Project 3: Main Script

Ka Heng (Helen) Lo

March 24, 2017

This file firstly details the model selection process for an advanced binary classification model, that takes in images and classifies them as either a labradoodle ("1") or fried chicken ("0") - simply referred to as the Advanced Model. Secondly, it runs evaluation experiments to compare the Baseline Model - gradient boosting model (gbm) with parameters tuned on the original SIFT features - with our Advanced Model.

#Load (and install if necessary) required packages  
if(!require("EBImage")){  
 source("https://bioconductor.org/biocLite.R")  
 biocLite("EBImage")  
}

## Loading required package: EBImage

needed <- setdiff(c("gbm", "data.table", "e1071", "class", "adabag","caret"),  
 rownames(installed.packages()))  
if (length(needed) > 0 ){  
 install.packages(needed)  
}  
library(EBImage)  
library(gbm)

## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.1

library(data.table)

##   
## Attaching package: 'data.table'

## The following object is masked from 'package:EBImage':  
##   
## transpose

library(e1071)  
library(class)  
library(adabag)

## Loading required package: rpart

## Loading required package: mlbench

## Loading required package: caret

## Loading required package: ggplot2

##   
## Attaching package: 'caret'

## The following object is masked from 'package:survival':  
##   
## cluster

library(caret)

### Import class labels for training images

We code labradoodles as "1" and fried chicken as "0" for binary classification.

label\_train <- read.csv("../data/train/labels.csv")  
label\_train <- as.matrix(label\_train)[,1]   
#some of our models require labels to be numeric in [0,1], so we keep the labels as 0's and 1's  
#for any model that requires labels to be factors we pass the labels to the tune or train function  
#specific to the model as numeric values and coerce them inside the function to be factors

## Selecting the Advanced Model:

### Model selection with provided SIFT features of training images

Import the SIFT features of the set of 2000 training images

train\_sift <- read.csv("../data/train/sift\_features.csv")  
train\_sift <- t(as.matrix(train\_sift))

#### Tune parameters of each model via cross-validation

Set up control for tuning models under consideration based on the provided SIFT features of the training images.  
+ (T/F) tune models under consideration based on provided SIFT features

###### T/F to reproduce the tuning process for all models considered ######  
run.tune.sift = FALSE

* Parameter-tuning process:

#get summary tables of best tuned parameters for each model considered, i.e. lowest cv error  
#6 columns: "Model", "Best\_Param\_1", "Best\_Param\_2", "Best\_Param\_3",   
# "Best\_Error", "Training\_Time"  
# - NA values for Best\_Param\_2 and/or Best\_Param\_3 if model has less than 3 params  
############################################################################################  
  
  
#source("../lib/xgboost.R")  
#source("../lib/svm.R")  
#source("../lib/knn.R")  
#source("../lib/AdaBag\_AdaBoost.R")  
#source("../lib/blgbm.R")  
  
if(run.tune.sift) {  
 source("../lib/xgboost.R")  
 source("../lib/svm.R")  
 source("../lib/knn.R")  
 source("../lib/AdaBag\_AdaBoost.R")  
 source("../lib/blgbm.R")  
   
 summary.xgb <- tune.xgb(train\_sift,label\_train)  
 #save(summary.xgb, file="output/summary\_best\_xgb.RData")  
  
 summary.svm.lin <- tune.svm.lin(train\_sift,label\_train)  
 #save(summary.svm.lin, file="output/summary\_best\_svm\_lin.Rdata")  
  
 summary.svm.rad <- tune.svm.rad(train\_sift,label\_train)  
 #save(summary.svm.rad,file="output/summary\_best\_svm\_rad.Rdata")  
  
 summary.knn <- tune.knn(train\_sift,label\_train)  
 #save(summary.knn, file="output/summary\_best\_knn.Rdata")  
  
 summary.AdaBag <- tune.AdaBag(train\_sift,label\_train)  
 #save(summary.AdaBag,file="output/summary\_best\_AdaBag2.Rdata")  
  
 summary.AdaBoost.M1 <- tune.AdaBoost.M1(train\_sift,label\_train)  
 #save(summary.AdaBoost.M1,file="output/summary\_best\_AdaBoost.M1.Rdata")  
  
 summary.AdaBoost\_SAMME <- tune.AdaBoost\_SAMME(train\_sift,label\_train)  
 #save(summary.AdaBoost\_SAMME,file="output/summary\_best\_AdaBoost\_SAMME.Rdata")  
   
 summary.bl<- tune.bl(train\_sift, label\_train)  
 #save(summary.bl,file="output/summary\_best\_blgbm.RData")  
}

#### Summary Table: parameter-tuning process on original SIFT features

Summary table of *best* models (i.e. models with *best* parameters) tuned on original SIFT features of the training set via cross-validation, sorted by CV Error.

library(data.table)  
#load already produced summary tables for best tuned parameters for each model considered  
load("../output/summary\_best\_xgb.RData")  
load("../output/summary\_best\_svm\_lin.Rdata")  
load("../output/summary\_best\_svm\_rad.Rdata")  
load("../output/summary\_best\_knn.Rdata")  
  
load("../output/summary\_best\_AdaBag2.Rdata")   
  
load("../output/summary\_best\_AdaBoost.M1.Rdata")  
load("../output/summary\_best\_AdaBoost\_SAMME.Rdata")  
  
load("../output/summary\_best\_blgbm.Rdata")  
  
#6 columns: "Model", "Best\_Param\_1", "Best\_Param\_2", "Best\_Param\_3",   
# "Best\_Error", "Training\_Time"  
# - NA values for Best\_Param\_2 and/or Best\_Param\_3 if model has less than 3 params  
summary1 <- rbind(summary.xgb, summary.svm.lin,  
 summary.svm.rad,   
 summary.AdaBag2, summary.AdaBoost.M1,  
 summary.AdaBoost\_SAMME,  
 summary.knn,summary.bl)   
  
#sort table by Best\_Error in ascending order  
summary1 <- summary1[order(summary1$Best\_Error)]  
  
#add column for feature set models are tuned on  
(summary1 <- data.table(summary1[,1], Features = rep("Original SIFT",8), summary1[,2:6]))

## Model Features Best\_Param\_1 Best\_Param\_2  
## 1: XGBoost Original SIFT max\_depth = 5 eta = 0.5  
## 2: BL GBM Original SIFT shrinkage = 0.16 ntrees= 64  
## 3: SVM with linear kernel Original SIFT cost = 5000 NA  
## 4: AdaBag Original SIFT mfinal = 100 NA  
## 5: AdaBoost.M1 Original SIFT mfinal = 15 NA  
## 6: AdaBoost\_SAMME Original SIFT mfinal = 15 NA  
## 7: KNN Original SIFT k = 1 NA  
## 8: SVM with Radial Kernel Original SIFT cost = 500 gamma = 2  
## Best\_Param\_3 Best\_Error Training\_Time  
## 1: nrounds = 169 5.267342e-05 49.148 s  
## 2: NA 2.365000e-01 15.234 s  
## 3: NA 2.520000e-01 73.452 s  
## 4: NA 2.915000e-01 1780.513 s  
## 5: NA 3.095000e-01 239.031 s  
## 6: NA 3.190000e-01 249.718 s  
## 7: NA 3.735000e-01 NA  
## 8: NA 4.020000e-01 73.638 s

save(summary1,file="../output/summary\_best\_models1.Rdata")

### Model selection with new visual features

#### Import sets of new visual features

#import the three csv files - the three new sets of visual features  
new\_train\_feat1 <- read.csv("../data/train/sift\_features\_resize+adaptive.csv")  
new\_train\_feat1 <- t(as.matrix(new\_train\_feat1))  
  
new\_train\_feat2 <- read.csv("../data/train/sift\_features\_resize.csv")  
new\_train\_feat2 <- t(as.matrix(new\_train\_feat2))  
  
new\_train\_feat3 <- read.csv("../data/train/sift\_features\_adaptive.csv")  
new\_train\_feat3 <- t(as.matrix(new\_train\_feat3))

#### Tune parameters of each model via cross-validation

* Set up control for tuning models under consideration based on three new sets of features of the training images.
* (T/F) tune models under consideration based on three new sets of visual features

###### T/F to reproduce the tuning process for all models considered ######  
run.tune.new = FALSE

* Parameter-tuning process:

if (run.tune.new){  
 source("../lib/xgboost.R")  
 source("../lib/blgbm.R")  
   
 ###\*\*\*\*\*\*\* Tune for xgBoost  
 # 1)tune on the first set of new visual features for the training images  
 summary.xgb.new1 <- tune.xgb(new\_train\_feat1,label\_train)  
 #save(summary.xgb.new1,file="../output/summary\_best\_xgb\_new1.Rdata")  
   
 # 2)tune on the second set of new visual features for the training images  
 summary.xgb.new2 <- tune.xgb(new\_train\_feat2,label\_train)  
 #save(summary.xgb.new2,file="../output/summary\_best\_xgb\_new2.Rdata")  
   
 # 3)tune on the third set of new visual features for the training images  
 summary.xgb.new3 <- tune.xgb(new\_train\_feat3,label\_train)  
 #save(summary.xgb.new3,file="../output/summary\_best\_xgb\_new3.Rdata")  
   
   
 ###\*\*\*\*\*\*\* Tune for gbm   
 # 1)tune on the first set of new visual features for the training images  
 summary.gbm.new1 <- bl.tune(new\_train\_feat1,label\_train)  
 #save(summary.gbm.new1,file="../output/summary\_best\_gbm\_new1.Rdata")  
  
 # 2)tune on the second set of new visual features for the training images  
 summary.gbm.new2 <- bl.tune(new\_train\_feat2,label\_train)  
 #save(summary.gbm.new2,file="../output/summary\_best\_gbm\_new2.Rdata")  
   
 # 3)tune on the third set of new visual features for the training images  
 summary.gbm.new3 <- bl.tune(new\_train\_feat3,label\_train)  
 #save(summary.gbm.new3,file="../output/summary\_best\_gbm\_new3.Rdata")  
   
}

#### Summary Table: parameter-tuning process on new features

#load Rdata files  
load("../output/summary\_best\_xgb\_new1.Rdata")  
load("../output/summary\_best\_xgb\_new2.Rdata")  
load("../output/summary\_best\_xgb\_new3.Rdata")  
load("../output/summary\_best\_gbm\_new1.Rdata")  
load("../output/summary\_best\_gbm\_new2.Rdata")  
load("../output/summary\_best\_gbm\_new3.Rdata")  
  
summary2 <- rbind(summary.xgb.new1, summary.xgb.new2,  
 summary.xgb.new3, summary.gbm.new1,  
 summary.gbm.new2, summary.gbm.new3)   
#sort table by Best\_Error in ascending order  
summary2 <- summary2[order(summary2$Best\_Error)]  
  
#add column for feature set models are tuned on  
(summary2 <- data.table(summary2[,1], Features = c("SIFT- resize+adaptive","SIFT- resize",  
 "SIFT- adaptive", "SIFT- resize+adaptive",  
 "SIFT- resize", "SIFT- adaptive"),  
 summary2[,2:6]))

## Model Features Best\_Param\_1 Best\_Param\_2  
## 1: XGBoost SIFT- resize+adaptive max\_depth = 7 eta = 0.5  
## 2: BL GBM SIFT- resize shrinkage = 0.21 ntrees= 48  
## 3: XGBoost SIFT- adaptive max\_depth = 7 eta = 0.2  
## 4: BL GBM SIFT- resize+adaptive shrinkage = 0.16 ntrees= 127  
## 5: BL GBM SIFT- resize shrinkage = 0.21 ntrees= 46  
## 6: XGBoost SIFT- adaptive max\_depth = 7 eta = 0.2  
## Best\_Param\_3 Best\_Error Training\_Time  
## 1: nrounds = 45 0.1890 19.032 s  
## 2: NA 0.1900 11.324 s  
## 3: nrounds = 62 0.1905 26.374 s  
## 4: NA 0.1935 23.067 s  
## 5: NA 0.2470 12.802 s  
## 6: nrounds = 44 0.2575 20.311 s

save(summary2,file="../output/summary\_best\_models2.Rdata")

### Convolutional Neural Network (CNN)

#### Tune CNN model on set of raw training images via cross-validation

* Set up control for tuning CNN on set of raw images
* (T/F) tune CNN

###### T/F to reproduce the tuning process for all models considered ######  
run.tune.cnn = FALSE

* Parameter-tuning process

if (run.tune.cnn) {  
 img\_train\_dir <- "../data/train/raw\_images"  
 source("../lib/CNN.R")  
 source("../lib/CNNcv.R")  
   
 #first resize the raw images and then split into test set(200) and train set(1800)  
 data\_list <- to.resize.split(img\_train\_dir,label\_train)  
   
   
 #cross-validation on the resized train set (1800 images)  
 iter <- seq(60,100,by=10) #iter is the range of the tune parameters  
 #get best num.rounds  
 best\_num.rounds <- CNNcv(data\_list$train\_data,iter)  
   
 #retrain data (1800 images) ; get run.time; and test error using test set(200 images)  
 cnn.output <- CNN(data\_list$train\_data,data\_list$test\_data)  
   
 summary.cnn <- data.table(Model = "CNN", Features = NA,  
 Best\_Param\_1 = paste("num.rounds =",best\_num.rounds),  
 Best\_Param\_2 = NA,  
 Best\_Param\_3 = NA,  
 Best\_Error = cnn.output$test\_err,  
 Training\_Time = paste(cnn.output$train\_time, "s"))  
 #save(summary.cnn, file="output/summary\_best\_cnn.RData")  
  
}

### Selecting the *best* Advanced Model

#### Summary Table of best models tuned on various feature sets (original SIFT & new) + CNN

#load Rdata files   
load("../output/summary\_best\_models1.Rdata")  
load("../output/summary\_best\_models2.Rdata")  
  
load("../output/summary\_best\_cnn.Rdata")  
  
summary3 <- rbind(summary1,summary2,summary.cnn)  
(summary3 <- summary3[order(summary3$Best\_Error)])

## Model Features Best\_Param\_1  
## 1: XGBoost Original SIFT max\_depth = 5  
## 2: CNN NA num.round = 60  
## 3: XGBoost SIFT- resize+adaptive max\_depth = 7  
## 4: BL GBM SIFT- resize shrinkage = 0.21  
## 5: XGBoost SIFT- adaptive max\_depth = 7  
## 6: BL GBM SIFT- resize+adaptive shrinkage = 0.16  
## 7: BL GBM Original SIFT shrinkage = 0.16  
## 8: BL GBM SIFT- resize shrinkage = 0.21  
## 9: SVM with linear kernel Original SIFT cost = 5000  
## 10: XGBoost SIFT- adaptive max\_depth = 7  
## 11: AdaBag Original SIFT mfinal = 100  
## 12: AdaBoost.M1 Original SIFT mfinal = 15  
## 13: AdaBoost\_SAMME Original SIFT mfinal = 15  
## 14: KNN Original SIFT k = 1  
## 15: SVM with Radial Kernel Original SIFT cost = 500  
## Best\_Param\_2 Best\_Param\_3 Best\_Error Training\_Time  
## 1: eta = 0.5 nrounds = 169 5.267342e-05 49.148 s  
## 2: NA NA 1.355000e-01 321.5955 s  
## 3: eta = 0.5 nrounds = 45 1.890000e-01 19.032 s  
## 4: ntrees= 48 NA 1.900000e-01 11.324 s  
## 5: eta = 0.2 nrounds = 62 1.905000e-01 26.374 s  
## 6: ntrees= 127 NA 1.935000e-01 23.067 s  
## 7: ntrees= 64 NA 2.365000e-01 15.234 s  
## 8: ntrees= 46 NA 2.470000e-01 12.802 s  
## 9: NA NA 2.520000e-01 73.452 s  
## 10: eta = 0.2 nrounds = 44 2.575000e-01 20.311 s  
## 11: NA NA 2.915000e-01 1780.513 s  
## 12: NA NA 3.095000e-01 239.031 s  
## 13: NA NA 3.190000e-01 249.718 s  
## 14: NA NA 3.735000e-01 NA  
## 15: gamma = 2 NA 4.020000e-01 73.638 s

save(summary3,file="../output/summary\_best\_models3.Rdata")  
save(summary3,file="../output/summary\_best\_models3.Rdata")

In the summary table, the models under consideration are ranked by best (min) error. The Baseline Model (GBM with original SIFT) is ranked 7th with cross-validation estimated prediction error of .2365 and training time of 15.234 seconds. Model 1 in the table (XGBoost with original SIFT) has the lowest estimated prediction error of 5.267342e-05 (<<.2365), but it has a higher training time of 49.148 seconds (>15.234 seconds). Model 2 in the table (CNN with 1800 of the 2000 raw images) has an test error (testing with the remaining 200 of the 2000 raw images) of .1355, but it has a really large training time of 321.5955 seconds (>>15.234 seconds). Model 3 in the table (XGBoost with SIFT- resize+adaptive) has a decently low cross-validation estimated prediction error of .189 (<.2365) and a decently low training time of 19.032 seconds (only slightly larger, but comparable to 15.234 seconds). Thus, we chose Model 3 (XGBoost with SIFT- resize+adaptive) to be our advanced model, as it seems the best compromise between a low estimated prediction error and a small training time.

## Comparing the Baseline Model and the Advanced Model

### Set up controls for evaluation experiments: Baseline Model vs. Advanced Model

* (T/F) train Baseline Model and Advanced Model
* (T/F) process features for training set
* (T/F) run evaluation on an independent test set for both Baseline Model and Advanced Model
* (T/F) process features for test set

run.train=TRUE # train 'best' model   
run.feature.train=TRUE # process features for training set  
run.test=FALSE # run evaluation on an independent test set  
run.feature.test=FALSE # process features for test set

### Import given SIFT features new visual features of the test set

We construct the new visual features outside of R/Rstudio, so we just import the .csv files with the new feature data.

#Recall for the training set we imported the .csv files before  
if (run.feature.train){  
 #dat\_train\_sift <- read.csv("../data/train/sift\_features.csv")  
 #dat\_train\_sift <- t(as.matrix(dat\_train\_sift))  
 dat\_train\_sift <- train\_sift  
 save(dat\_train\_sift,file="../output/dat\_train\_sift.Rdata")  
   
 #dat\_train\_new <- read.csv("../data/train/sift\_features\_resize+adaptive.csv")  
 #dat\_train\_new <- t(as.matrix(dat\_train\_new))  
 dat\_train\_new <- new\_train\_feat1 #we selected new feature set 1: SIFT- resize+adaptive  
 save(dat\_train\_new,file="../output/dat\_train\_new.Rdata")  
}  
  
##Now for the test set:   
if (run.feature.test){  
 #Import the provide SIFT features of the test set  
 dat\_test\_sift <- read.csv("../data/test/sift\_features\_test.csv")  
 dat\_test\_sift <- t(as.matrix(dat\_test\_sift))  
 save(dat\_test\_sift,file="../output/dat\_test\_sift.Rdata")  
 #Import the set of new visual features (constructed outside of R)  
 dat\_test\_new <- read.csv("../data/test/sift\_features\_resize+adaptive.csv") # \*\*\*or other file name  
 dat\_test\_new <- t(as.matrix(dat\_test\_new))  
 save(dat\_test\_new,file="../output/dat\_test\_new.Rdata")  
}  
  
#tm\_feature\_test = NA

### Train a classification model with training images

train.R and test.R should be wrappers for all your model training steps and your classification/prediction steps. + train.R + Input: an R object that contains processed training set features. + Input: an R object of training sample labels. + Output: an RData file that contains trained classifiers in the forms of R objects: models/settings/links to external trained configurations. + test.R + Input: a path that points to the test set features. + Input: an R object that contains a trained classifiers. + Output: an R object of class label predictions on the test set. If there are multiple classifiers under evaluation, there should be multiple sets of label predictions.

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

### Train Baseline Model and Advanced Model

Here we call the train() function in train.R. We fit the entire set of constructed visual features of the training images to the Baseline Model and the Advanced Model.

#Here, we call train() from train.R  
# -note that train() returns a list of two model objects: list(baseline\_fit,advanced\_fit)  
tm\_train=NA ; tm\_train\_base=NA ; tm\_train\_adv=NA  
if(run.train){  
 load(file="../output/dat\_train\_sift.Rdata")  
 load(file="../output/dat\_train\_new.Rdata")  
 tm\_train <- system.time(fit\_train <- train(dat\_train\_base = dat\_train\_sift,   
 dat\_train\_adv = dat\_train\_new,   
 label\_train=label\_train))  
 save(fit\_train, file="../output/fit\_train.Rdata")  
   
 #individual times for training base model and advanced model   
 tm\_train\_base <- system.time(fit\_train\_base <- train(dat\_train\_base = dat\_train\_sift,  
 label\_train=label\_train, model="base"))  
 tm\_train\_adv <- system.time(fit\_train\_adv <- train(dat\_train\_adv = dat\_train\_new,  
 label\_train=label\_train,model="advanced"))  
}

## Loading required package: xgboost

### Make predictions

Here we call the test() function in test.R. Feed the Baseline Model and the Advanced Model with the completely holdout testing data.

#################################################################################################  
#to test the test() function, get predictions on training data for base model and advanced model  
load(file="../output/fit\_train.Rdata")  
load(file="../output/dat\_train\_sift.Rdata")  
load(file="../output/dat\_train\_new.Rdata")  
(pred\_train <- test(fit\_train = fit\_train, dat\_test\_base = dat\_train\_sift,  
 dat\_test\_adv = dat\_train\_new))

## $baseline\_pred  
## [1] 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0  
## [35] 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0  
## [69] 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0  
## [103] 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0  
## [137] 0 0 0 1 1 0 1 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0  
## [171] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1  
## [205] 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0  
## [239] 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0  
## [273] 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0  
## [307] 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0  
## [341] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 1  
## [375] 0 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1  
## [409] 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0  
## [443] 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0  
## [477] 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0  
## [511] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0  
## [545] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0  
## [579] 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1  
## [613] 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0  
## [647] 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0  
## [681] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0  
## [715] 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0  
## [749] 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 1  
## [783] 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0  
## [817] 1 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [851] 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0  
## [885] 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0  
## [919] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0  
## [953] 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0  
## [987] 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1  
## [1021] 1 1 1 0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1  
## [1055] 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1  
## [1089] 1 1 0 1 0 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1  
## [1123] 1 1 0 1 1 0 0 1 1 1 1 1 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 0 1  
## [1157] 1 0 1 1 1 0 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0 1 0 0 1 1  
## [1191] 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0  
## [1225] 1 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0  
## [1259] 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1  
## [1293] 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1  
## [1327] 0 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1  
## [1361] 1 1 0 0 0 0 1 1 1 0 0 1 0 1 0 0 1 1 1 0 1 0 1 1 1 0 0 1 1 1 1 1 1 1  
## [1395] 0 0 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0  
## [1429] 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1  
## [1463] 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0  
## [1497] 1 1 1 1 0 1 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0  
## [1531] 0 1 1 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1  
## [1565] 1 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1  
## [1599] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1  
## [1633] 1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1667] 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1  
## [1701] 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 1 1 1  
## [1735] 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1  
## [1769] 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1  
## [1803] 1 1 1 1 0 0 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1837] 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 0 1 1 0 1 1 0 0 1 1 0  
## [1871] 1 0 0 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1  
## [1905] 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1  
## [1939] 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1  
## [1973] 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0  
##   
## $advanced\_pred  
## [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [35] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [69] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [103] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [137] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [171] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [205] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [239] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [273] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [307] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [341] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [375] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [409] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [443] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [477] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [511] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [545] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [579] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [613] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [647] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [681] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [715] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [749] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [783] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [817] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [851] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [885] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [919] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [953] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [987] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1021] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1055] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1089] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1123] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1157] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1191] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1225] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1259] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1293] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1327] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1361] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1395] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1429] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1463] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1497] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1531] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1565] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1599] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1633] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1667] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1701] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1735] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1769] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1803] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1837] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1871] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1905] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1939] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [1973] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

#################################################################################################  
#Make predictions and record the time it takes   
tm\_test=NA  
if(run.test){  
 load(file="../output/dat\_test\_sift.Rdata")  
 load(file="../output/dat\_test\_new.Rdata")  
 load(file="../output/fit\_train.Rdata")  
 tm\_test <- system.time(pred\_test <- test(fit\_train = fit\_train,   
 dat\_test\_base = dat\_test\_sift,  
 dat\_test\_adv = dat\_test\_new))  
 save(pred\_test, file="../output/pred\_test.RData")  
   
 #individual times for making predictions using base model and advanced model  
 tm\_test\_base <- system.time(pred\_test\_base <- test(fit\_train = fit\_train\_base,   
 dat\_test\_base = dat\_test\_sift,  
 model="base"))  
 tm\_test\_adv <- system.time(pred\_test\_adv <- test(fit\_train = fit\_train\_adv,   
 dat\_test\_adv = dat\_test\_new,   
 model="advanced"))  
}

### 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.

if(run.feature.train){  
 load("../output/tm\_new\_feature\_train.Rdata") #tm\_new\_feature\_train  
 cat("Time for constructing new training features=", tm\_new\_feature\_train[1], "s \n")  
}

## Time for constructing new training features= 1163.52 s

if (run.feature.test){  
load("../output/tm\_new\_feature\_test.Rdata") #tm\_new\_feature\_test  
cat("Time for constructing new testing features=", tm\_new\_feature\_test[1], "s \n")  
}  
if (run.train){  
cat("Time for training models (Baseline + Advanced)=", tm\_train[1], "s \n")  
cat("Time for training Baseline Model=", tm\_train\_base[1], "s \n")  
cat("Time for training Advanced Model=", tm\_train\_adv[1], "s \n")  
}

## Time for training models (Baseline + Advanced)= 31.142 s   
## Time for training Baseline Model= 10.985 s   
## Time for training Advanced Model= 19.311 s

if(run.test){  
cat("Time for making predictions (Baseline + Advanced)=", tm\_test[1], "s \n")  
cat("Time for making predictions with Baseline Model=", tm\_test\_base[1], "s \n")  
cat("Time for making predictions with Advanced Model=", tm\_test\_adv[1], "s \n")  
}