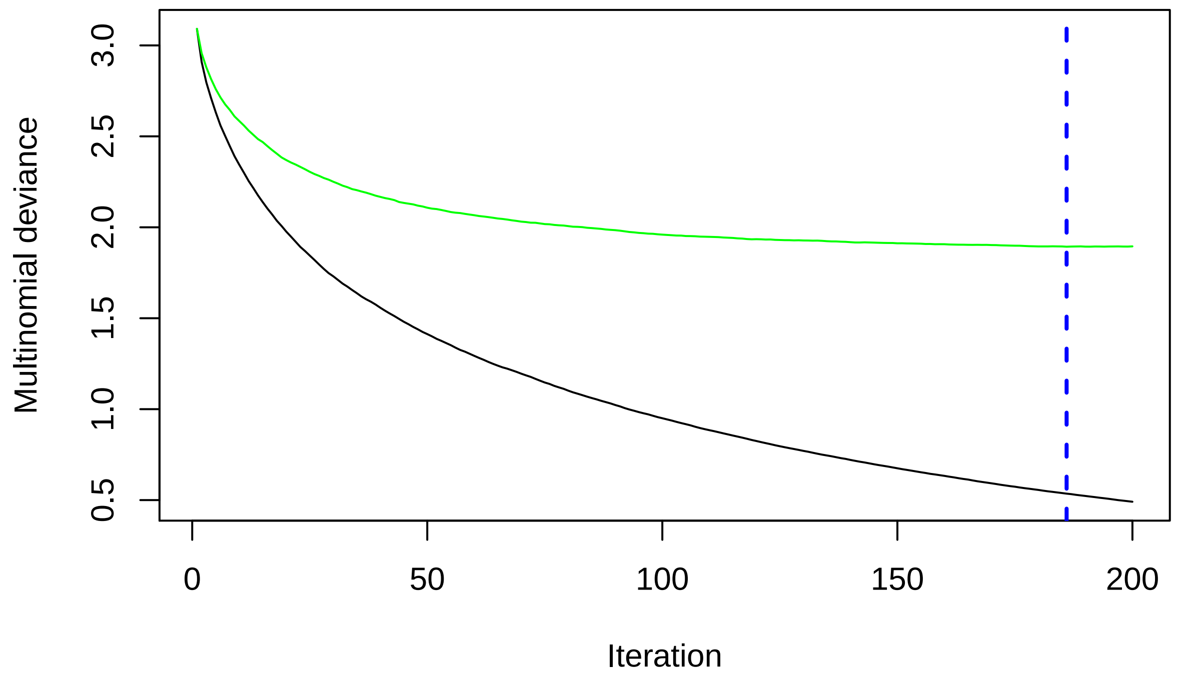
For each model we constructed, we use that model to make prediction on the test set, then we give the confusion matrix of the prediction and the predicting accuracy repectively.

1.Baseline model with dist feature in GBM

Due to the size limit of github. We can't upload 'gbm_model_baseline'. So we put the run.baseline to FALSE. If you want to run baseline. Please run model training of gbm in baseline first.

```
#prediction of the gbm model with only baseline features
if(run.baseline){
  source("../lib/test_gbm.R")
  load("../output/gbm_model_baseline.RData")
  load("../output/feature_test_baseline.RData")

  gbm_result_test_baseline = gbm_test(gbm_model_baseline, dat_test_baseline)
  gbm_pred_baseline = gbm_result_test_baseline[[1]]
  tm.gbm.test_baseline = gbm_result_test_baseline[[2]]
  gbm_pred_class_baseline = apply(gbm_pred_baseline, 1, which.max)
}
```



```r
#get the confusion matrix of the prediction
if(run.baseline){
  confusionMatrix(factor(gbm_pred_class_baseline), dat_test_baseline$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 15  0  2  1  0  0  1  0  0  2  0  0  1  0  0  0  0  1  0  0  0  0
##          2  0 15  0  0  0  0  0  6  5  1  0  0  0  0  0  0  0  0  0  0  1  0
##          3  2  0 13  1  0  0  0  0  0  3  1  0  1  0  1  1  0  0  0  0  0  2
##          4  1  0  1 12  0  2  0  0  0  3  2  0  2  0  0  1  0  0  0  0  0  2
##          5  0  0  1  0 10  0  2  0  0  0  0  0  0  0  1  0  3  0  0  0  0  0
##          6  0  0  1  2  0 14  0  0  0  1  6  2  0  0  0  0  0  0  0  0  0  3
##          7  0  0  0  0  3  1 12  2  0  0  0  0  0  0  0  0  0  0  1  0  3  1
##          8  0  1  0  0  0  0  0 13  2  0  0  0  0  0  0  0  3  0  0  0  0  0
##          9  0  4  1  0  0  0  0  0 12  0  0  0  0  0  0  0  0  0  0  0  0  2
##          10 1  0  1  2  0  0  0  0  0 10  1  1  2  0  0  0  0  0  0  0  0  0
##          11 1  0  0  1  0  0  0  0  0  2  8  3  2  0  0  0  0  0  0  0  0  1
##          12 0  0  0  0  0  0  0  0  0  0  5  8  3  0  0  0  0  0  0  0  1  0
##          13 0  0  0  3  0  1  0  0  0  3  0  2  0  0  0  1  0  0  0  0  1  2
##          14 0  0  0  0  0  1  0  0  0  0  0  1  0 19  2  0  1  0  1  0  3  0
##          15 0  0  0  0  2  1  1  0  0  0  0  0  0  3  2  0  0  0  1  1  0  0
##          16 1  0  0  0  0  1  0  0  0  0  0  0  0  0  0 13  0  0  0  1  0  0
##          17 0  0  0  0  2  0  2  1  0  0  0  0  0  1  1  1  9  6  1  2  2  0
##          18 0  0  0  0  0  0  2  0  0  0  0  0  0  0  1  0  7  8  1  0  0  0
##        19  0  0  0  0  0  1  2  0  0  1  0  0  0  1  5  0  2  1  8  6  4  1
##        20  0  1  0  0  0  0  1  1  1  0  0  0  2  0  1  1  4  0  3  8  3  5
##        21  0  0  0  0  0  0  1  0  0  0  0  1  0  1  1  0  0  0  4  4  6  3
##        22  0  0  3  1  0  1  0  0  0  0  1  3  2  0  0  0  0  0  1  0  2  3
##
## Overall Statistics
##
##                Accuracy : 0.4561
##                  95% CI : (0.4108, 0.5019)
##     No Information Rate : 0.0607
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4296
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.71429  0.71429  0.56522  0.52174  0.58824  0.60870
## Specificity           0.98249  0.97155  0.97363  0.96923  0.98482  0.96703
## Pos Pred Value        0.65217  0.53571  0.52000  0.46154  0.58824  0.48276
## Neg Pred Value        0.98681  0.98667  0.97792  0.97566  0.98482  0.97996
## Prevalence            0.04393  0.04393  0.04812  0.04812  0.03556  0.04812
## Detection Rate        0.03138  0.03138  0.02720  0.02510  0.02092  0.02929
## Detection Prevalence  0.04812  0.05858  0.05230  0.05439  0.03556  0.06067
## Balanced Accuracy     0.84839  0.84292  0.76942  0.74548  0.78653  0.78786
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity           0.50000  0.56522  0.60000   0.38462   0.33333   0.38095
## Specificity           0.97577  0.98681  0.98472   0.98230   0.97797   0.98031
## Pos Pred Value        0.52174  0.68421  0.63158   0.55556   0.44444   0.47059
## Neg Pred Value        0.97363  0.97821  0.98257   0.96522   0.96522   0.97180
## Prevalence            0.05021  0.04812  0.04184   0.05439   0.05021   0.04393
## Detection Rate        0.02510  0.02720  0.02510   0.02092   0.01674   0.01674
## Detection Prevalence  0.04812  0.03975  0.03975   0.03766   0.03766   0.03556
## Balanced Accuracy     0.73789  0.77602  0.79236   0.68346   0.65565   0.68063
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity            0.00000   0.76000  0.133333   0.72222   0.31034
## Specificity            0.97192   0.98013  0.980562   0.99348   0.95768
## Pos Pred Value         0.00000   0.67857  0.181818   0.81250   0.32143
## Neg Pred Value         0.96774   0.98667  0.972163   0.98918   0.95556
## Prevalence             0.03138   0.05230  0.031381   0.03766   0.06067
## Detection Rate         0.00000   0.03975  0.004184   0.02720   0.01883
## Detection Prevalence   0.02720   0.05858  0.023013   0.03347   0.05858
## Balanced Accuracy      0.48596   0.87007  0.556947   0.85785   0.63401
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity            0.50000   0.38095   0.36364   0.23077  0.120000
## Specificity            0.97619   0.94748   0.94956   0.96681  0.969095
## Pos Pred Value         0.42105   0.25000   0.25806   0.28571  0.176471
## Neg Pred Value         0.98257   0.97085   0.96868   0.95624  0.952278
## Prevalence             0.03347   0.04393   0.04603   0.05439  0.052301
## Detection Rate         0.01674   0.01674   0.01674   0.01255  0.006276
## Detection Prevalence   0.03975   0.06695   0.06485   0.04393  0.035565
## Balanced Accuracy      0.73810   0.66422   0.65660   0.59879  0.544547
```

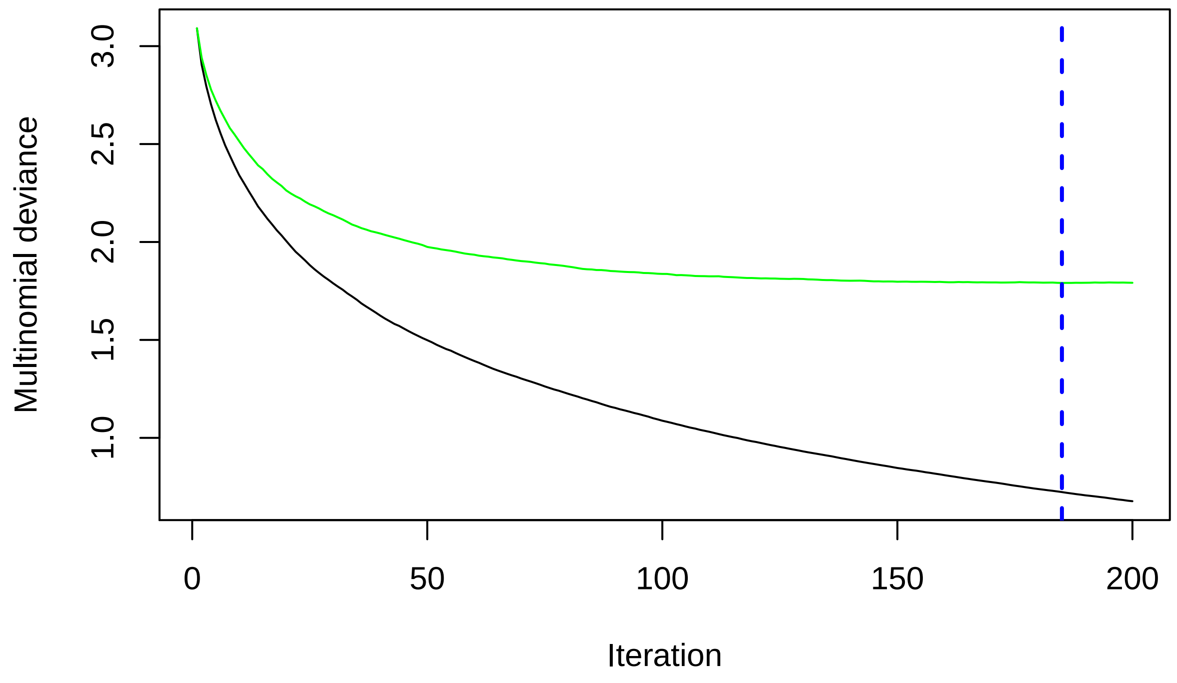
```
#accuracy of the above gbm model
if(run.baseline){
  gbm_accuracy_test_baseline = mean(dat_test_baseline$emotion_idx == gbm_pred_class_baseline)
  print(paste0("The accuracy for gbm model baseline is: ", gbm_accuracy_test_baseline))
}
```

```
## [1] "The accuracy for gbm model baseline is: 0.456066945606695"
```

2.Improved feature in GBM

```
#prediction of the gbm model with improved features
source("../lib/test_gbm.R")
load("../output/gbm_model.RData")
load("../output/feature_test.RData")

gbm_result_test = gbm_test(gbm_model, dat_test)
```



```
gbm_pred = gbm_result_test[[1]]
tm.gbm.test = gbm_result_test[[2]]
gbm_pred_class = apply(gbm_pred, 1, which.max)
```

```
#get the confusion matrix of the prediction
confusionMatrix(factor(gbm_pred_class), 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 13  0  1  2  0  1  1  0  0  1  1  0  0  0  0  0  0  1  0  0  0  1
##         2  0 16  0  0  0  1  0  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3  5  0 16  2  0  0  0  0  0  4  0  1  1  0  0  2  0  0  0  0  0  2
##         4  2  0  0 14  0  1  1  0  0  5  1  0  0  0  0  1  0  0  0  0  0  0
##         5  0  0  0  0 12  0  2  0  0  0  0  0  0  3  3  0  1  2  0  0  1  0
##         6  0  0  1  2  0 12  0  0  0  1  5  4  2  0  1  0  0  0  0  1  1  1
##         7  0  0  0  0  1  0 13  0  1  0  0  0  0  1  1  0  3  0  1  0  1  0
##         8  0  1  0  0  0  0  1 16  2  0  0  0  0  0  0  0  3  1  0  0  1  0
##         9  0  3  0  0  0  0  0  1 10  0  0  0  1  0  0  0  1  0  0  1  1  1
##         10  0  0  0  0  0  0  1  0  0  9  2  0  1  0  1  0  0  0  0  0  0  0
##         11  0  0  0  0  0  2  0  0  0  2  9  1  2  0  0  0  0  0  0  0  0  0
##         12  0  0  0  0  0  1  0  0  0  1  5  8  5  0  0  0  0  0  0  0  1  2
##         13  0  0  1  2  0  0  0  0  0  2  0  1  1  1  0  0  0  0  0  0  0  1
##         14  0  0  0  0  1  1  0  0  0  0  0  1  0 14  1  0  0  0  1  0  3  0
##         15  0  0  0  1  0  0  0  0  0  0  0  0  0  3  7  0  0  0  3  0  0  0
##         16  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11  0  0  0  0  0  3
##         17  0  0  0  0  1  0  2  1  0  0  0  0  0  0  0  1 15  8  1  5  3  0
##         18  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  1  4  2  1  1  0  0
##         19  0  0  2  0  0  2  0  0  0  1  0  0  0  1  1  0  2  1  4  3  0  5
##         20  0  1  0  0  0  0  1  0  0  0  0  1  0  1  0  0  0  1  4  8  5  3
##         21  0  0  0  0  0  0  1  0  1  0  1  0  0  1  0  0  0  0  3  1  6  2
##         22  0  0  2  0  0  2  1  0  0  0  0  4  2  0  0  2  0  0  3  2  3  4
##
## Overall Statistics
##
##                Accuracy : 0.4603
##                  95% CI : (0.4149, 0.5061)
##     No Information Rate : 0.0607
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4338
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.61905  0.76190  0.69565  0.60870  0.70588  0.52174
## Specificity           0.98031  0.97374  0.96264  0.97582  0.97397  0.95824
## Pos Pred Value        0.59091  0.57143  0.48485  0.56000  0.50000  0.38710
## Neg Pred Value        0.98246  0.98889  0.98427  0.98013  0.98899  0.97539
## Prevalence            0.04393  0.04393  0.04812  0.04812  0.03556  0.04812
## Detection Rate        0.02720  0.03347  0.03347  0.02929  0.02510  0.02510
## Detection Prevalence  0.04603  0.05858  0.06904  0.05230  0.05021  0.06485
## Balanced Accuracy     0.79968  0.86782  0.82914  0.79226  0.83993  0.73999
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity           0.54167  0.69565  0.50000   0.34615   0.37500   0.38095
## Specificity           0.98018  0.98022  0.98035   0.98894   0.98458   0.96718
## Pos Pred Value        0.59091  0.64000  0.52632   0.64286   0.56250   0.34783
## Neg Pred Value        0.97588  0.98455  0.97821   0.96336   0.96753   0.97143
## Prevalence            0.05021  0.04812  0.04184   0.05439   0.05021   0.04393
## Detection Rate        0.02720  0.03347  0.02092   0.01883   0.01883   0.01674
## Detection Prevalence  0.04603  0.05230  0.03975   0.02929   0.03347   0.04812
## Balanced Accuracy     0.76092  0.83794  0.74017   0.66755   0.67979   0.67406
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity           0.066667   0.56000   0.46667   0.61111   0.51724
## Specificity           0.982721   0.98234   0.98488   0.99130   0.95100
## Pos Pred Value        0.111111   0.63636   0.50000   0.73333   0.40541
## Neg Pred Value        0.970149   0.97588   0.98276   0.98488   0.96825
## Prevalence            0.031381   0.05230   0.03138   0.03766   0.06067
## Detection Rate        0.002092   0.02929   0.01464   0.02301   0.03138
## Detection Prevalence  0.018828   0.04603   0.02929   0.03138   0.07741
## Balanced Accuracy     0.524694   0.77117   0.72577   0.80121   0.73412
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity           0.125000  0.190476   0.36364   0.23077  0.160000
## Specificity           0.980519  0.960613   0.96272   0.97788  0.953642
## Pos Pred Value        0.181818  0.181818   0.32000   0.37500  0.160000
## Neg Pred Value        0.970021  0.962719   0.96909   0.95671  0.953642
## Prevalence            0.033473  0.043933   0.04603   0.05439  0.052301
## Detection Rate        0.004184  0.008368   0.01674   0.01255  0.008368
## Detection Prevalence  0.023013  0.046025   0.05230   0.03347  0.052301
## Balanced Accuracy     0.552760  0.575544   0.66318   0.60432  0.556821
```

```
#get the accuracy of the prediction of gmb model with improved features
gbm_accuracy_test = mean(dat_test$emotion_idx == gbm_pred_class)
print(paste0("The accuracy for gbm model is: ", gbm_accuracy_test))
```

```
## [1] "The accuracy for gbm model is: 0.460251046025105"
```

3.Improved feature in SVM

```
#prediction of the svm model with improved features
source("../lib/test_SVM.R")
load("../output/svm_model.RData")
load("../output/feature_test.RData")

svm_result_test = svm_test(svm_model, dat_test)
svm_pred_class = svm_result_test[[1]]
tm.svm.test = svm_result_test[[2]]
```

```
#get the confusion matrix of the prediction
confusionMatrix(factor(svm_pred_class), 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  1  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##          2   0 16  0  0  0  0  0  5  5  0  0  0  0  0  0  0  0  0  0  0  0  0
##          3   3  0 14  0  0  0  0  0  0  3  0  0  1  1  0  1  0  0  1  0  1  1
##          4   0  0  0 14  0  1  0  0  0  6  1  1  1  0  0  1  0  0  0  1  0  1
##          5   0  0  0  0 15  0  1  0  0  0  0  0  0  0  0  0  1  3  0  0  0  0
##          6   0  0  0  2  0 14  0  0  0  0  4  1  0  0  0  1  0  0  0  0  1  1
##          7   0  0  1  0  0  0 15  0  0  0  0  0  0  0  0  1  3  0  0  0  0  0
##          8   0  1  0  0  0  0  0 17  0  0  0  0  0  0  0  0  1  0  0  0  0  0
##          9   0  4  0  0  0  0  0  0 14  1  0  1  0  0  0  0  0  0  0  0  0  1
##          10  0  0  1  2  0  0  0  0  0 12  3  0  2  0  0  0  0  0  0  0  0  0
##          11  0  0  0  0  0  0  0  0  0  1  9  2  3  0  0  0  0  0  0  0  0  2
##         12  0  0  0  0  0  2  0  0  0  0  5 12  5  0  0  0  0  0  0  0  0  2
##         13  0  0  1  3  0  2  0  0  0  1  2  1  3  0  0  0  0  0  0  0  0  2
##         14  0  0  0  0  1  2  1  0  0  0  0  1  0 18  2  0  0  0  1  0  2  0
##         15  0  0  1  0  0  0  1  0  0  0  0  0  0  2 10  0  0  0  3  1  0  0
##         16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11  0  0  0  0  0  0
##         17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 14  6  0  1  3  0
##         18  0  0  0  0  1  0  0  1  0  0  0  0  0  0  0  1  5  7  2  0  0  0
##         19  0  0  3  1  0  0  0  0  0  1  0  0  0  2  0  0  1  0  6  1  2  0
##         20  0  0  0  0  0  1  1  0  1  0  0  0  0  1  0  0  3  0  2  9  1  2
##         21  0  0  0  0  0  0  3  0  0  1  0  0  0  1  2  0  1  0  5  6 15  3
##         22  0  0  2  0  0  1  0  0  0  0  0  2  0  0  1  1  0  0  1  3  1 10
##
## Overall Statistics
##
##                Accuracy : 0.5711
##                  95% CI : (0.5254, 0.616)
##     No Information Rate : 0.0607
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5502
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.85714  0.76190  0.60870  0.60870  0.88235  0.60870
## Specificity           0.99344  0.97812  0.97363  0.97143  0.98915  0.97802
## Pos Pred Value        0.85714  0.61538  0.53846  0.51852  0.75000  0.58333
## Neg Pred Value        0.99344  0.98894  0.98009  0.98004  0.99563  0.98018
## Prevalence            0.04393  0.04393  0.04812  0.04812  0.03556  0.04812
## Detection Rate        0.03766  0.03347  0.02929  0.02929  0.03138  0.02929
## Detection Prevalence  0.04393  0.05439  0.05439  0.05649  0.04184  0.05021
## Balanced Accuracy     0.92529  0.87001  0.79116  0.79006  0.93575  0.79336
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity           0.62500  0.73913  0.70000   0.46154   0.37500   0.57143
## Specificity           0.98899  0.99560  0.98472   0.98230   0.98238   0.96937
## Pos Pred Value        0.75000  0.89474  0.66667   0.60000   0.52941   0.46154
## Neg Pred Value        0.98035  0.98693  0.98687   0.96943   0.96746   0.98009
## Prevalence            0.05021  0.04812  0.04184   0.05439   0.05021   0.04393
## Detection Rate        0.03138  0.03556  0.02929   0.02510   0.01883   0.02510
## Detection Prevalence  0.04184  0.03975  0.04393   0.04184   0.03556   0.05439
## Balanced Accuracy     0.80699  0.86737  0.84236   0.72192   0.67869   0.77040
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity           0.200000   0.72000   0.66667   0.61111   0.48276
## Specificity           0.974082   0.97792   0.98272   1.00000   0.97550
## Pos Pred Value        0.200000   0.64286   0.55556   1.00000   0.56000
## Neg Pred Value        0.974082   0.98444   0.98913   0.98501   0.96689
## Prevalence            0.031381   0.05230   0.03138   0.03766   0.06067
## Detection Rate        0.006276   0.03766   0.02092   0.02301   0.02929
## Detection Prevalence  0.031381   0.05858   0.03766   0.02301   0.05230
## Balanced Accuracy     0.587041   0.84896   0.82469   0.80556   0.72913
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity            0.43750   0.28571   0.40909   0.57692   0.40000
## Specificity           0.97835   0.97593   0.97368   0.95133   0.97351
## Pos Pred Value        0.41176   0.35294   0.42857   0.40541   0.45455
## Neg Pred Value        0.98048   0.96746   0.97155   0.97506   0.96711
## Prevalence            0.03347   0.04393   0.04603   0.05439   0.05230
## Detection Rate        0.01464   0.01255   0.01883   0.03138   0.02092
## Detection Prevalence  0.03556   0.03556   0.04393   0.07741   0.04603
## Balanced Accuracy     0.70793   0.63082   0.69139   0.76413   0.68675
```

```r
#get the accuracy of the prediction of the svm model with improved features
svm_accuracy_test = mean(dat_test$emotion_idx == svm_pred_class)
print(paste0("The accuracy for svm model is: ", svm_accuracy_test))
```

```
## [1] "The accuracy for svm model is: 0.571129707112971"
```

4.Improved feature in XGB

```r
#prediction of the xgb model with improved features
source("../lib/xgb_test.R")
load("../output/xgb_model.RData")
load("../output/feature_test.RData")

xgb_result_test = xgb_test(xgb_model, dat_test[,-ncol(dat_test)])
xgb_pred = xgb_result_test[[1]]
tm.xgb.test = xgb_result_test[[2]]
xgb_pred_class = apply(xgb_pred, 1, which.max)-1
```

```r
#get the confusion matrix of the prediction
confusionMatrix(factor(xgb_pred_class), 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  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##         2  0 14  0  0  0  1  0  4  4  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3  6  0 15  2  0  0  0  0  0  3  0  1  0  0  0  1  0  0  0  0  0  2
##         4  1  0  1 14  0  1  0  0  0  5  2  0  0  0  0  1  0  0  0  0  0  0
##         5  0  0  0  0 13  0  1  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0
##         6  0  0  0  2  0 14  0  0  0  0  2  4  0  0  0  0  0  0  0  0  0  2
##         7  0  0  0  0  1  0 18  2  0  0  0  0  0  0  2  0  4  0  0  0  1  0
##         8  0  2  0  0  0  0  0 15  1  0  0  0  0  0  0  0  2  0  0  0  0  0
##         9  0  5  0  0  0  0  0  0 12  1  1  0  2  0  0  0  0  0  0  2  1  2
##        10  0  0  0  0  0  0  0  0  0 10  2  0  1  0  0  0  0  0  0  0  0  0
##        11  0  0  0  3  0  2  0  0  1  2 10  1  3  0  0  0  0  0  0  0  0  1
##        12  0  0  1  1  0  0  0  0  0  0  5  8  7  0  0  0  0  0  0  0  0  1
##        13  0  0  1  0  0  1  0  0  0  3  1  0  2  1  0  0  0  0  0  0  2  1
##        14  0  0  0  0  2  1  1  0  0  0  0  2  0 18  2  0  1  0  1  2  2  1
##        15  0  0  0  0  0  0  0  0  0  0  0  0  0  4  9  0  0  0  1  0  3  0
##        16  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  2
##        17  0  0  0  0  1  0  0  2  0  0  0  0  0  0  0  0 13  8  1  1  3  0
##        18  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  5  7  1  0  0  0
##        19  0  0  1  0  0  2  1  0  1  1  0  0  0  2  1  0  2  0  9  3  2  3
##        20  0  0  0  0  0  0  1  0  1  0  0  0  0  0  1  0  1  0  2 11  4  3
##        21  0  0  0  0  0  1  2  0  0  0  1  0  0  0  0  0  0  0  5  2  8  1
##        22  0  0  2  0  0  0  0  0  0  1  0  5  0  0  0  2  0  0  1  1  0  6
```

```
##
## Overall Statistics
##
##                Accuracy : 0.5293
##                  95% CI : (0.4834, 0.5748)
##     No Information Rate : 0.0607
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5063
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.66667  0.66667  0.65217  0.60870  0.76471  0.60870
## Specificity           0.99562  0.98031  0.96703  0.97582  0.99349  0.97802
## Pos Pred Value        0.87500  0.60870  0.50000  0.56000  0.81250  0.58333
## Neg Pred Value        0.98485  0.98462  0.98214  0.98013  0.99134  0.98018
## Prevalence            0.04393  0.04393  0.04812  0.04812  0.03556  0.04812
## Detection Rate        0.02929  0.02929  0.03138  0.02929  0.02720  0.02929
## Detection Prevalence  0.03347  0.04812  0.06276  0.05230  0.03347  0.05021
## Balanced Accuracy     0.83115  0.82349  0.80960  0.79226  0.87910  0.79336
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity           0.75000  0.65217  0.60000   0.38462   0.41667   0.38095
## Specificity           0.97797  0.98901  0.96943   0.99336   0.97137   0.96718
## Pos Pred Value        0.64286  0.75000  0.46154   0.76923   0.43478   0.34783
## Neg Pred Value        0.98667  0.98253  0.98230   0.96559   0.96923   0.97143
## Prevalence            0.05021  0.04812  0.04184   0.05439   0.05021   0.04393
## Detection Rate        0.03766  0.03138  0.02510   0.02092   0.02092   0.01674
## Detection Prevalence  0.05858  0.04184  0.05439   0.02720   0.04812   0.04812
## Balanced Accuracy     0.86399  0.82059  0.78472   0.68899   0.69402   0.67406
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity           0.133333   0.72000   0.60000   0.72222   0.44828
## Specificity           0.978402   0.96689   0.98272   0.99348   0.96437
## Pos Pred Value        0.166667   0.54545   0.52941   0.81250   0.44828
## Neg Pred Value        0.972103   0.98427   0.98698   0.98918   0.96437
## Prevalence            0.031381   0.05230   0.03138   0.03766   0.06067
## Detection Rate        0.004184   0.03766   0.01883   0.02720   0.02720
## Detection Prevalence  0.025105   0.06904   0.03556   0.03347   0.06067
## Balanced Accuracy     0.555868   0.84344   0.79136   0.85785   0.70632
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity            0.43750   0.42857   0.50000   0.30769   0.24000
## Specificity            0.98485   0.95842   0.97149   0.97345   0.97351
## Pos Pred Value         0.50000   0.32143   0.45833   0.40000   0.33333
## Neg Pred Value         0.98060   0.97333   0.97577   0.96070   0.95870
## Prevalence             0.03347   0.04393   0.04603   0.05439   0.05230
## Detection Rate         0.01464   0.01883   0.02301   0.01674   0.01255
## Detection Prevalence   0.02929   0.05858   0.05021   0.04184   0.03766
## Balanced Accuracy      0.71117   0.69350   0.73575   0.64057   0.60675
```

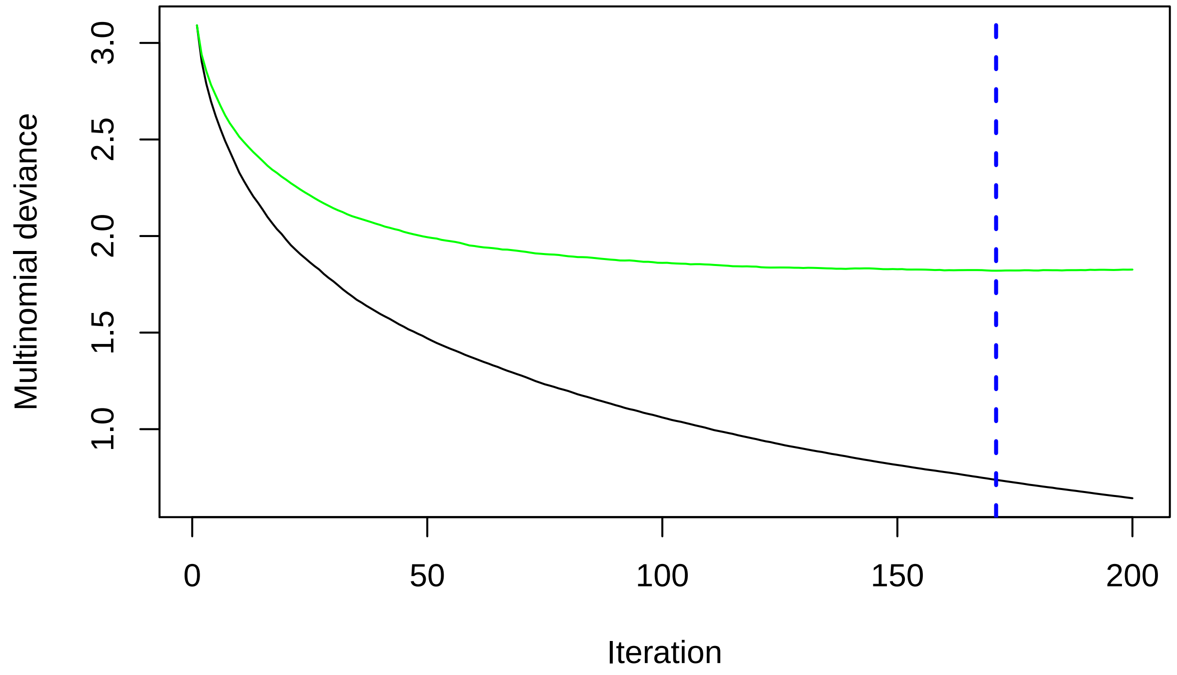
```r
#get the accuracy of the prediction of the xgb model with improved features
xgb_accuracy_test = mean(dat_test$emotion_idx == xgb_pred_class)
print(paste0("The accuracy for xgb model is: ", xgb_accuracy_test))
```

```
## [1] "The accuracy for xgb model is: 0.52928870292887"
```

5.Improved feature with color in GBM

```
#prediction of the gbm model with improved features and colors features
source("../lib/test_gbm.R")
load("../output/gbm_model_color.RData")
load("../output/feature_test_color.RData")

gbm_result_test_color = gbm_test(gbm_model_color, dat_test_color)
```



```
gbm_pred_color = gbm_result_test_color[[1]]
tm.gbm.test_color = gbm_result_test_color[[2]]
gbm_pred_class_color = apply(gbm_pred_color, 1, which.max)
```

```
#get the confusion matrix of the prediction
confusionMatrix(factor(gbm_pred_class_color), dat_test_color$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 16  0  1  1  0  1  1  0  0  1  1  0  0  0  0  0  0  1  0  0  0  0
##          2  0 19  0  0  0  0  0  6  6  0  0  0  0  0  0  0  0  0  0  0  0  0
##          3  4  0 18  2  0  0  0  0  0  5  1  1  1  0  0  2  0  0  0  0  1  2
##          4  1  0  0 16  0  1  1  0  0  6  1  0  1  0  0  1  0  0  0  0  0  0
##          5  0  0  0  0 12  0  1  0  0  0  0  0  0  2  3  0  1  2  1  0  1  0
##         6   0  0  0  2  0 13  0  0  1  1  4  4  2  0  1  0  0  0  0  0  2  2
##         7   0  0  0  0  1  0 13  1  1  0  0  0  0  1  1  1  2  0  2  0  0  0
##         8   0  0  0  0  0  0  1 13  1  0  0  0  0  0  0  0  3  2  0  0  0  0
##         9   0  1  0  0  0  0  0  1 10  0  0  0  1  0  0  0  0  0  0  1  1  1
##         10  0  0  0  0  0  0  0  0  0  7  1  0  2  0  0  0  0  0  0  0  1  0
##         11  0  0  0  0  0  2  0  0  0  1  9  2  0  0  0  0  0  0  0  2  0  0
##         12  0  0  0  0  0  1  0  0  1  1  5  8  5  0  0  0  0  0  0  0  0  2
##         13  0  0  1  2  0  1  0  0  0  2  0  1  1  1  0  0  0  0  0  0  0  1
##         14  0  0  0  0  1  1  0  0  0  0  0  1  0 15  2  0  0  0  0  0  1  0
##         15  0  0  0  0  0  0  0  0  0  0  0  0  0  3  7  0  0  0  4  0  1  0
##         16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 10  0  0  0  0  0  3
##         17  0  0  0  0  1  0  1  2  0  0  0  0  0  1  0  1 14  6  1  3  4  0
##         18  0  0  0  0  2  0  1  0  0  0  0  0  0  0  0  1  5  4  0  0  0  0
##         19  0  0  1  0  0  2  1  0  0  2  0  0  0  1  1  0  3  1  5  2  2  5
##         20  0  0  0  0  0  0  2  0  0  0  0  1  0  0  0  0  1  0  4 11  4  4
##         21  0  1  0  0  0  0  1  0  0  0  1  0  0  1  0  0  0  0  4  2  7  1
##         22  0  0  2  0  0  1  1  0  0  0  1  3  2  0  0  2  0  0  0  1  1  4
##
## Overall Statistics
##
##                Accuracy : 0.4854
##                  95% CI : (0.4397, 0.5312)
##     No Information Rate : 0.0607
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4603
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.76190  0.90476  0.78261  0.69565  0.70588  0.56522
## Specificity           0.98468  0.97374  0.95824  0.97363  0.97614  0.95824
## Pos Pred Value        0.69565  0.61290  0.48649  0.57143  0.52174  0.40625
## Neg Pred Value        0.98901  0.99553  0.98866  0.98444  0.98901  0.97758
## Prevalence            0.04393  0.04393  0.04812  0.04812  0.03556  0.04812
## Detection Rate        0.03347  0.03975  0.03766  0.03347  0.02510  0.02720
## Detection Prevalence  0.04812  0.06485  0.07741  0.05858  0.04812  0.06695
## Balanced Accuracy     0.87329  0.93925  0.87043  0.83464  0.84101  0.76173
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity           0.54167  0.56522  0.50000   0.26923   0.37500   0.38095
## Specificity           0.97797  0.98462  0.98690   0.99115   0.98458   0.96718
## Pos Pred Value        0.56522  0.65000  0.62500   0.63636   0.56250   0.34783
## Neg Pred Value        0.97582  0.97817  0.97835   0.95931   0.96753   0.97143
## Prevalence            0.05021  0.04812  0.04184   0.05439   0.05021   0.04393
## Detection Rate        0.02720  0.02720  0.02092   0.01464   0.01883   0.01674
## Detection Prevalence  0.04812  0.04184  0.03347   0.02301   0.03347   0.04812
## Balanced Accuracy     0.75982  0.77492  0.74345   0.63019   0.67979   0.67406
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity           0.066667   0.60000   0.46667   0.55556   0.48276
## Specificity           0.980562   0.98675   0.98272   0.99348   0.95546
## Pos Pred Value        0.100000   0.71429   0.46667   0.76923   0.41176
## Neg Pred Value        0.970085   0.97812   0.98272   0.98280   0.96622
## Prevalence             0.031381   0.05230   0.03138   0.03766   0.06067
## Detection Rate         0.002092   0.03138   0.01464   0.02092   0.02929
## Detection Prevalence   0.020921   0.04393   0.03138   0.02720   0.07113
## Balanced Accuracy      0.523614   0.79338   0.72469   0.77452   0.71911
##                       Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity            0.250000   0.23810   0.50000   0.26923  0.160000
## Specificity            0.980519   0.95405   0.96491   0.97566  0.969095
## Pos Pred Value         0.307692   0.19231   0.40741   0.38889  0.222222
## Neg Pred Value         0.974194   0.96460   0.97561   0.95870  0.954348
## Prevalence             0.033473   0.04393   0.04603   0.05439  0.052301
## Detection Rate         0.008368   0.01046   0.02301   0.01464  0.008368
## Detection Prevalence   0.027197   0.05439   0.05649   0.03766  0.037657
## Balanced Accuracy      0.615260   0.59607   0.73246   0.62245  0.564547
```

```
#get the accuracy of the prediction of the gbm model with improved features and color features
gbm_accuracy_test_color = mean(dat_test_color$emotion_idx == gbm_pred_class_color)
print(paste0("The accuracy for gbm model is: ", gbm_accuracy_test_color))
```

```
## [1] "The accuracy for gbm model is: 0.485355648535565"
```

6.Improved feature with color in SVM

```
#prediction of the svm model with improved features and colors features
source("../lib/test_SVM.R")
load("../output/svm_model_color.RData")
load("../output/feature_test_color.RData")

svm_result_test_color = svm_test(svm_model_color, dat_test_color)
svm_pred_class_color = svm_result_test_color[[1]]
tm.svm.test_color = svm_result_test_color[[2]]
```

```
#get the confusion matrix of the prediction
confusionMatrix(factor(svm_pred_class_color), dat_test_color$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 19  0  1  1  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##          2  0 19  0  0  0  0  0  5  3  0  0  0  0  0  0  0  0  0  0  0  0  0
##          3  2  0 14  0  0  0  0  0  0  1  1  0  1  0  0  1  0  0  0  0  1  1
##          4  0  0  0 14  0  1  0  0  0  7  1  1  1  0  0  1  0  0  0  0  0  0
##          5  0  0  0  0 16  0  1  0  0  0  0  0  0  0  0  0  1  2  0  0  0  0
##          6  0  0  0  2  0 13  0  0  0  0  2  1  0  0  0  0  0  0  0  0  0  2
##          7  0  0  0  0  0  0 16  0  0  0  0  0  0  0  2  1  2  0  1  0  1  0
##          8  0  1  0  0  0  0  0 16  0  0  0  0  0  0  0  0  2  0  0  0  0  0
##          9  0  1  0  0  0  0  0  0 17  1  0  0  0  0  0  0  0  0  0  1  1  1
##         10  0  0  1  1  0  0  0  0  0 11  1  0  2  0  0  0  0  0  0  0  1  0
##         11  0  0  0  1  0  1  0  0  0  1 11  2  1  0  0  0  0  0  0  0  0  1
##         12  0  0  0  0  0  1  0  0  0  1  7 12  5  0  0  0  0  0  0  0  0  3
##         13  0  0  1  3  0  3  0  0  0  1  1  2  4  1  0  0  0  0  0  1  1  2
##         14  0  0  0  0  0  1  0  0  0  0  0  1  0 18  0  0  0  0  1  0  1  0
##         15  0  0  0  1  0  0  1  0  0  0  0  0  0  3  9  0  0  0  2  0  0  0
##         16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0  0  0  0
##         17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 17  5  2  2  3  0
##         18  0  0  0  0  1  0  0  1  0  0  0  0  0  0  0  1  6  9  1  0  0  0
##         19  0  0  4  0  0  0  1  0  0  1  0  0  0  1  2  0  0  0  7  1  1  0
##         20  0  0  0  0  0  0  1  1  0  0  0  0  1  0  0  0  0  1 10  1  2
##         21  0  0  0  0  0  1  2  0  0  1  0  0  0  1  1  0  1  0  5  4 14  2
##         22  0  0  2  0  0  2  0  0  0  1  0  2  1  0  1  1  0  0  1  3  1 11
##
## Overall Statistics
##
##                Accuracy : 0.6046
##                  95% CI : (0.5592, 0.6487)
##     No Information Rate : 0.0607
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5854
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.90476  0.90476  0.60870  0.60870  0.94118  0.56522
## Specificity           0.99125  0.98249  0.98242  0.97363  0.99132  0.98462
## Pos Pred Value        0.82609  0.70370  0.63636  0.53846  0.80000  0.65000
## Neg Pred Value        0.99560  0.99557  0.98026  0.98009  0.99782  0.97817
## Prevalence            0.04393  0.04393  0.04812  0.04812  0.03556  0.04812
## Detection Rate        0.03975  0.03975  0.02929  0.02929  0.03347  0.02720
## Detection Prevalence  0.04812  0.05649  0.04603  0.05439  0.04184  0.04184
## Balanced Accuracy     0.94800  0.94363  0.79556  0.79116  0.96625  0.77492
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity           0.66667  0.69565  0.85000   0.42308   0.45833   0.57143
## Specificity           0.98458  0.99341  0.98908   0.98673   0.98458   0.96280
## Pos Pred Value        0.69565  0.84211  0.77273   0.64706   0.61111   0.41379
## Neg Pred Value        0.98242  0.98475  0.99342   0.96746   0.97174   0.97996
## Prevalence            0.05021  0.04812  0.04184   0.05439   0.05021   0.04393
## Detection Rate        0.03347  0.03347  0.03556   0.02301   0.02301   0.02510
## Detection Prevalence  0.04812  0.03975  0.04603   0.03556   0.03766   0.06067
## Balanced Accuracy     0.82562  0.84453  0.91954   0.70490   0.72146   0.76711
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity           0.266667   0.72000   0.60000   0.66667   0.58621
## Specificity           0.965443   0.99117   0.98488   1.00000   0.97105
## Pos Pred Value        0.200000   0.81818   0.56250   1.00000   0.56667
## Neg Pred Value        0.975983   0.98465   0.98701   0.98712   0.97321
## Prevalence            0.031381   0.05230   0.03138   0.03766   0.06067
## Detection Rate        0.008368   0.03766   0.01883   0.02510   0.03556
## Detection Prevalence  0.041841   0.04603   0.03347   0.02510   0.06276
## Balanced Accuracy     0.616055   0.85558   0.79244   0.83333   0.77863
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity            0.56250   0.33333   0.45455   0.53846   0.44000
## Specificity            0.97835   0.97593   0.98465   0.96018   0.96689
## Pos Pred Value         0.47368   0.38889   0.58824   0.43750   0.42308
## Neg Pred Value         0.98475   0.96957   0.97397   0.97309   0.96903
## Prevalence             0.03347   0.04393   0.04603   0.05439   0.05230
## Detection Rate         0.01883   0.01464   0.02092   0.02929   0.02301
## Detection Prevalence   0.03975   0.03766   0.03556   0.06695   0.05439
## Balanced Accuracy      0.77043   0.65463   0.71960   0.74932   0.70344
```

```
#get the accuracy of the prediction of the gbm model with improved features and color features
svm_accuracy_test_color = mean(dat_test_color$emotion_idx == svm_pred_class_color)
print(paste0("The accuracy for svm model is: ", svm_accuracy_test_color))
```

```
## [1] "The accuracy for svm model is: 0.604602510460251"
```

7.Improved feature with color in XGB

```
#prediction of the xgb model with improved features and colors features
source("../lib/xgb_test.R")
load("../output/xgb_model_color.RData")
load("../output/feature_test_color.RData")

xgb_result_test_color = xgb_test(xgb_model_color, dat_test_color[,-ncol(dat_test_color)])
xgb_pred_color = xgb_result_test_color[[1]]
tm.xgb.test_color = xgb_result_test_color[[2]]
xgb_pred_class_color = apply(xgb_pred_color, 1, which.max)-1
```

```
#get the confusion matrix of the prediction
confusionMatrix(factor(xgb_pred_class_color), dat_test_color$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  2  0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  0  0  0  0
##         2   0 16  0  0  0  0  0  3  4  0  0  0  0  0  0  0  0  0  0  0  0  0
##         3   1  0 19  3  0  0  0  0  0  2  3  0  0  0  0  1  0  0  0  0  0  2
##         4   1  0  0 17  0  1  0  0  0  5  1  0  0  0  0  1  0  0  0  0  0  0
##         5   0  0  0  0 14  0  1  0  0  0  0  0  0  1  0  0  1  1  0  0  0  0
##         6   0  0  0  2  0 14  0  0  0  0  3  4  0  0  0  0  0  0  0  0  0  2
##         7   0  0  0  0  1  0 16  2  0  0  0  0  0  0  2  0  4  0  0  0  1  0
##         8   0  2  0  0  0  0  1 17  1  0  0  0  0  0  0  0  2  0  0  0  0  0
##         9   0  3  0  0  0  0  0  0 13  1  0  0  2  0  0  0  0  0  0  2  1  1
##         10  1  0  0  0  0  0  0  0  0 12  2  0  1  0  0  0  0  0  0  0  1  0
##         11  0  0  0  1  0  3  0  0  1  2  8  0  2  0  0  0  0  0  0  0  0  1
##         12  0  0  0  0  0  0  0  0  0  0  5  9  6  0  0  0  0  0  0  0  0  0
##         13  0  0  1  0  0  1  0  0  0  3  1  1  1  1  0  0  0  0  0  0  1  1
##         14  0  0  0  0  0  1  1  0  0  0  0  2  0 19  0  0  0  0  0  0  2  1
##         15  0  0  0  0  1  0  0  0  0  0  0  0  0  3 10  0  0  0  2  0  3  0
##         16  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0 12  0  0  0  0  0  2
##         17  0  0  0  0  1  0  0  1  0  0  0  0  0  0  0  0 15  9  2  2  3  0
##         18  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  1  3  6  1  0  0  0
##         19  0  0  1  0  0  1  1  0  0  1  0  0  0  1  1  0  2  0  7  4  0  3
##         20  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  2  0  2 11  4  3
##         21  0  0  0  0  0  1  1  0  1  0  1  0  0  0  2  0  0  0  6  3  9  3
##         22  0  0  0  0  0  1  0  0  0  0  0  5  1  0  0  2  0  0  1  0  1  6
##
## Overall Statistics
##
##                Accuracy : 0.5628
##                  95% CI : (0.517, 0.6078)
##     No Information Rate : 0.0607
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5413
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.85714  0.76190  0.82609  0.73913  0.82353  0.60870
## Specificity           0.99125  0.98468  0.97363  0.98022  0.99132  0.97582
## Pos Pred Value        0.81818  0.69565  0.61290  0.65385  0.77778  0.56000
## Neg Pred Value        0.99342  0.98901  0.99105  0.98673  0.99348  0.98013
## Prevalence            0.04393  0.04393  0.04812  0.04812  0.03556  0.04812
## Detection Rate        0.03766  0.03347  0.03975  0.03556  0.02929  0.02929
## Detection Prevalence  0.04603  0.04812  0.06485  0.05439  0.03766  0.05230
## Balanced Accuracy     0.92420  0.87329  0.89986  0.85968  0.90743  0.79226
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity           0.66667  0.73913  0.65000   0.46154   0.33333   0.42857
## Specificity           0.97797  0.98681  0.97817   0.98894   0.97797   0.97593
## Pos Pred Value        0.61538  0.73913  0.56522   0.70588   0.44444   0.45000
## Neg Pred Value        0.98230  0.98681  0.98462   0.96963   0.96522   0.97380
## Prevalence            0.05021  0.04812  0.04184   0.05439   0.05021   0.04393
## Detection Rate        0.03347  0.03556  0.02720   0.02510   0.01674   0.01883
## Detection Prevalence  0.05439  0.04812  0.04812   0.03556   0.03766   0.04184
## Balanced Accuracy     0.82232  0.86297  0.81408   0.72524   0.65565   0.70225
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 17
## Sensitivity           0.066667   0.76000   0.66667   0.66667   0.51724
## Specificity           0.978402   0.98455   0.98056   0.99348   0.95991
## Pos Pred Value        0.090909   0.73077   0.52632   0.80000   0.45455
## Neg Pred Value        0.970021   0.98673   0.98911   0.98704   0.96854
## Prevalence            0.031381   0.05230   0.03138   0.03766   0.06067
## Detection Rate        0.002092   0.03975   0.02092   0.02510   0.03138
## Detection Prevalence  0.023013   0.05439   0.03975   0.03138   0.06904
## Balanced Accuracy     0.522534   0.87227   0.82361   0.83007   0.73858
##                      Class: 18 Class: 19 Class: 20 Class: 21 Class: 22
## Sensitivity            0.37500   0.33333   0.50000   0.34615   0.24000
## Specificity            0.98701   0.96718   0.97149   0.96018   0.97572
## Pos Pred Value         0.50000   0.31818   0.45833   0.33333   0.35294
## Neg Pred Value         0.97854   0.96930   0.97577   0.96231   0.95879
## Prevalence             0.03347   0.04393   0.04603   0.05439   0.05230
## Detection Rate         0.01255   0.01464   0.02301   0.01883   0.01255
## Detection Prevalence   0.02510   0.04603   0.05021   0.05649   0.03556
## Balanced Accuracy      0.68101   0.65026   0.73575   0.65317   0.60786
```

```r
#get the accuracy of the prediction of the gbm model with improved features and color features
xgb_accuracy_test_color = mean(dat_test_color$emotion_idx == xgb_pred_class_color)
print(paste0("The accuracy for xgb model is: ", xgb_accuracy_test_color))
```

```
## [1] "The accuracy for xgb model is: 0.562761506276151"
```

Step 6: Conclusion

Summarize Accuracy

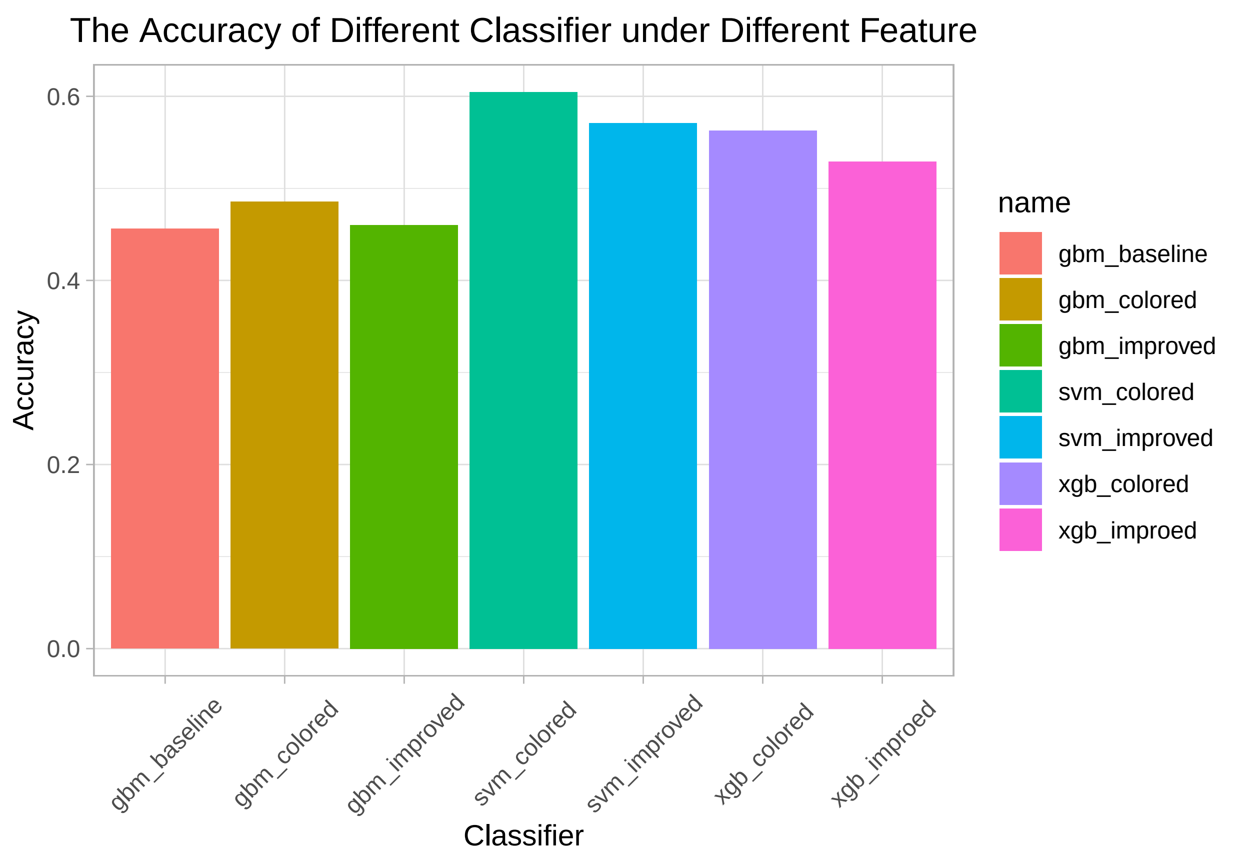
```
my_theme = theme_light() +
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 0.5, vjust = 0.5))
```

```
#create a tibble of predict accuracy of each model
if(run.baseline){
  accuracy_test = tibble(gbm_baseline = gbm_accuracy_test_baseline,
                         gbm_improved = gbm_accuracy_test,
                         svm_improved = svm_accuracy_test,
                         xgb_improed = xgb_accuracy_test,
                         gbm_colored = gbm_accuracy_test_color,
                         svm_colored = svm_accuracy_test_color,
                         xgb_colored = xgb_accuracy_test_color)
}else{
  accuracy_test = tibble(gbm_improved = gbm_accuracy_test,
                         svm_improved = svm_accuracy_test,
                         xgb_improed  = xgb_accuracy_test,
                         gbm_colored = gbm_accuracy_test_color,
                         svm_colored = svm_accuracy_test_color,
                         xgb_colored = xgb_accuracy_test_color)
}
```

```
g_accuracy_test = accuracy_test %>% pivot_longer(1:ncol(accuracy_test))%>%
  ggplot(aes(x = name, fill = name)) +
  geom_bar(aes(weight = value)) +
  labs(x = 'Classifier', y = 'Accuracy',
       title = 'The Accuracy of Different Classifier under Different Feature') +
  my_theme
ggsave("../figs/g_accuracy_test.jpg", plot = g_accuracy_test)
```

```
## Saving 6.5 x 4.5 in image
```

```
g_accuracy_test
```



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, we would want a model woth relatively high accuracy and relatively low running time.

```r
load("../output/tm.run.all_points.RData")
load("../output/tm_feature_train_baseline.RData")
load("../output/tm_feature_test_baseline.RData")
load("../output/tm_feature_train.RData")
load("../output/tm_feature_train_color.RData")
load("../output/tm_feature_test.RData")
load("../output/tm_feature_test_color.RData")
load("../output/tm.gbm.train_baseline.RData")
load("../output/tm.gbm.train.RData")
load("../output/tm.svm.train.RData")
load("../output/tm.xgb.train.RData")
load("../output/tm.gbm.train_color.RData")
load("../output/tm.svm.train_color.RData")
load("../output/tm.xgb.train_color.RData")
```

```r
#get a tibble of the running time, tm.train for the training and tm.test for the test
if(run.baseline){
  tm.train = tibble(gbm_baseline = tm.gbm.train_baseline[3],
                    gbm_improved = tm.gbm.train[3],
                    svm_improved = tm.svm.train[3],
                    xgb_improved = tm.xgb.train[3],
                    gbm_color = tm.gbm.train_color[3],
                    svm_color = tm.svm.train_color[3],
                    xgb_color = tm.xgb.train_color[3])
  tm.test= tibble(gbm_baseline = tm.gbm.test_baseline[3],
                  gbm_improved = tm.gbm.test[3],
                  svm_improved = tm.svm.test[3],
                  xgb_improved = tm.xgb.test[3],
                  gbm_color = tm.gbm.test_color[3],
                  svm_color = tm.svm.test_color[3],
                  xgb_color = tm.xgb.test_color[3])
}else{
  tm.train = tibble(gbm_improved = tm.gbm.train[3],
                    svm_improved = tm.svm.train[3],
                    xgb_improved = tm.xgb.train[3],
                    gbm_color = tm.gbm.train_color[3],
                    svm_color = tm.svm.train_color[3],
                    xgb_color = tm.xgb.train_color[3])
  tm.test = tibble(gbm_improved = tm.gbm.test[3],
                   svm_improved = tm.svm.test[3],
                   xgb_improved = tm.xgb.test[3],
                   gbm_color = tm.gbm.test_color[3],
                   svm_color = tm.svm.test_color[3],
                   xgb_color = tm.xgb.test_color[3])
}
```

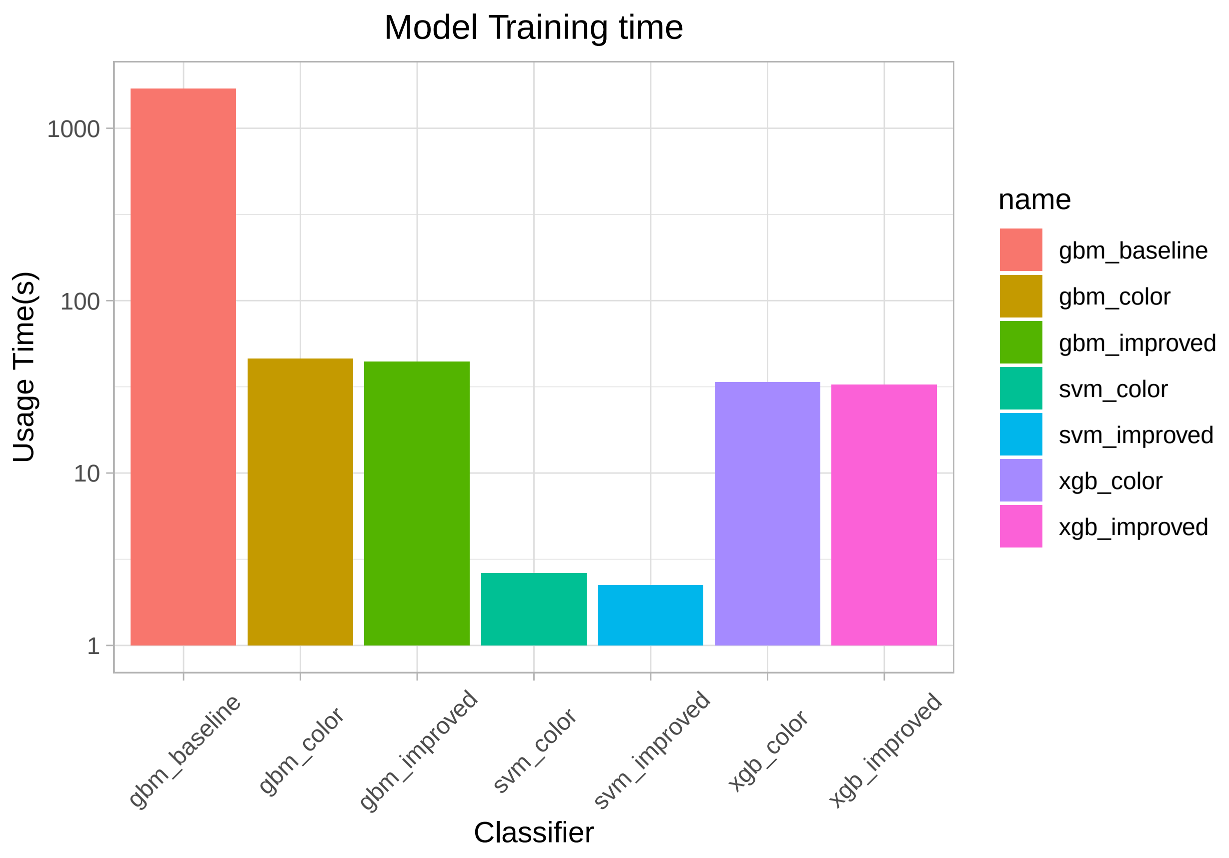
```
#get a tibble of the total running time which used the formula at the begining of the report
#for training, total running time is the sum of data manipulation, extraction of features, and trainig
#for test, the total running time is the sum of data manipulation, extraction of feature, and testing
if(run.baseline){
  tm.train.with = tibble(gbm_baseline = tm.gbm.train_baseline[3]+tm_feature_train_baseline[3],
                    gbm_improved = tm.gbm.train[3]+tm_feature_train[3]+tm.run.all_points[3]*0.8,
                    svm_improved = tm.svm.train[3]+tm_feature_train[3]+tm.run.all_points[3]*0.8,
                    xgb_improved = tm.xgb.train[3]+tm_feature_train[3]+tm.run.all_points[3]*0.8,
                    gbm_color = tm.gbm.train_color[3]+tm_feature_train_color[3]+tm.run.all_points[3]*0.
                    svm_color = tm.svm.train_color[3]+tm_feature_train_color[3]+tm.run.all_points[3]*0.
                    xgb_color = tm.xgb.train_color[3]+tm_feature_train_color[3]+tm.run.all_points[3]*0.
  tm.test.with= tibble(gbm_baseline = tm.gbm.test_baseline[3]+tm_feature_test_baseline[3],
                  gbm_improved = tm.gbm.test[3]+tm_feature_test[3]+tm.run.all_points[3]*0.2,
                  svm_improved = tm.svm.test[3]+tm_feature_test[3]+tm.run.all_points[3]*0.2,
                  xgb_improved = tm.xgb.test[3]+tm_feature_test[3]+tm.run.all_points[3]*0.2,
                  gbm_color = tm.gbm.test_color[3]+tm_feature_test_color[3]+tm.run.all_points[3]*0.2,
                  svm_color = tm.svm.test_color[3]+tm_feature_test_color[3]+tm.run.all_points[3]*0.2,
                  xgb_color = tm.xgb.test_color[3]+tm_feature_test_color[3]+tm.run.all_points[3]*0.2)
}else{
  tm.train.with = tibble(gbm_improved = tm.gbm.train[3]+tm_feature_train[3]+tm.run.all_points[3]*0.8,
                    svm_improved = tm.svm.train[3]+tm_feature_train[3]+tm.run.all_points[3]*0.8,
                    xgb_improved = tm.xgb.train[3]+tm_feature_train[3]+tm.run.all_points[3]*0.8,
                    gbm_color = tm.gbm.train_color[3]+tm_feature_train_color[3]+tm.run.all_points[3]*0.
                    svm_color = tm.svm.train_color[3]+tm_feature_train_color[3]+tm.run.all_points[3]*0.
                    xgb_color = tm.xgb.train_color[3]+tm_feature_train_color[3]+tm.run.all_points[3]*0.
  tm.test.with = tibble(gbm_improved = tm.gbm.test[3]+tm_feature_test[3]+tm.run.all_points[3]*0.2,
                   svm_improved = tm.svm.test[3]+tm_feature_test[3]+tm.run.all_points[3]*0.2,
                   xgb_improved = tm.xgb.test[3]+tm_feature_test[3]+tm.run.all_points[3]*0.2,
                   gbm_color = tm.gbm.test_color[3]+tm_feature_test_color[3]+tm.run.all_points[3]*0.2,
                   svm_color = tm.svm.test_color[3]+tm_feature_test_color[3]+tm.run.all_points[3]*0.2,
                   xgb_color = tm.xgb.test_color[3]+tm_feature_test_color[3]+tm.run.all_points[3]*0.2)
}
```

```
#plot the train time without feature extraction
g_time_train = tm.train %>% pivot_longer(1:ncol(tm.train)) %>%
  ggplot(aes(x = name, fill = name)) +
  geom_bar(aes(weight = value)) +
  scale_y_log10() +
  labs(x = 'Classifier', y = 'Usage Time(s)',
       title = 'Model Training time') +
  my_theme
ggsave("../figs/g_time_train.jpg", plot = g_time_train)
```

Training time without feature extraction

```
## Saving 6.5 x 4.5 in image
```

```
g_time_train
```

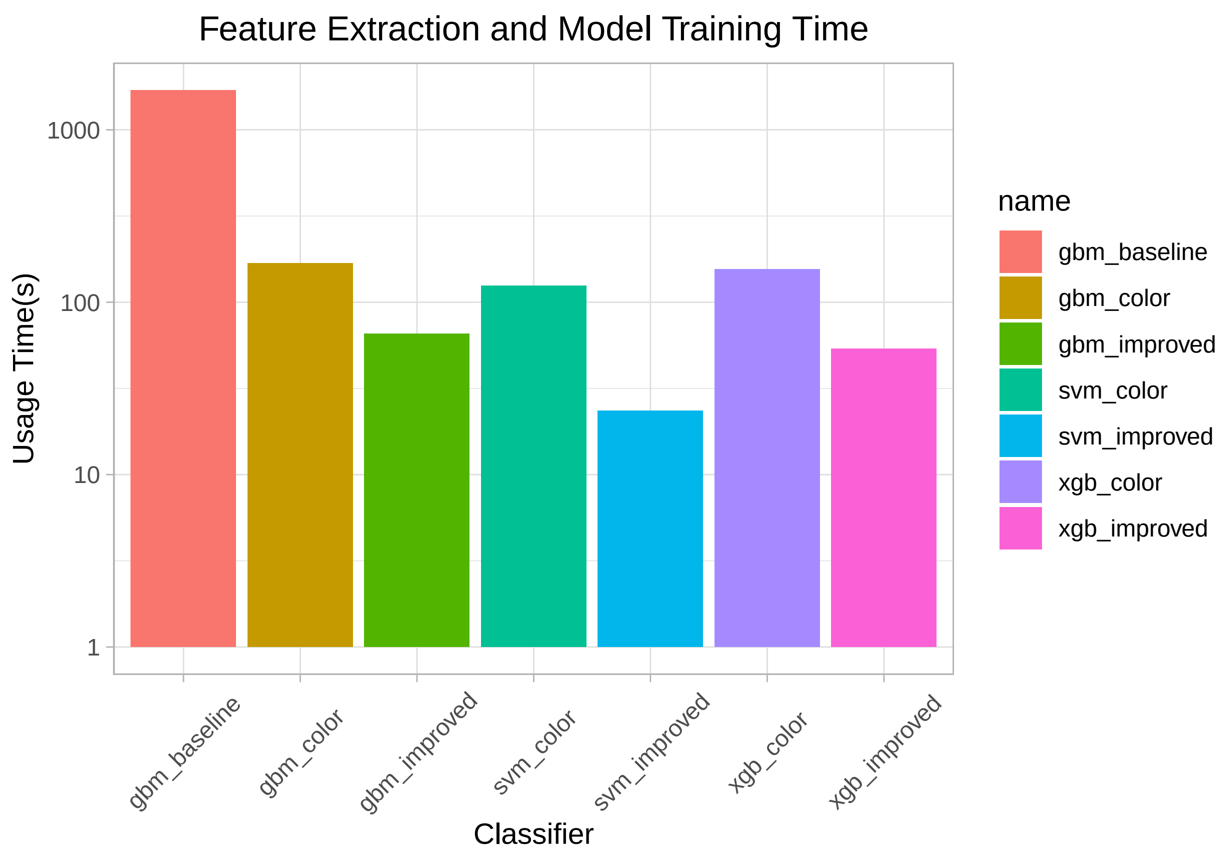


```
#plot the train time with feature extraction
g_time_train_with = tm.train.with %>% pivot_longer(1:ncol(tm.train.with)) %>%
  ggplot(aes(x = name, fill = name)) +
  geom_bar(aes(weight = value)) +
  scale_y_log10() +
  labs(x = 'Classifier', y = 'Usage Time(s)',
       title = 'Feature Extraction and Model Training Time') +
  my_theme
ggsave("../figs/g_time_train_with.jpg", plot = g_time_train_with)
```

Training time with feature extraction

```
## Saving 6.5 x 4.5 in image
```

```
g_time_train_with
```

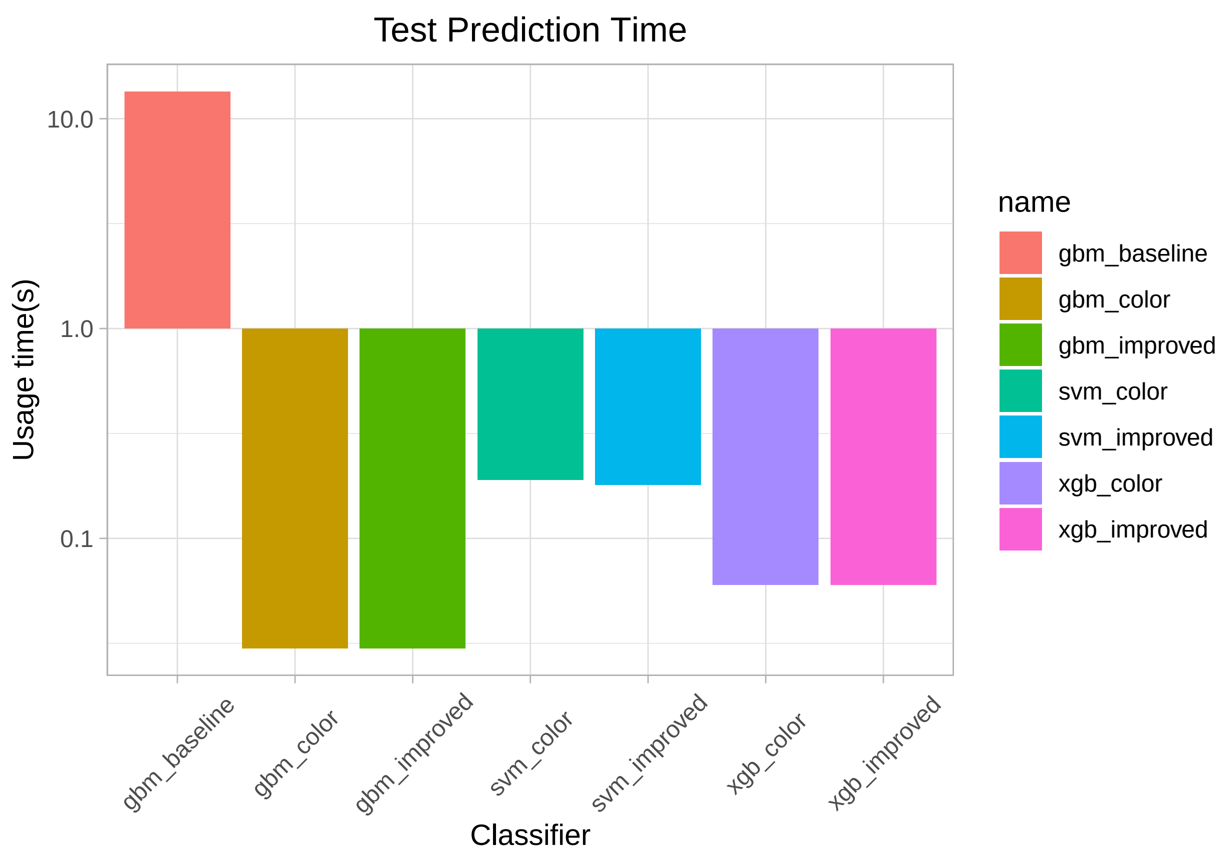


```
#plot the test time without feature extraction
g_time_test = tm.test %>% pivot_longer(1:ncol(tm.test)) %>%
  ggplot(aes(x = name, fill = name)) +
  geom_bar(aes(weight = value),) +
  labs(x = 'Classifier', y = 'Usage time(s)',
       title = 'Test Prediction Time') +
  scale_y_log10() +
  my_theme
ggsave("../figs/g_time_test.jpg", plot = g_time_test)
```

Test time without feature extraction

```
## Saving 6.5 x 4.5 in image
```

```
g_time_test
```

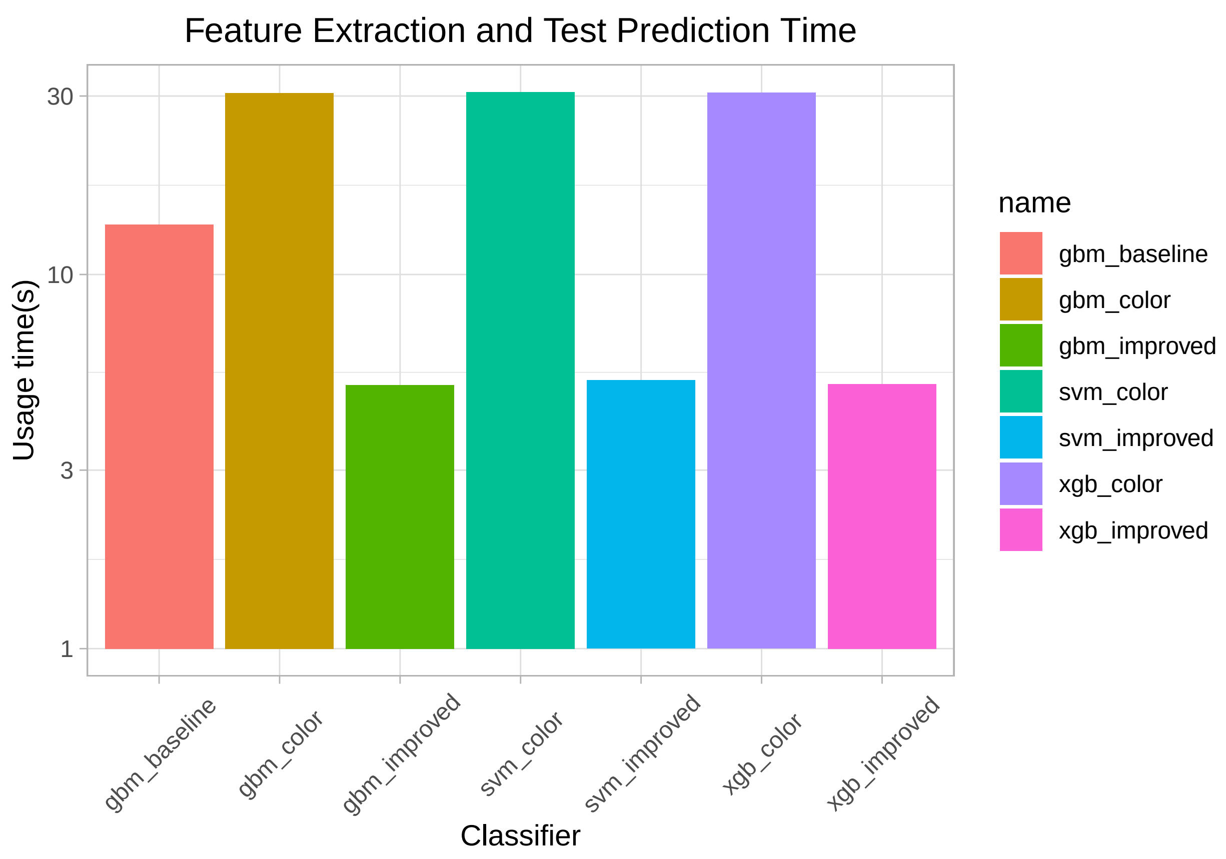


```
#plot the test time with feature extraction
g_time_test_with = tm.test.with %>% pivot_longer(1:ncol(tm.test.with)) %>%
  ggplot(aes(x = name, fill = name)) +
  geom_bar(aes(weight = value),) +
  scale_y_log10() +
  labs(x = 'Classifier', y = 'Usage time(s)',
       title = 'Feature Extraction and Test Prediction Time') +
  my_theme
ggsave("../figs/g_time_test_with.jpg", plot = g_time_test_with)
```

Test time with feature extraction

```
## Saving 6.5 x 4.5 in image
```

```
g_time_test_with
```



Reference

- Du, S., Tao, Y., & Martinez, A. M. (2014). Compound facial expressions of emotion. Proceedings of the National Academy of Sciences, 111(15), E1454-E1462.
- Cross-Validation on Xgboost, Fall2019 Project3 Section2 Group5.