# Main

In your final repo, there should be an R markdown file that organizes **all computational steps** for evaluating your proposed Facial Expression Recognition framework.

This file is currently a template for running evaluation experiments. You should update it according to your codes but following precisely the same structure.

```
if(!require("EBImage")){
  source("https://bioconductor.org/biocLite.R")
  biocLite("EBImage")
}
```

```
## Loading required package: EBImage
```

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

```
## Loading required package: R.matlab
```

```
## R.matlab v3.6.2 (2018-09-26) successfully loaded. See ?R.matlab for help.
```

```
##
## Attaching package: 'R.matlab'
```

```
## The following objects are masked from 'package:base':
##
##     getOption, isOpen
```

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

```
## Loading required package: readxl
```

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

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:EBImage':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

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

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

```
## Loading required package: ggplot2
```

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
library(R.matlab)
library(readxl)
library(dplyr)
library(EBImage)
library(ggplot2)
library(caret)
```

### Step 0 set work directories, extract paths, summarize

```
set.seed(0)
setwd("~/Google Drive (yw3285@columbia.edu)/RA/FER/FER/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
```

Notice that images and their annotations are stored in 6 main folders and 12 subfolders, we need to firstly extract paths to reach them.

```
path = '../data'
# Use paste to get 6 paths, such as: "../data/01-04/ImageMarking1"
folderpath.anno <- paste(path,
                         paste(list.files(pattern = "^.{2}-.{2}$",path = path),
                         paste("ImageMarking", 1:6, sep = ""), sep = "/"),
                         sep = "/")
folderpath.anno
```

```
## [1] "../data/01-04/ImageMarking1" "../data/05-08/ImageMarking2"
## [3] "../data/09-12/ImageMarking3" "../data/13-16/ImageMarking4"
## [5] "../data/17-20/ImageMarking5" "../data/21-26/ImageMarking6"
```

```
# Within each folder, extract file names, return a list of 6
filename <- lapply(folderpath.anno, list.files, pattern = "\\.mat$")

# Define a function:
# input: "a", "b"
# return: "b/a"
paste.rev <- function(a,b){
  return(paste(b, a, sep = "/"))
}

# Paste folder paths and filenames
annotation_path <- c()
```

```
for (i in 1:length(filename)){
  annotation_path <- c(annotation_path,
                       unlist(lapply(filename[i], paste.rev, folderpath.anno[i])))
}

annotation_path[1:10]
```

```
##  [1] "../data/01-04/ImageMarking1/01_110_2527.mat"
##  [2] "../data/01-04/ImageMarking1/01_111_2624.mat"
##  [3] "../data/01-04/ImageMarking1/01_113_2715.mat"
##  [4] "../data/01-04/ImageMarking1/01_114_2824.mat"
##  [5] "../data/01-04/ImageMarking1/01_115_2920.mat"
##  [6] "../data/01-04/ImageMarking1/01_116_3013.mat"
##  [7] "../data/01-04/ImageMarking1/01_117_3123.mat"
##  [8] "../data/01-04/ImageMarking1/01_118_3242.mat"
##  [9] "../data/01-04/ImageMarking1/01_119_3348.mat"
## [10] "../data/01-04/ImageMarking1/01_121_3473.mat"
```

```
#image paths
tmp <- gsub(annotation_path, pattern = 'ImageMarking', replacement = 'Images')
image_path <- gsub(tmp, pattern = 'mat',replacement = 'jpg')
image_path[1:10]
```

```
##  [1] "../data/01-04/Images1/01_110_2527.jpg"
##  [2] "../data/01-04/Images1/01_111_2624.jpg"
##  [3] "../data/01-04/Images1/01_113_2715.jpg"
##  [4] "../data/01-04/Images1/01_114_2824.jpg"
##  [5] "../data/01-04/Images1/01_115_2920.jpg"
##  [6] "../data/01-04/Images1/01_116_3013.jpg"
##  [7] "../data/01-04/Images1/01_117_3123.jpg"
##  [8] "../data/01-04/Images1/01_118_3242.jpg"
##  [9] "../data/01-04/Images1/01_119_3348.jpg"
## [10] "../data/01-04/Images1/01_121_3473.jpg"
```

Summary

```
#Infomation table
categoryID <- substr(annotation_path, 29,30)
categoryID[categoryID<10] <- substr(categoryID[categoryID<10],2,2)
identity <- substr(annotation_path, 32, 34)
emotion_table <- read_xls("../data/AU_annotation_all_subjects.xls")

info <- data.frame(identity, annotation_path, image_path, categoryID = as.numeric(categoryID)) %>%
  left_join(emotion_table, by = c('categoryID' = 'idx')) %>% na.omit()
info <- info %>% mutate(Index = 1:nrow(info))

head(info)
```

```
##   identity                            annotation_path
## 1      110 ../data/01-04/ImageMarking1/01_110_2527.mat
## 2      111 ../data/01-04/ImageMarking1/01_111_2624.mat
## 3      113 ../data/01-04/ImageMarking1/01_113_2715.mat
## 4      114 ../data/01-04/ImageMarking1/01_114_2824.mat
## 5      115 ../data/01-04/ImageMarking1/01_115_2920.mat
## 6      116 ../data/01-04/ImageMarking1/01_116_3013.mat
##                              image_path categoryID emotion cat Index
```

```
## 1 ../data/01-04/Images1/01_110_2527.jpg          1     Neutral     1
## 2 ../data/01-04/Images1/01_111_2624.jpg          1     Neutral     2
## 3 ../data/01-04/Images1/01_113_2715.jpg          1     Neutral     3
## 4 ../data/01-04/Images1/01_114_2824.jpg          1     Neutral     4
## 5 ../data/01-04/Images1/01_115_2920.jpg          1     Neutral     5
## 6 ../data/01-04/Images1/01_116_3013.jpg          1     Neutral     6
```

**Step 1: set up controls for evaluation experiments.**

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

- (T/F) cross-validation on the training set
- (number) K, the number of CV folds
- (T/F) process features for training set
- (T/F) run evaluation on an independent test set
- (T/F) process features for test set

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

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications. In this example, we use SVM with different `gamma` and `cost`. In the following chunk, we list, in a matrix, setups corresponding to models that we will compare. In your project, you might compare very different classifiers. You can assign them numerical IDs and labels specific to your project.

```
gamma = 10^(-5:-1)
cost = 10^(-1:2)
model_values <- expand.grid(gamma,cost)
model_labels = paste(paste("SVM with gamma =", model_values[,1]),
                     paste(", cost =", model_values[,2]),
                     sep = "")
```

**Step 2: import data and train-test split**

```
#train-test split
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)
```

```
Image_list <- lapply(image_path[1:10], EBImage::readImage)
Image_list <- lapply(Image_list, imageData)
```

```
display(Image(Image_list[[1]], colormode = "Color"))
```

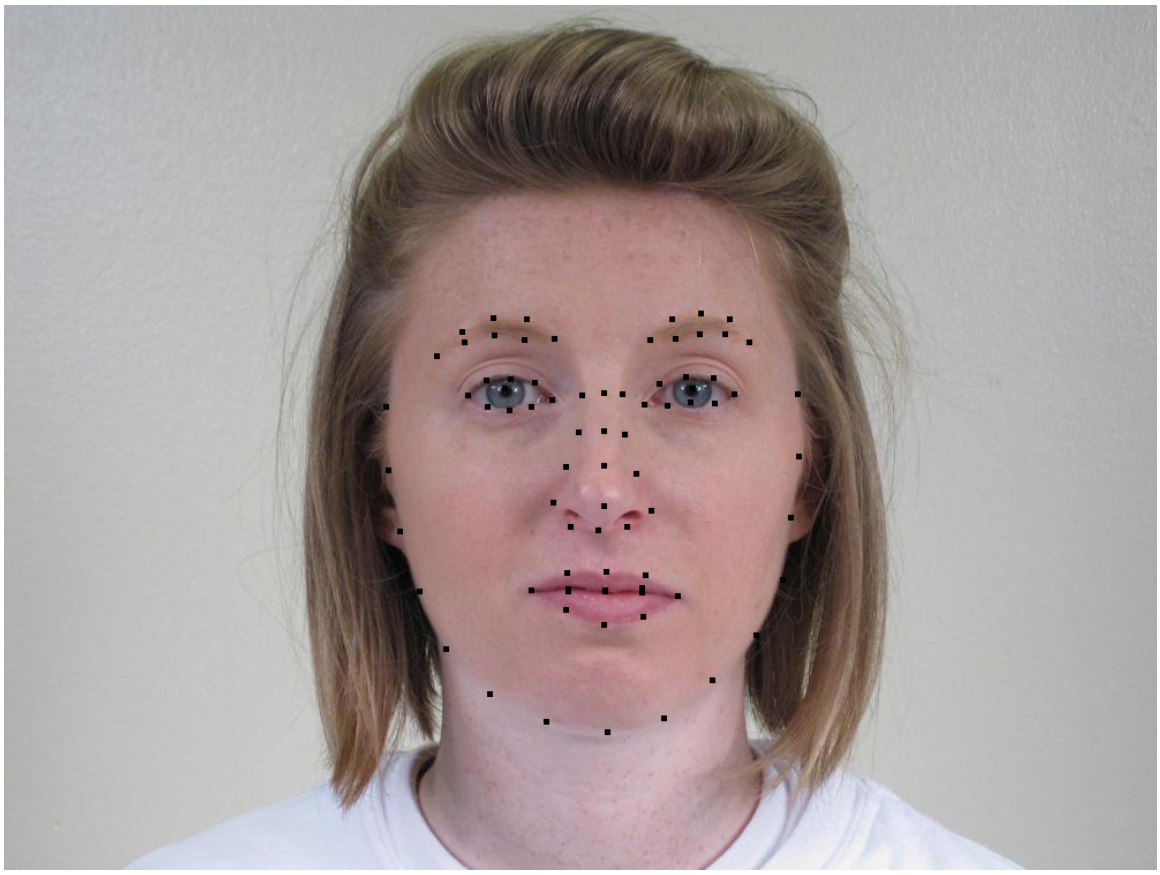
```
readMat.matrix <- function(path){
     return(round(readMat(path)[[1]],0))
}

#load fiducial points
fiducial_pt_list <- lapply(annotation_path[1:10], readMat.matrix)
```

```
#display image and fiducial points
Image_list_copy <- Image_list

display_fid_pt <- function(idx, pt_size = 2, pt_col = 1){
for (i in 1:nrow(fiducial_pt_list[[idx]])){
   print(i)
  Image_list_copy[[idx]][
    (fiducial_pt_list[[idx]][i,1]-pt_size):(fiducial_pt_list[[idx]][i,1]+pt_size),
    (fiducial_pt_list[[idx]][i,2]-pt_size):(fiducial_pt_list[[idx]][i,2]+pt_size),] <- pt_col
}
  display(Image(Image_list_copy[[idx]], colormode = 'Color'))
}

display_fid_pt(1, 2, 0)
```

```
## [1] 1
## [1] 2
```

```
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
```

```
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
```



### Step 3: construct features and responses

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features and responses that are required by all the models you are going to evaluate later. + `feature.R` + Input: list of images or fiducial point + Output: an RData file that contains extracted features and corresponding responses

```
fiducial_pt_list <- lapply(annotation_path, readMat.matrix)
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.train){
  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")
```

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

Call the train model and test model from library.

`train.R` and `test.R` should be wrappers for all your model training steps and your classification/prediction steps. + `train.R` + Input: a data frame containing features and labels and a parameter list. + Output:a trained model + `test.R` + Input: the fitted classification model using training data and processed features from testing images + Input: an R object that contains a trained classifier. + Output: training model specification

```
source("../lib/train.R")
source("../lib/test.R")
```

### Model selection with cross-validation

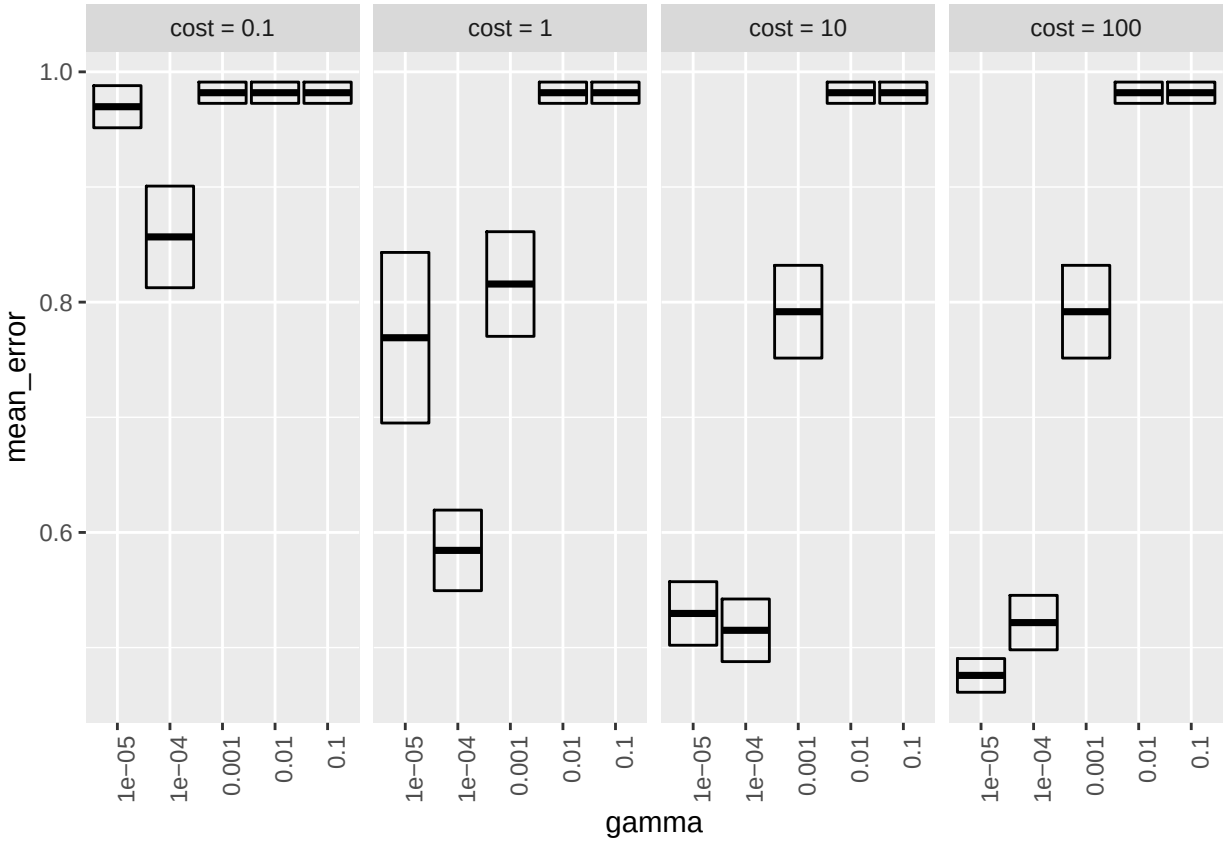
- Do model selection by choosing among different values of training model parameters, such as gamma and cost.

```
source("../lib/cross_validation.R")
if(run.cv){
  err_cv <- array(dim=c(nrow(model_values), 2))
  for(k in 1:nrow(model_values)){
    cat("k=", k, "\n")
    err_cv[k,] <- cv.function(dat_train, K, model_values[k,1], model_values[k,2])
  save(err_cv, file="../output/err_cv.RData")
  }
}
```

Visualize cross-validation results.

```
if(run.cv){
  load("../output/err_cv.RData")
```

```
  colnames(model_values) <- c("gamma", "cost")
  colnames(err_cv) <- c("mean_error", "sd_error")
  data.frame(cbind(model_values, err_cv)) %>%
    mutate(gamma = as.factor(gamma)) %>%
    mutate(cost = paste("cost =",as.factor(cost))) %>%
    ggplot(aes(x = gamma, y = mean_error,
               ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() +
    facet_grid(~cost) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
}
```




- Choose the "best" parameter value

```
model_best=model_values[1,]
if(run.cv){
  model_best <- model_values[which.min(err_cv[,1]),]
}
par_best <- list(gamma = model_best$gamma, cost = model_best$cost)
```

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

```
tm_train=NA
tm_train <- system.time(fit_train <- train(dat_train, par_best))
save(fit_train, file="../output/fit_train.RData")
```

**Step 5: Run test on test images**

```
tm_test=NA
if(run.test){
  load(file="../output/fit_train.RData")
  tm_test <- system.time(pred <- test(fit_train, dat_test))
}
```

- evaluation

```
accu <- mean(dat_test$categoryID == pred)
cat("The accuracy of model:", model_labels[which.min(err_cv[,1])], "is", accu, ".\n")
```

```
## The accuracy of model: SVM with gamma = 1e-05, cost = 100 is 0.5753968 .
```

```
library(caret)
confusionMatrix(pred, dat_test$categoryID)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  5  6  7  8  9 10 11 12 13 14 15 16 18 21 22 23 24 25
##         1  41  0  3  0  0  0  0  0  0  0  1  0  1  0  0  0  3  1  2  0  0
##         2   0 38  0  1  0  0  0  1  0  4  0  0  0  0  0  0  0  0  0  0  1
##         3   3  0 22  8  1  0  1  1  0  0  6  1  1  0  0  0  5  2  2  0  0
##         5   0  0  2 28  0  0  1  0  0  1  2  0  0  1  2  1  0  0  2  1  0
##         6   1  0  2  0 21  0  1  0  0  0  8  2  1  9  0  0  1  0  0  0  0
##         7   1  0  0  0  0 27  0  2  0  0  0  0  0  0  0  2  0  0  1  3  3
##         8   0  0  0  0  4  0 25  0  0  1  4  4  1  3  0  0  0  2  0  0  0
##         9   0  0  1  1  0  2  0 29  1  0  0  0  0  0  1  1  2  0  1  1  0
##         10  0  4  0  0  0  0  0  1 35  1  0  0  0  0  0  0  0  0  0  1  3
##         11  0  1  0  3  0  0  1  0  0 31  0  0  0  0  0  1  0  0  0  0  1
##         12  1  0  4  3  2  0  4  0  0  1 24  7  2  2  0  0  1  1  2  0  0
##         13  0  0  0  0  1  0  4  0  0  0  4 20  8  3  0  0  0  0  1  0  0
##         14  0  0  1  0  2  0  4  0  0  1  1  6 14  9  2  0  0  1  0  0  0
##         15  0  0  0  1  7  0  0  0  0  0  3  5  9 20  1  1  0  1  5  0  0
##         16  0  0  0  4  0  0  2  1  0  0  0  0  0  1 32  6  0  1  3  6  1
##         18  0  0  0  1  0  1  0  4  0  0  0  0  0  0  6 20  0  0  0  0  1
##         21  0  0  2  2  0  0  0  2  0  0  1  0  0  0  0  0 28  5  1  0  0
##         22  1  0  1  0  0  0  4  0  0  0  1  0  1  0  0  0  2 31  1  1  1
##         23  1  0  2  3  1  1  1  1  0  0  2  1  2  2  2  0  1  1 25  3  0
##         24  0  0  0  0  0  0  2  2  1  0  0  0  0  0  3  3  0  2  0 16  1
##         25  0  0  0  0  0  1  0  5  3  0  0  0  0  0  1  0  0  0  0  0 18
##         26  0  0  1  1  0  9  0  1  0  0  0  0  0  0  0  2  0  0  1  1 18
##           Reference
## Prediction 26
##         1   1
##         2   0
##         3   0
##         5   0
##         6   0
##         7   4
##         8   0
##         9   2
##         10  0
```

```
##         11  0
##         12  0
##         13  0
##         14  0
##         15  0
##         16  1
##         18  1
##         21  3
##         22  1
##         23  0
##         24  3
##         25  9
##         26 35
##
## Overall Statistics
##
##                Accuracy : 0.5754
##                  95% CI : (0.5442, 0.6061)
##     No Information Rate : 0.0595
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5548
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
## Sensitivity           0.83673  0.88372  0.53659  0.50000  0.53846  0.65854
## Specificity           0.98749  0.99275  0.96794  0.98634  0.97420  0.98345
## Pos Pred Value        0.77358  0.84444  0.41509  0.68293  0.45652  0.62791
## Neg Pred Value        0.99162  0.99481  0.98010  0.97104  0.98129  0.98549
## Prevalence            0.04861  0.04266  0.04067  0.05556  0.03869  0.04067
## Detection Rate        0.04067  0.03770  0.02183  0.02778  0.02083  0.02679
## Detection Prevalence  0.05258  0.04464  0.05258  0.04067  0.04563  0.04266
## Balanced Accuracy     0.91211  0.93823  0.75226  0.74317  0.75633  0.82100
##                      Class: 8 Class: 9 Class: 10 Class: 11 Class: 12
## Sensitivity           0.50000  0.58000   0.87500   0.77500   0.42105
## Specificity           0.98017  0.98643   0.98967   0.99277   0.96845
## Pos Pred Value        0.56818  0.69048   0.77778   0.81579   0.44444
## Neg Pred Value        0.97407  0.97826   0.99481   0.99072   0.96541
## Prevalence            0.04960  0.04960   0.03968   0.03968   0.05655
## Detection Rate        0.02480  0.02877   0.03472   0.03075   0.02381
## Detection Prevalence  0.04365  0.04167   0.04464   0.03770   0.05357
## Balanced Accuracy     0.74008  0.78322   0.93233   0.88388   0.69475
##                      Class: 13 Class: 14 Class: 15 Class: 16 Class: 18
## Sensitivity            0.43478   0.35000   0.40000   0.64000   0.54054
## Specificity            0.97817   0.97211   0.96555   0.97286   0.98558
## Pos Pred Value         0.48780   0.34146   0.37736   0.55172   0.58824
## Neg Pred Value         0.97311   0.97311   0.96859   0.98105   0.98255
## Prevalence             0.04563   0.03968   0.04960   0.04960   0.03671
## Detection Rate         0.01984   0.01389   0.01984   0.03175   0.01984
## Detection Prevalence   0.04067   0.04067   0.05258   0.05754   0.03373
## Balanced Accuracy      0.70648   0.66105   0.68278   0.80643   0.76306
```

```
##                      Class: 21 Class: 22 Class: 23 Class: 24 Class: 25
## Sensitivity            0.65116   0.64583   0.53191   0.48485   0.37500
## Specificity            0.98342   0.98542   0.97503   0.98256   0.98021
## Pos Pred Value         0.63636   0.68889   0.51020   0.48485   0.48649
## Neg Pred Value         0.98444   0.98235   0.97706   0.98256   0.96910
## Prevalence             0.04266   0.04762   0.04663   0.03274   0.04762
## Detection Rate         0.02778   0.03075   0.02480   0.01587   0.01786
## Detection Prevalence   0.04365   0.04464   0.04861   0.03274   0.03671
## Balanced Accuracy      0.81729   0.81563   0.75347   0.73371   0.67760
##                      Class: 26
## Sensitivity            0.58333
## Specificity            0.96414
## Pos Pred Value         0.50725
## Neg Pred Value         0.97338
## Prevalence             0.05952
## Detection Rate         0.03472
## Detection Prevalence   0.06845
## Balanced Accuracy      0.77373
```

### Summarize Running Time

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

```
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
```

```
## Time for constructing training features= 2.012 s
```

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

```
## Time for constructing testing features= 0.339 s
```

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

```
## Time for training model= 358.988 s
```

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

```
## Time for testing model= 33.815 s
```