Main: Facial Expression Recognition Framework

Group 1: Kristen Akey, Levi Lee, Yiran Lin, Hanyi Yang, Wen Yin

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.

Step 0 set work directories

```
set.seed(2020)
# setwd("~/GitHub/Fall2020-Project3-group1/doc")


# change the working directory as needed
# if someone can make this a relative path, that would be great!!!
```

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

```
# change the directory of the data to where it's stored in your local drive as needed
train_dir <- "../data/train_set/" # This will be modified for different data sets.






# train_dir <- "~/train_set/"
train_image_dir <- paste(train_dir, "images/", sep="")
train_pt_dir <- paste(train_dir,  "points/", sep="")
train_label_path <- paste(train_dir, "label.csv", sep="")
```

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
- (T/F) reweighting the samples for 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

```r
K <- 5  # number of CV folds

run.fudicial.list <- FALSE
run.feature.train <- FALSE # process features for training set
run.feature.test <- FALSE # process features for test set
sample.reweight <- TRUE # run sample reweighting in model training

run.cv.gbm <- FALSE # run cross-validation on the training set for gbm
run.train.gbm <- FALSE # run evaluation on entire train set
run.test.gbm <- TRUE # run evaluation on an independent test set

run.cv.xgboost <- FALSE # run cross-validation on the training set for xgboost
run.train.xgboost <- FALSE # run evaluation on entire train set
run.test.xgboost <- TRUE # run evaluation on an independent test set

# add controls here to make if else statements to either cross-validate, test, train, or to just load s
# for xgboost, we need to also train and test each time we knit to record the time for the model
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications. In this Starter Code, we tune parameter lambda (the amount of shrinkage) for logistic regression with LASSO penalty.

```r
# hyperparameters for our models

# gbm model (baseline)
hyper_grid_gbm <- expand.grid(
  shrinkage = c(0.001, 0.005, 0.010, 0.050, 0.100),
  n.trees = c(600, 1200, 1800)
)

# xgboost model
hyper_grid_xgboost <- expand.grid(
  eta = c(0.01, 0.05, 0.1, 0.2, 0.3),
  lambda = c(0.001, 0.005, 0.010, 0.050, 0.100),
  gamma = c(0, 5),
  nrounds = c(100, 200, 600)
)


# add more hyperparameters for each model as needed
```

Step 2: import data and train-test split

```r
#train-test split
info <- read.csv(train_label_path)
n <- nrow(info)
n_train <- round(n*(4/5), 0)
train_idx <- sample(info$Index, n_train, replace = F)
test_idx <- setdiff(info$Index, train_idx)
```

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

```r
n_files <- length(list.files(train_image_dir))

if (run.fudicial.list){
  #function to read fiducial points
  #input: index
  #output: matrix of fiducial points corresponding to the index
  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")
} else {
  load(file="../output/fiducial_pt_list.RData")
}
```

Step 3: construct features and responses

- The follow plots show how pairwise distance between fiducial points can work as feature for facial emotion recognition.
  - In the first column, 78 fiducials points of each emotion are marked in order.
  - In the second column distributions of vertical distance between right pupil(1) and right brow peak(21) are shown in histograms. For example, the distance of an angry face tends to be shorter than that of a surprised face.
  - The third column is the distributions of vertical distances between right mouth corner(50) and the midpoint of the upper lip(52). For example, the distance of an happy face tends to be shorter than that of a sad face.

`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

```r
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))
  save(dat_train, tm_feature_train, file="../output/feature_train.RData")
}else{
  load(file="../output/feature_train.RData")
}

tm_feature_test <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature(fiducial_pt_list, test_idx))
  save(dat_test, tm_feature_test, file="../output/feature_test.RData")
```

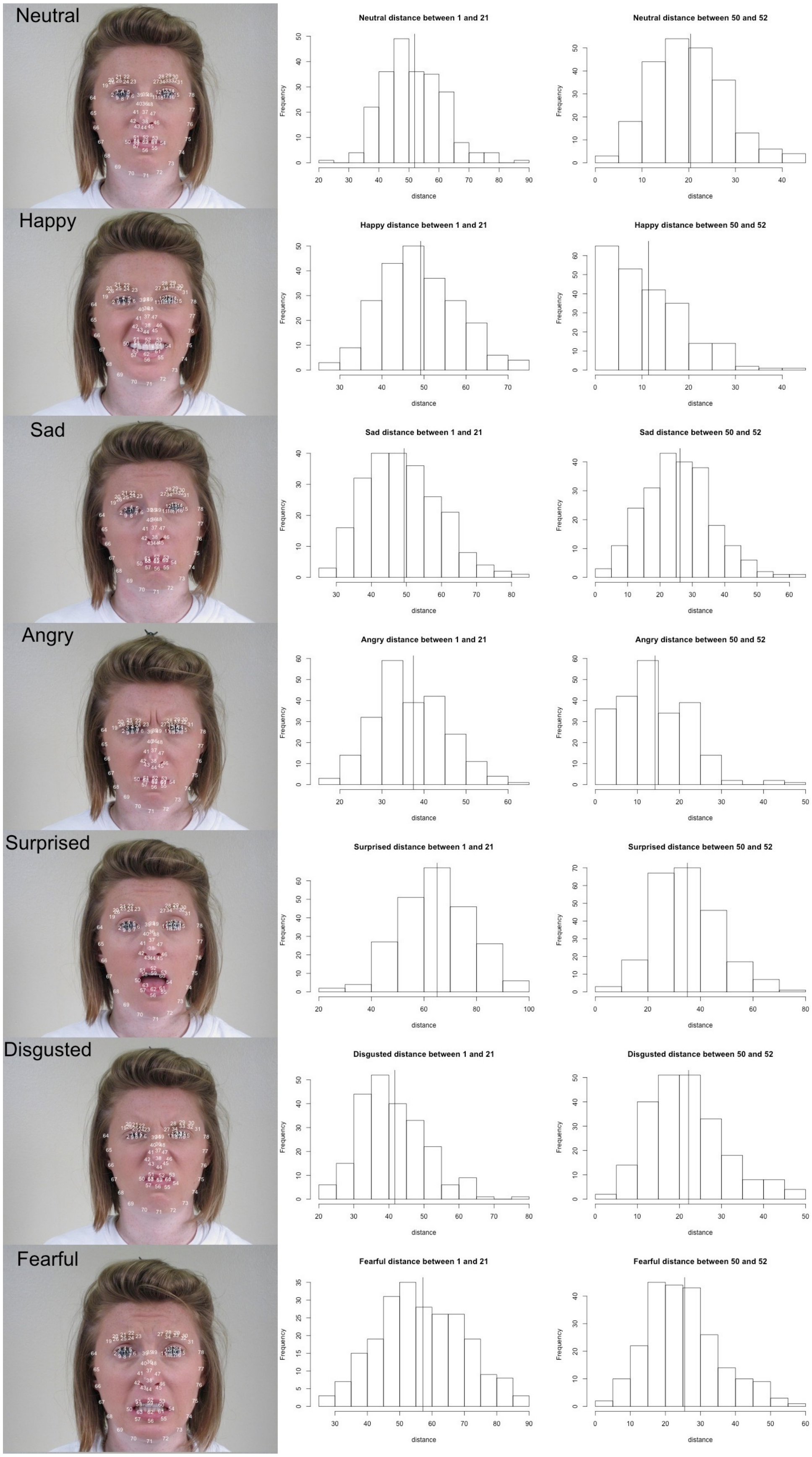


Figure 1: Figure1

```r
}else{
  load(file="../output/feature_test.RData")
}
```

Gradient Boosted Trees (gbm model) (Baseline Model)

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_gbm.R")
source("../lib/test_gbm.R")
```

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters.

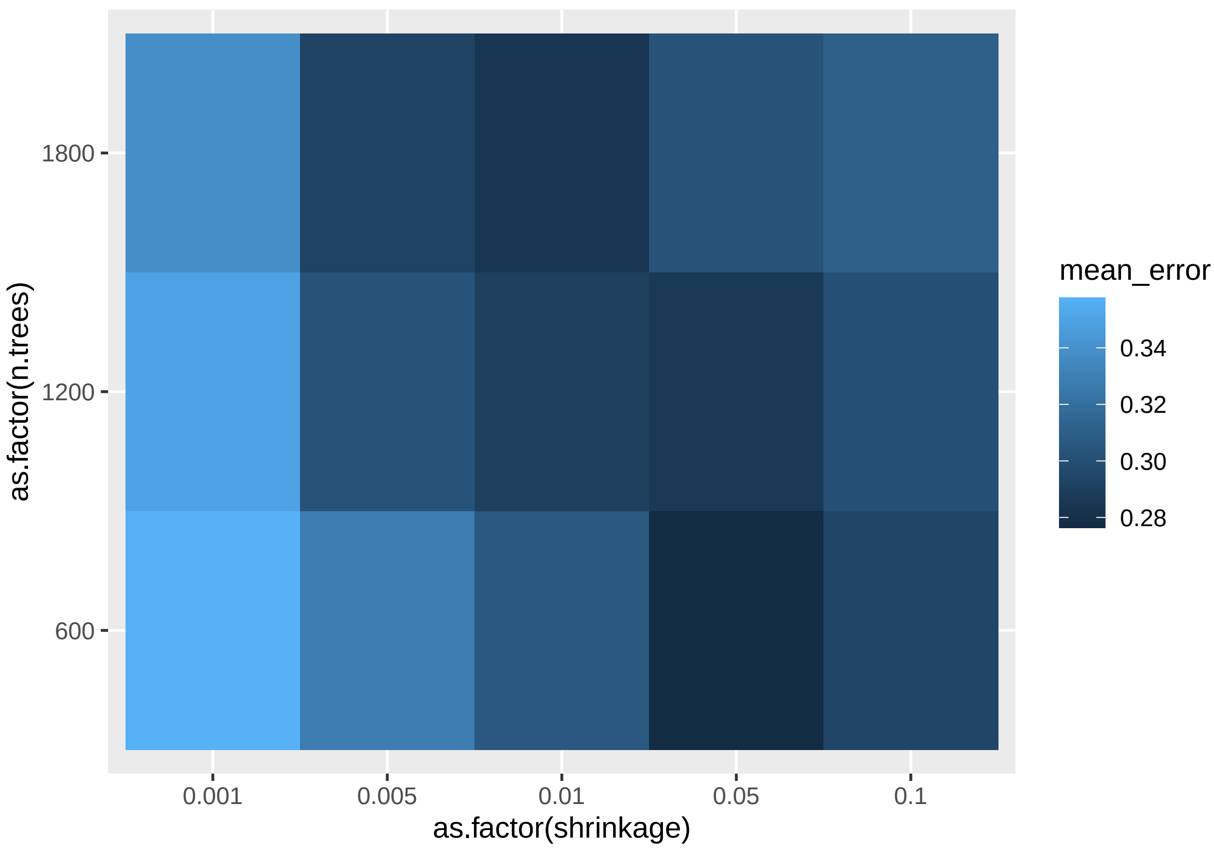
```
source("../lib/cross_validation_gbm.R")
feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)

if(run.cv.gbm){
  res_cv <- matrix(0, nrow = nrow(hyper_grid_gbm), ncol = 4)
  for(i in 1:nrow(hyper_grid_gbm)){
    cat("n.trees = ", hyper_grid_gbm$n.trees[i], ",
        shrinkage = ", hyper_grid_gbm$shrinkage[i],"\n", sep = "")
    res_cv[i,] <- cv.function(features = feature_train, labels = label_train,
                              num_trees = hyper_grid_gbm$n.trees[i],
                              shrink = hyper_grid_gbm$shrinkage[i],
                              K, reweight = sample.reweight)
  save(res_cv, file="../output/res_cv_gbm.RData")
  }
}else{
  load("../output/res_cv_gbm.RData")
}
```

*Visualize cross-validation results.

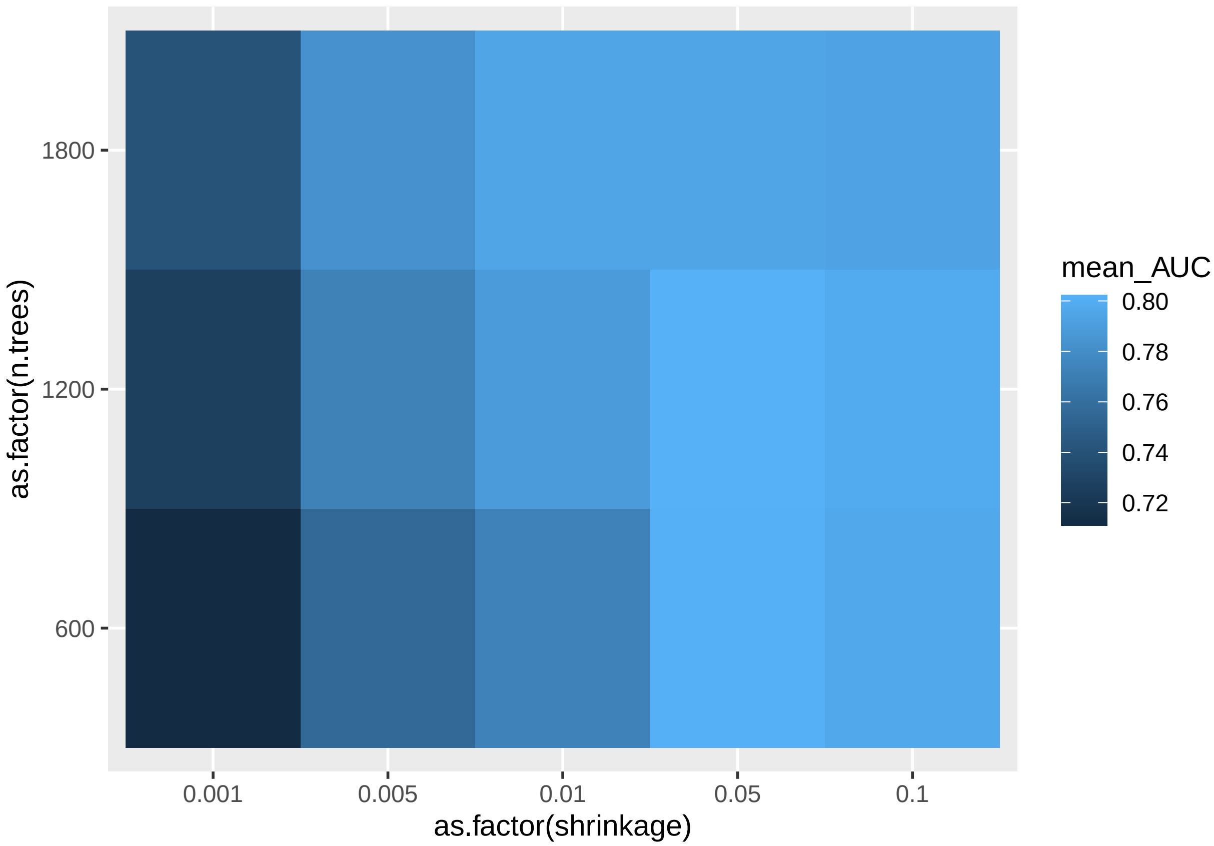
```
res_cv_gbm <- as.data.frame(res_cv)
colnames(res_cv_gbm) <- c("mean_error", "sd_error", "mean_AUC", "sd_AUC")

gbm_cv_results = data.frame(hyper_grid_gbm, res_cv_gbm)

gbm_cv_results
```

```
##    shrinkage n.trees mean_error    sd_error  mean_AUC      sd_AUC
## 1      0.001     600  0.3576347 0.023447152 0.7111546 0.020548747
## 2      0.005     600  0.3277413 0.011801334 0.7559083 0.015056717
## 3      0.010     600  0.3062226 0.010644901 0.7727803 0.015389634
## 4      0.050     600  0.2764453 0.008045855 0.8015074 0.009653744
## 5      0.100     600  0.2941406 0.026467746 0.7965060 0.021828465
## 6      0.001    1200  0.3494367 0.022503501 0.7269749 0.019971143
## 7      0.005    1200  0.3031100 0.012042165 0.7724047 0.015363576
## 8      0.010    1200  0.2905285 0.022779998 0.7885403 0.015763480
## 9      0.050    1200  0.2861576 0.026049925 0.8022183 0.018723658
## 10     0.100    1200  0.3004216 0.031100034 0.7986702 0.024253701
## 11     0.001    1800  0.3378972 0.010955498 0.7404200 0.020218630
## 12     0.005    1800  0.2925555 0.015907800 0.7828471 0.016634467
## 13     0.010    1800  0.2835979 0.007752051 0.7947552 0.017362962
## 14     0.050    1800  0.3031199 0.027850857 0.7949472 0.019940282
## 15     0.100    1800  0.3107072 0.028527596 0.7929691 0.021709803
```

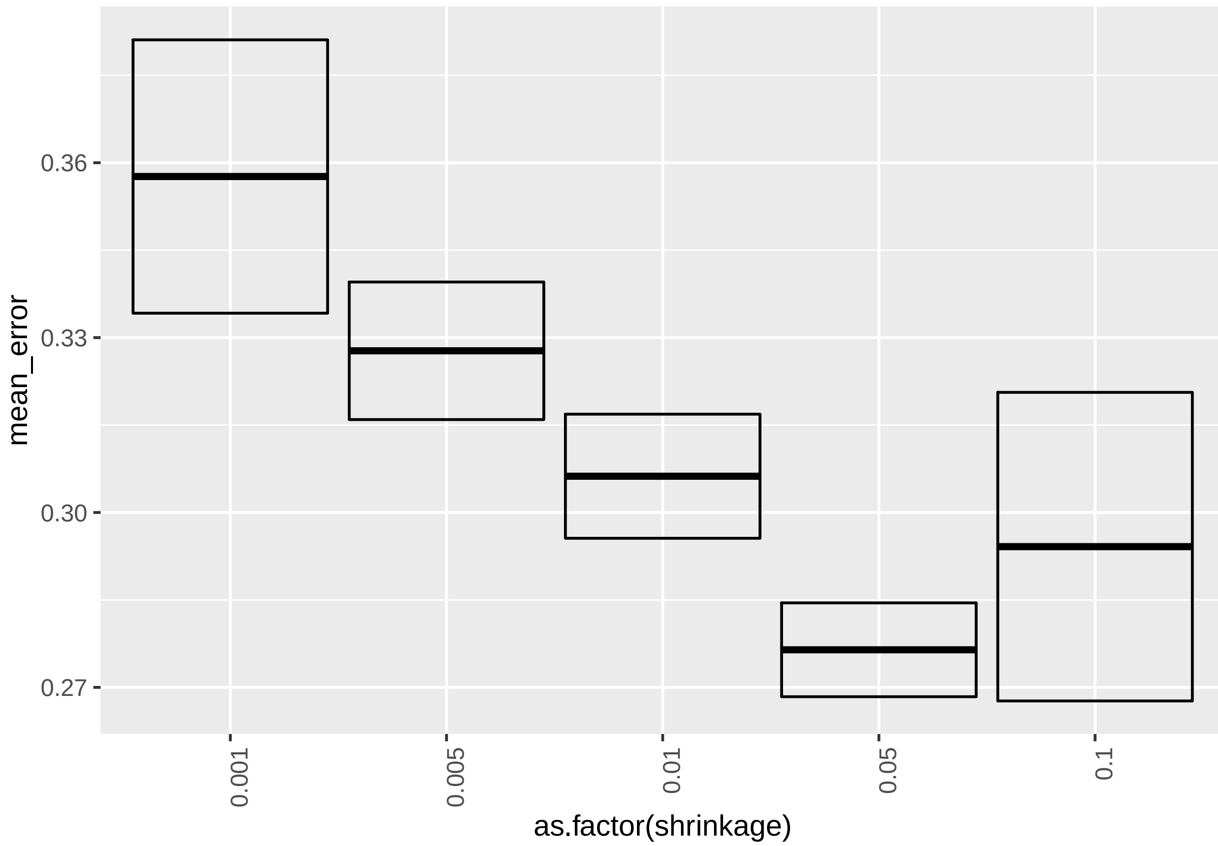
```r
# Mean Error
ggplot(gbm_cv_results, aes(as.factor(shrinkage), as.factor(n.trees), fill = mean_error)) +
  geom_tile()
```



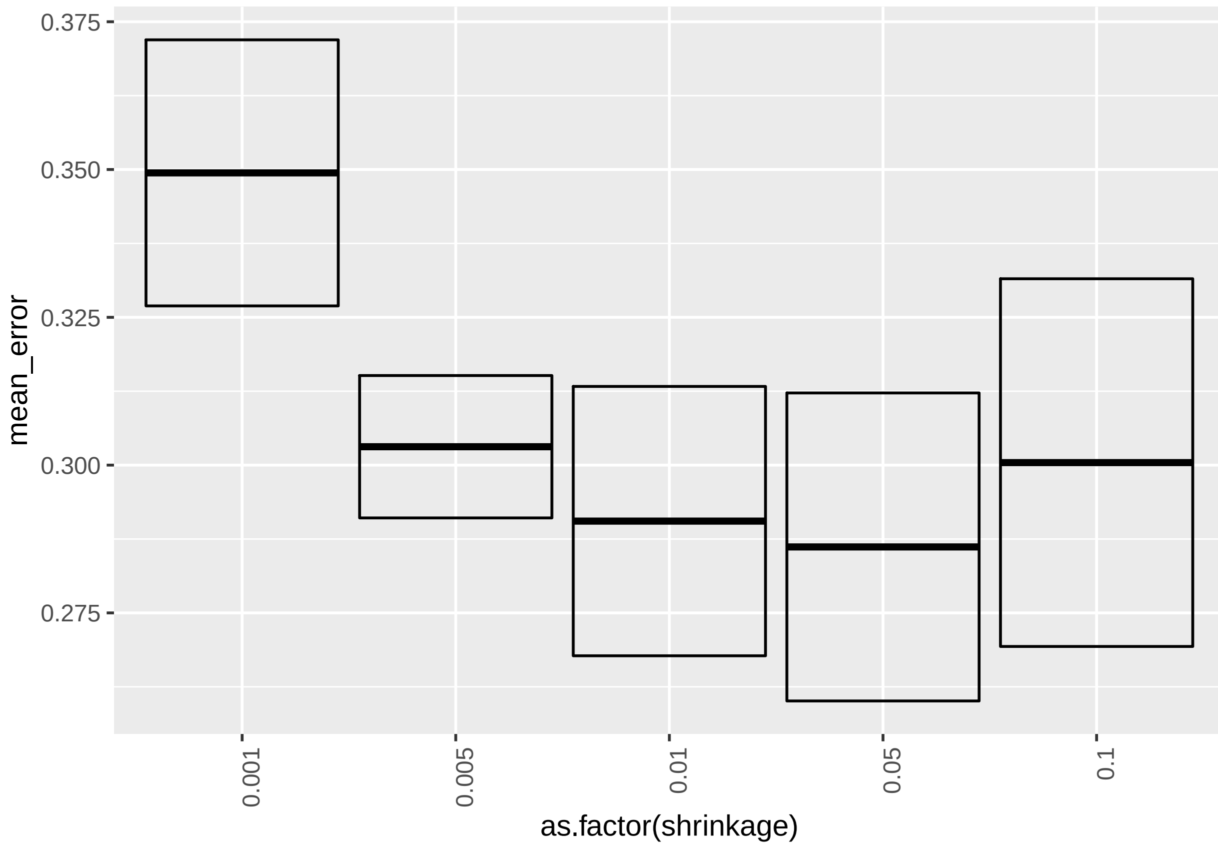

```r
# Mean AUC
ggplot(gbm_cv_results, aes(as.factor(shrinkage), as.factor(n.trees), fill = mean_AUC)) +
  geom_tile()
```



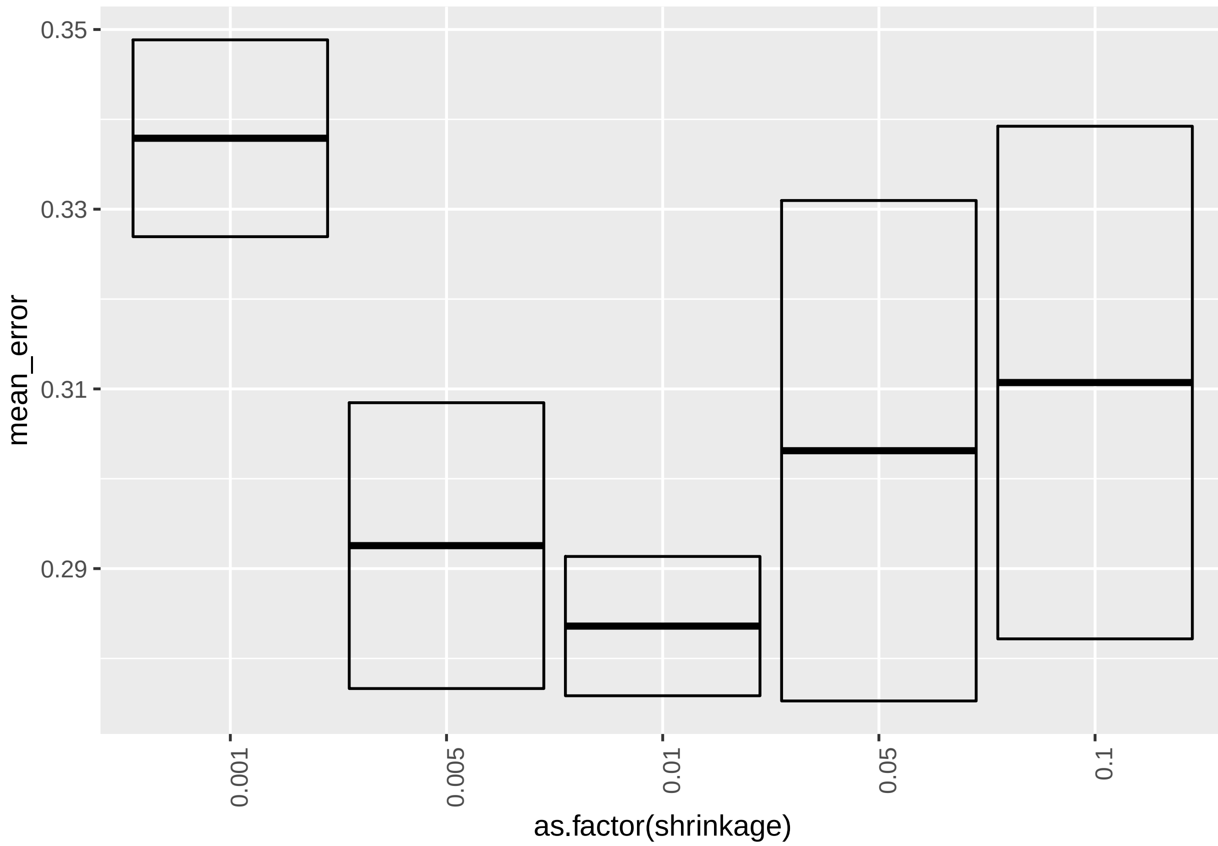

```r
# Mean Error
# N.Trees = 600
ggplot(gbm_cv_results[gbm_cv_results$n.trees == 600, ],
       aes(x = as.factor(shrinkage), y = mean_error,
           ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



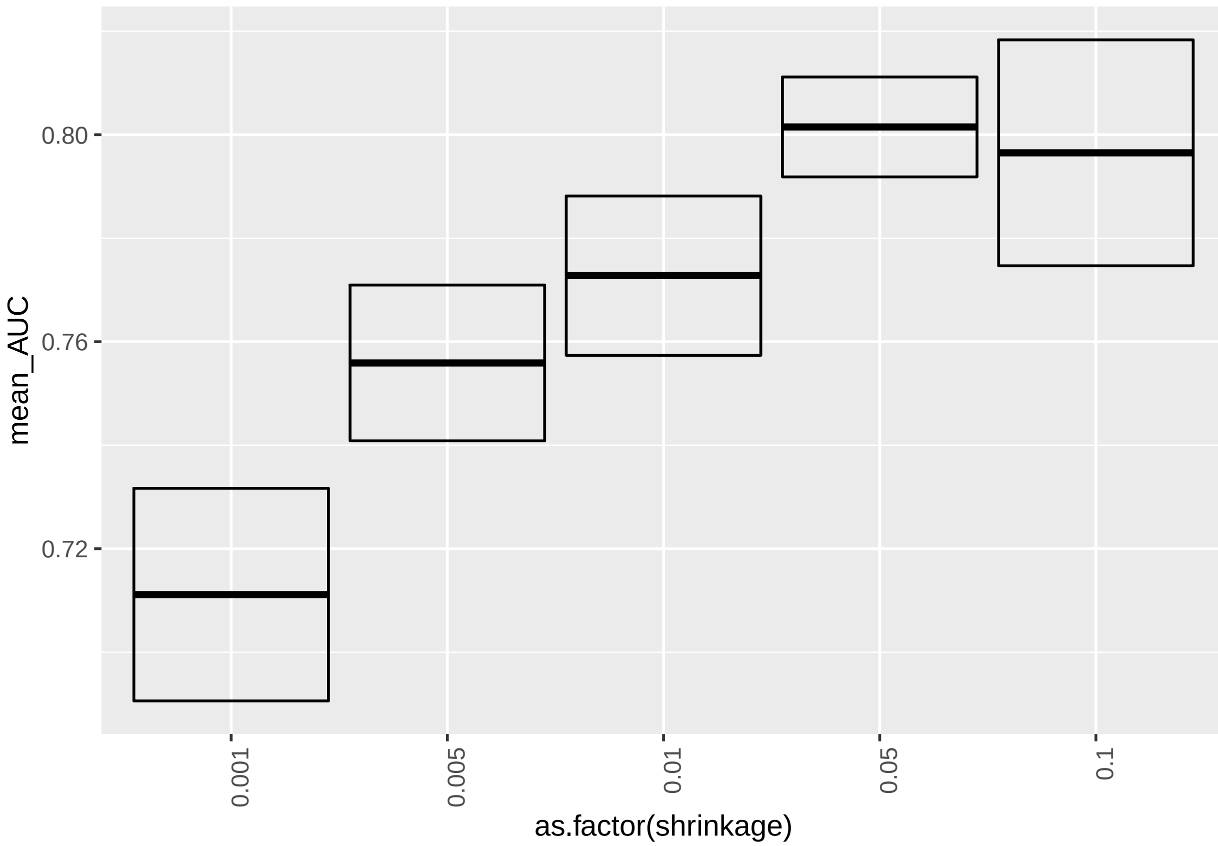

```
# N.Trees = 1200
ggplot(gbm_cv_results[gbm_cv_results$n.trees == 1200, ],
       aes(x = as.factor(shrinkage), y = mean_error,
           ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



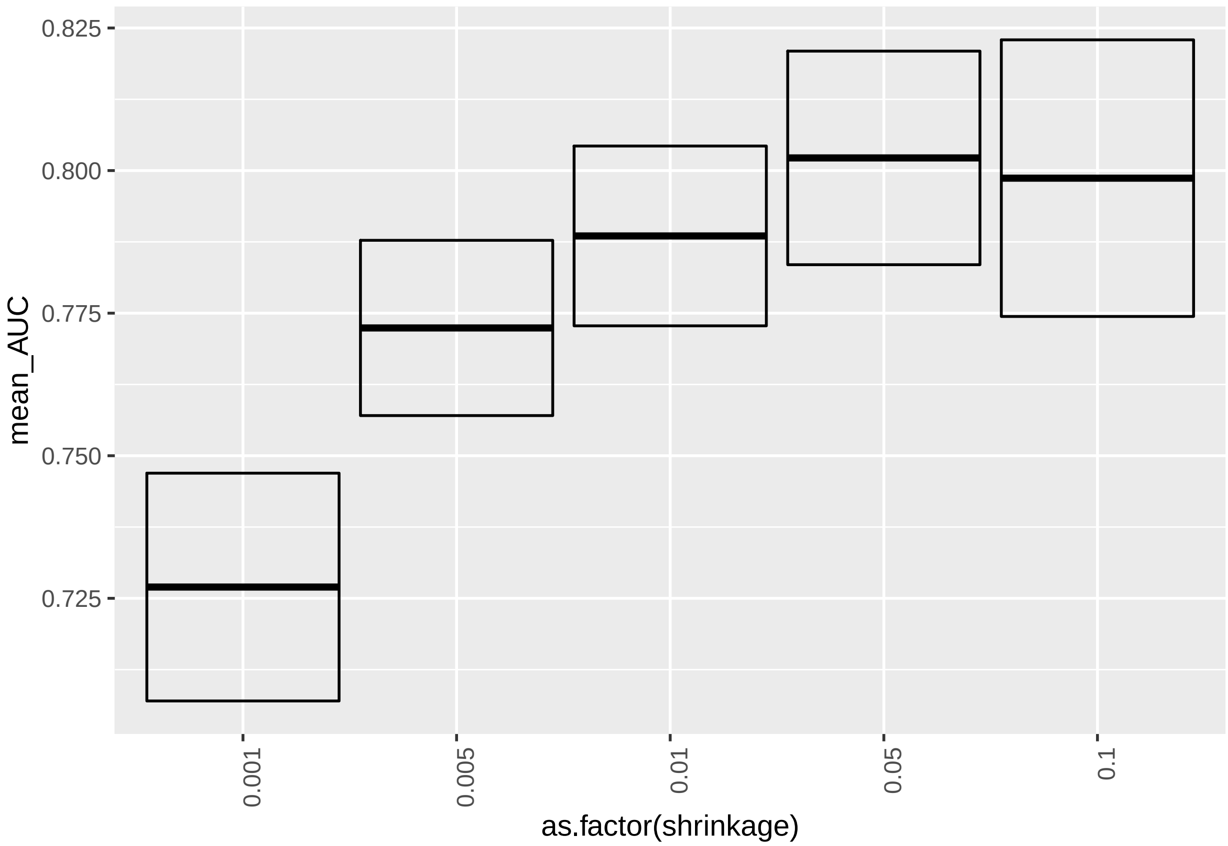

```
# N.Trees = 1800
ggplot(gbm_cv_results[gbm_cv_results$n.trees == 1800, ],
       aes(x = as.factor(shrinkage), y = mean_error,
           ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



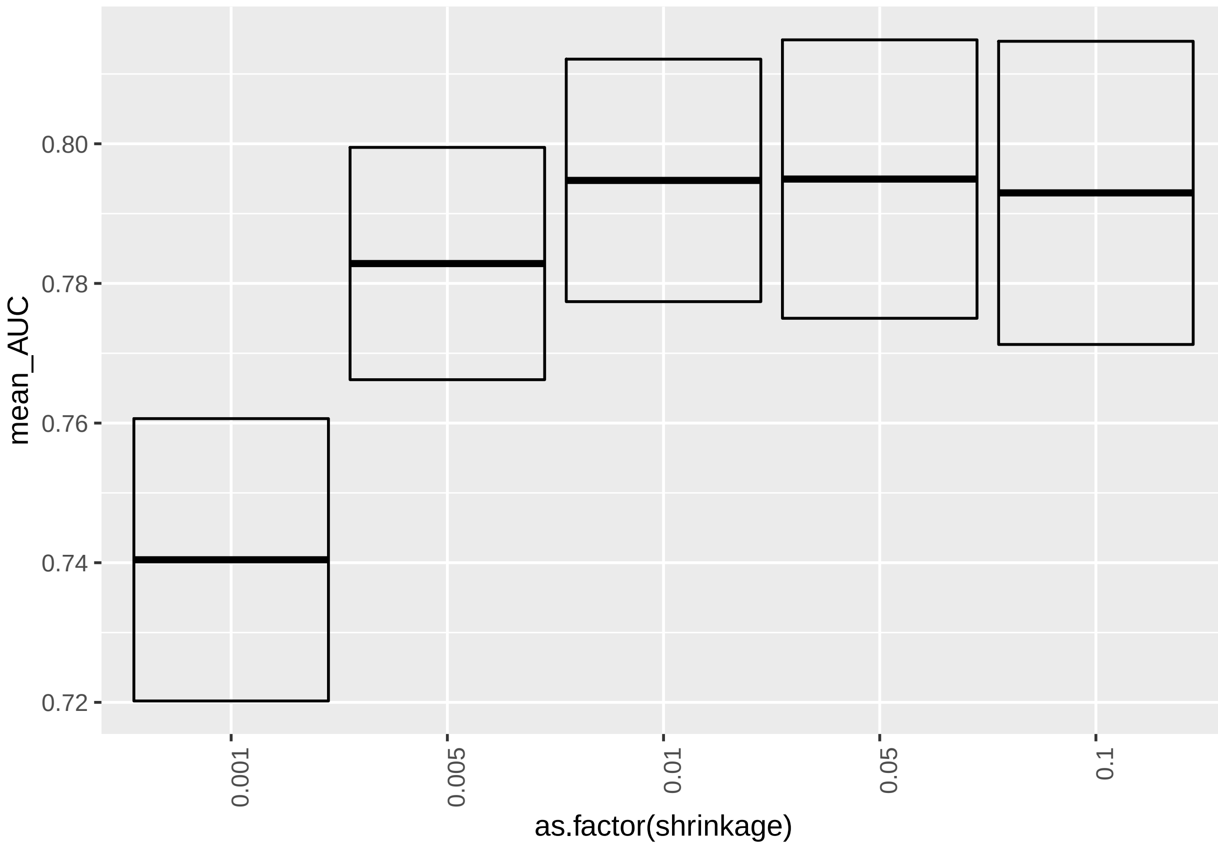

```r
# Mean AUC
# N.Trees = 600
ggplot(gbm_cv_results[gbm_cv_results$n.trees == 600, ],
       aes(x = as.factor(shrinkage), y = mean_AUC,
           ymin = mean_AUC - sd_AUC, ymax = mean_AUC + sd_AUC)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```r
# N.Trees = 1200
ggplot(gbm_cv_results[gbm_cv_results$n.trees == 1200, ],
       aes(x = as.factor(shrinkage), y = mean_AUC,
           ymin = mean_AUC - sd_AUC, ymax = mean_AUC + sd_AUC)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```
# N.Trees = 1800
ggplot(gbm_cv_results[gbm_cv_results$n.trees == 1800, ],
       aes(x = as.factor(shrinkage), y = mean_AUC,
           ymin = mean_AUC - sd_AUC, ymax = mean_AUC + sd_AUC)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



- Choose the "best" parameter value

```
par_best_gbm_ind <- which(gbm_cv_results$mean_AUC == max(gbm_cv_results$mean_AUC))
par_best_gbm_shrinkage <- gbm_cv_results$shrinkage[par_best_gbm_ind]
par_best_gbm_n.trees <- gbm_cv_results$n.trees[par_best_gbm_ind]
```

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

```
if (run.train.gbm) {
  # training weights
  weight_train <- rep(NA, length(label_train))
  for (v in unique(label_train)){
    weight_train[label_train == v] = 0.5 * length(label_train) / length(label_train[label_train == v])
  }

  if (sample.reweight){
    tm_train_gbm <- system.time(fit_train_gbm <- train(feature_train, label_train, w = weight_train,
                                                       num_trees = par_best_gbm_n.trees,
                                                       shrink = par_best_gbm_shrinkage))
  } else {
    tm_train_gbm <- system.time(fit_train_gbm <- train(feature_train, label_train, w = NULL,
                                                       num_trees = par_best_gbm_n.trees,
                                                       shrink = par_best_gbm_shrinkage))
  }
```

```r
  save(fit_train_gbm, tm_train_gbm, file="../output/fit_train_gbm.RData")

} else {
  load(file="../output/fit_train_gbm.RData")
}
```

Step 5: Run test on test images

```r
tm_test_gbm = NA
feature_test <- as.matrix(dat_test[, -6007])
label_test <- as.integer(dat_test$label)

if(run.test.gbm){
  load(file="../output/fit_train_gbm.RData")
  tm_test_gbm <- system.time({prob_pred <- test(fit_train_gbm, feature_test, pred.type = 'response');
                              label_pred <- ifelse(prob_pred >= 0.5, 1, 0)})
}
```

- Evaluation

```r
## reweight the test data to represent a balanced label distribution

weight_test <- rep(NA, length(label_test))
for (v in unique(label_test)){
  weight_test[label_test == v] = 0.5 * length(label_test) / length(label_test[label_test == v])
}


# convert the original 1-2 class into numeric 0s and 1s
label_test <- ifelse(label_test == 2, 0, 1)

accu <- sum(weight_test * (label_pred == label_test)) / sum(weight_test)
tpr.fpr <- WeightedROC(prob_pred, label_test, weight_test)
auc <- WeightedAUC(tpr.fpr)
```

```
## The accuracy of the gbm model (shinkage = 0.05, n.trees = 1200) is 74.4%.

## The AUC of the gbm model (shinkage = 0.05, n.trees = 1200) is 0.8086568.
```

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.

```
## Time for constructing training features = 2.14 seconds

## Time for constructing testing features = 0.16 seconds

## Time for training gbm model = 233.97 seconds

## Time for testing gbm model = 12.732 seconds
```

xgboost Model (Proposed Model)

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/cross_validation_xgboost.R")
source("../lib/train_xgboost.R")
source("../lib/test_xgboost.R")
```

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters.

```
if(run.cv.xgboost){
  res_cv <- matrix(0, nrow = nrow(hyper_grid_xgboost), ncol = 4)
  for (i in 1:nrow(hyper_grid_xgboost)){
    print(i)
    res_cv[i,] <- cv.function(features = feature_train, labels = label_train,
                              K,
                              eta_val = hyper_grid_xgboost$eta[i],
                              lmd = hyper_grid_xgboost$lambda[i],
                              gam = hyper_grid_xgboost$gamma[i],
                              nr = hyper_grid_xgboost$nrounds[i])
  save(res_cv, file="../output/res_cv_xgboost.RData")
  }
}else{
  load("../output/res_cv_xgboost.RData")
}
```

*Visualize cross-validation results.

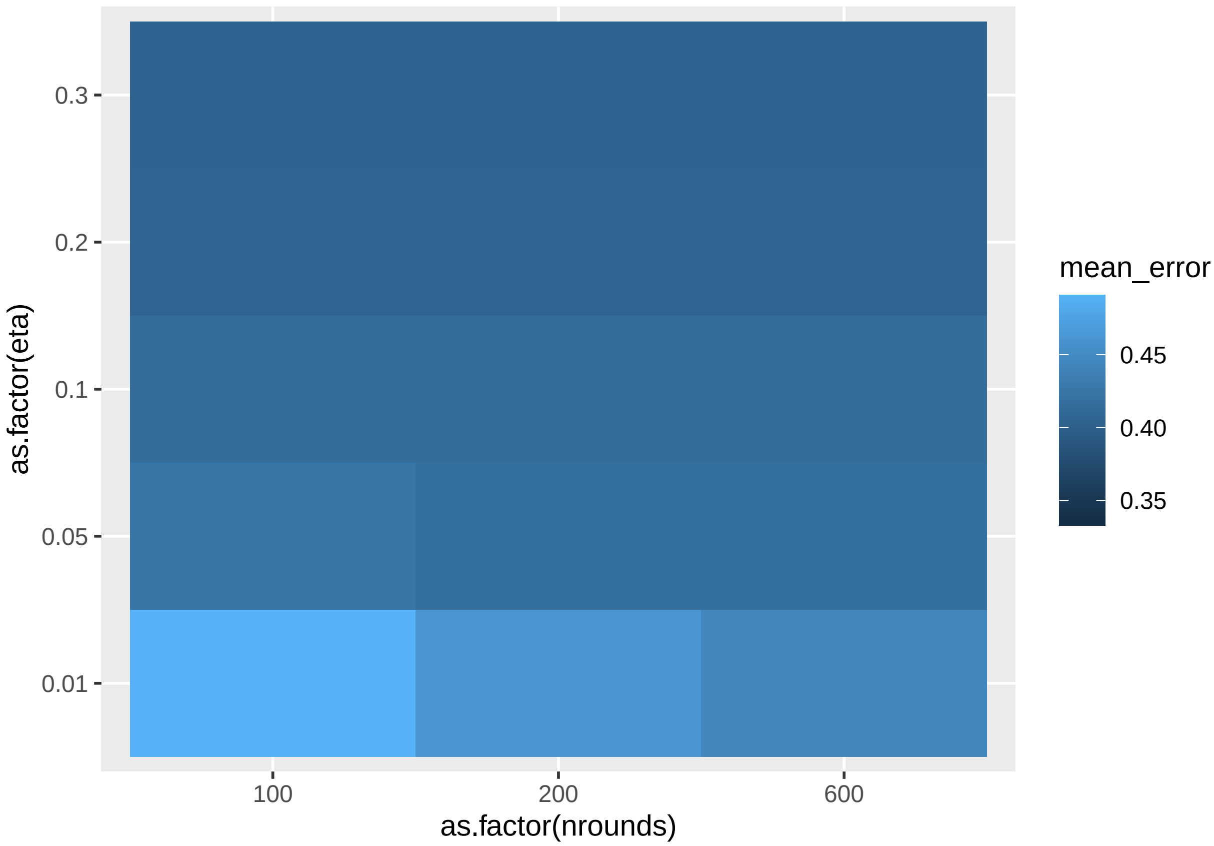
```
res_cv_xgboost <- as.data.frame(res_cv)
colnames(res_cv_xgboost) <- c("mean_error", "sd_error", "mean_AUC", "sd_AUC")

res_cv_xgboost_cv_results = data.frame(hyper_grid_xgboost, res_cv_xgboost)

res_cv_xgboost_cv_results
```

```
##      eta lambda gamma nrounds mean_error    sd_error  mean_AUC      sd_AUC
## 1   0.01  0.001     0     100  0.4906777 0.007370532 0.7493713 0.021512810
## 2   0.05  0.001     0     100  0.4269490 0.017502201 0.7984128 0.015150812
## 3   0.10  0.001     0     100  0.3967334 0.013854762 0.8080448 0.018028352
## 4   0.20  0.001     0     100  0.3802263 0.022275637 0.8004366 0.024310280
## 5   0.30  0.001     0     100  0.3618273 0.019997257 0.7924177 0.024227634
## 6   0.01  0.005     0     100  0.4906777 0.007370532 0.7493712 0.021511501
## 7   0.05  0.005     0     100  0.4251475 0.019096921 0.7991652 0.015752604
## 8   0.10  0.005     0     100  0.3976521 0.016905339 0.8068678 0.015141950
## 9   0.20  0.005     0     100  0.3714554 0.028059747 0.8020856 0.021416240
## 10  0.30  0.005     0     100  0.3588479 0.016613817 0.7986248 0.027706959
## 11  0.01  0.010     0     100  0.4906777 0.007370532 0.7493895 0.021463878
## 12  0.05  0.010     0     100  0.4251462 0.018664204 0.7986868 0.015603221
## 13  0.10  0.010     0     100  0.4035937 0.012926598 0.8074199 0.016325413
## 14  0.20  0.010     0     100  0.3770305 0.020917023 0.8024737 0.022217211
## 15  0.30  0.010     0     100  0.3500278 0.021987925 0.7976123 0.023782641
## 16  0.01  0.050     0     100  0.4906777 0.007370532 0.7495704 0.021375234
## 17  0.05  0.050     0     100  0.4258744 0.017794101 0.7989016 0.014980665
## 18  0.10  0.050     0     100  0.4014677 0.016798251 0.8078401 0.015908736
## 19  0.20  0.050     0     100  0.3729058 0.026252876 0.8042287 0.019369699
## 20  0.30  0.050     0     100  0.3625486 0.024690928 0.7940729 0.023655181
## 21  0.01  0.100     0     100  0.4904138 0.007135743 0.7498269 0.021594887
## 22  0.05  0.100     0     100  0.4259168 0.015373404 0.7994809 0.015485758
## 23  0.10  0.100     0     100  0.3987534 0.015274758 0.8081563 0.015002756
## 24  0.20  0.100     0     100  0.3762554 0.026295556 0.8032019 0.019209179
## 25  0.30  0.100     0     100  0.3575550 0.020906754 0.7963820 0.023517710
## 26  0.01  0.001     5     100  0.4906777 0.007370532 0.7493713 0.021512810
## 27  0.05  0.001     5     100  0.4269490 0.017502201 0.7984128 0.015150812
## 28  0.10  0.001     5     100  0.4154400 0.012458015 0.8051049 0.014138614
## 29  0.20  0.001     5     100  0.4107942 0.014726956 0.8033595 0.016394417
## 30  0.30  0.001     5     100  0.4046499 0.018198233 0.7979770 0.008939835
## 31  0.01  0.005     5     100  0.4906777 0.007370532 0.7493712 0.021511501
## 32  0.05  0.005     5     100  0.4251475 0.019096921 0.7991652 0.015752604
## 33  0.10  0.005     5     100  0.4144201 0.015494709 0.8052435 0.015479767
## 34  0.20  0.005     5     100  0.4115338 0.016405299 0.8017982 0.016251374
## 35  0.30  0.005     5     100  0.4027665 0.021082382 0.8034172 0.010682815
## 36  0.01  0.010     5     100  0.4906777 0.007370532 0.7493895 0.021463878
## 37  0.05  0.010     5     100  0.4251462 0.018664204 0.7986868 0.015603221
## 38  0.10  0.010     5     100  0.4156444 0.011930234 0.8044954 0.015731661
## 39  0.20  0.010     5     100  0.4065506 0.014654309 0.8016072 0.016248020
## 40  0.30  0.010     5     100  0.4011374 0.020653877 0.7984039 0.015230475
## 41  0.01  0.050     5     100  0.4906777 0.007370532 0.7495704 0.021375234
## 42  0.05  0.050     5     100  0.4258744 0.017794101 0.7989016 0.014980665
## 43  0.10  0.050     5     100  0.4143769 0.017366280 0.8050385 0.014988197
## 44  0.20  0.050     5     100  0.4050338 0.014409292 0.8028591 0.010353641
## 45  0.30  0.050     5     100  0.4057419 0.020123531 0.8003049 0.012840485
## 46  0.01  0.100     5     100  0.4904138 0.007135743 0.7498269 0.021594887
## 47  0.05  0.100     5     100  0.4259168 0.015373404 0.7994809 0.015485758
## 48  0.10  0.100     5     100  0.4137265 0.015457852 0.8058696 0.015549654
## 49  0.20  0.100     5     100  0.4061859 0.012503112 0.8036462 0.007516747
## 50  0.30  0.100     5     100  0.4048464 0.017908749 0.7960404 0.012265871
## 51  0.01  0.001     0     200  0.4615955 0.009160897 0.7788102 0.015696314
## 52  0.05  0.001     0     200  0.4048485 0.013761008 0.8074209 0.015090616
## 53  0.10  0.001     0     200  0.3769119 0.024008194 0.8035949 0.023844967
```

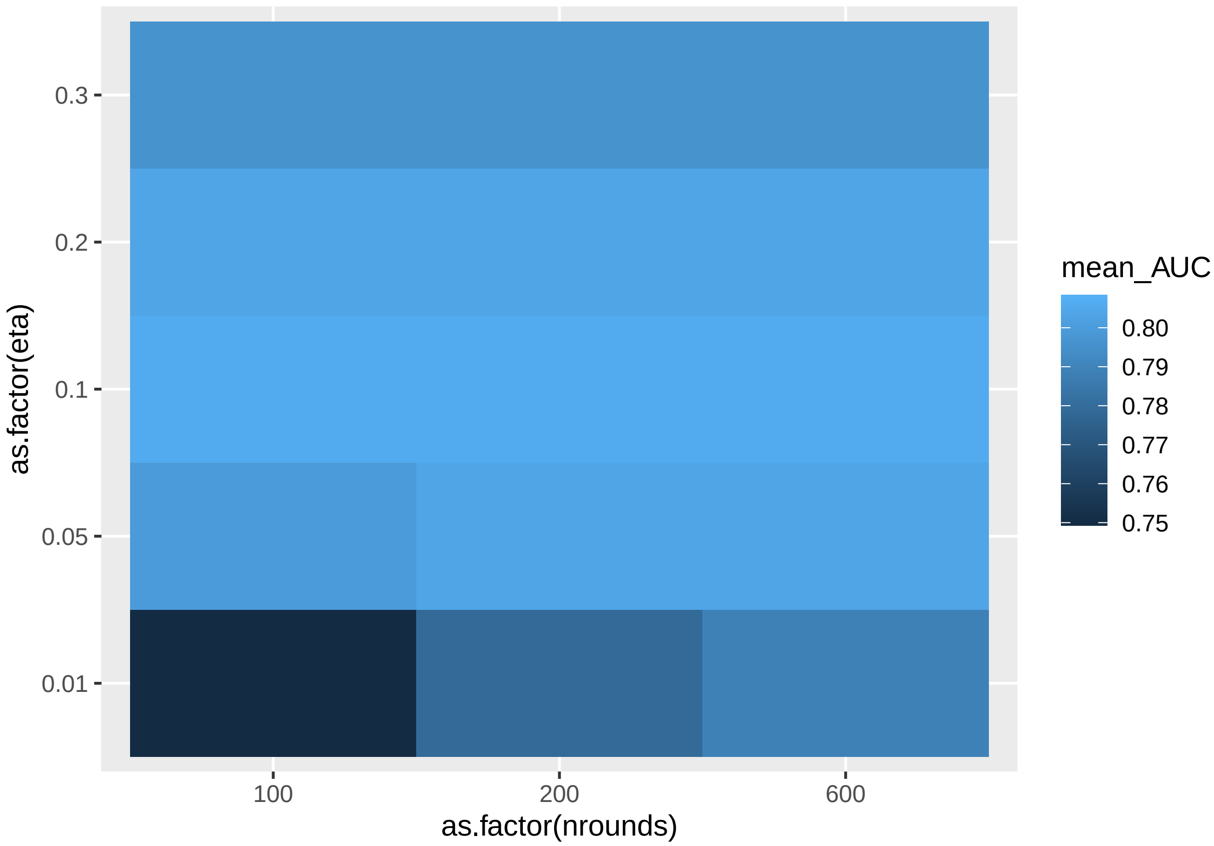
```
## 54  0.20  0.001     0      200   0.3566192 0.012450444 0.7932479 0.027131994
## 55  0.30  0.001     0      200   0.3503374 0.023226630 0.7910814 0.023516309
## 56  0.01  0.005     0      200   0.4615955 0.009160897 0.7787822 0.016130688
## 57  0.05  0.005     0      200   0.4054213 0.015531547 0.8072834 0.014597854
## 58  0.10  0.005     0      200   0.3801714 0.021434839 0.8020552 0.022992163
## 59  0.20  0.005     0      200   0.3563650 0.018533892 0.7954614 0.022996039
## 60  0.30  0.005     0      200   0.3392984 0.028925628 0.7964171 0.025232192
## 61  0.01  0.010     0      200   0.4615955 0.009160897 0.7787671 0.015849051
## 62  0.05  0.010     0      200   0.4094424 0.020839728 0.8072104 0.014290764
## 63  0.10  0.010     0      200   0.3776749 0.020324924 0.8028266 0.023601778
## 64  0.20  0.010     0      200   0.3575453 0.017639274 0.7947047 0.024452694
## 65  0.30  0.010     0      200   0.3470346 0.027459587 0.7935134 0.025482033
## 66  0.01  0.050     0      200   0.4615955 0.009160897 0.7786848 0.016080235
## 67  0.05  0.050     0      200   0.4054330 0.018383518 0.8083090 0.014793996
## 68  0.10  0.050     0      200   0.3813304 0.022033539 0.8034491 0.024120929
## 69  0.20  0.050     0      200   0.3574039 0.020227396 0.7956913 0.022024514
## 70  0.30  0.050     0      200   0.3481230 0.011280241 0.7934894 0.022611256
## 71  0.01  0.100     0      200   0.4615955 0.009160897 0.7788813 0.016026083
## 72  0.05  0.100     0      200   0.4019064 0.011636505 0.8072567 0.014257891
## 73  0.10  0.100     0      200   0.3779758 0.019775270 0.8055107 0.022806607
## 74  0.20  0.100     0      200   0.3499127 0.019270100 0.7978881 0.023613965
## 75  0.30  0.100     0      200   0.3416265 0.015418106 0.7951568 0.023665776
## 76  0.01  0.001     5      200   0.4615955 0.009160897 0.7788102 0.015696314
## 77  0.05  0.001     5      200   0.4185294 0.015184471 0.8035309 0.013806411
## 78  0.10  0.001     5      200   0.4154400 0.012458015 0.8051049 0.014138614
## 79  0.20  0.001     5      200   0.4107942 0.014726956 0.8033595 0.016394417
## 80  0.30  0.001     5      200   0.4046499 0.018198233 0.7979770 0.008939835
## 81  0.01  0.005     5      200   0.4615955 0.009160897 0.7787822 0.016130688
## 82  0.05  0.005     5      200   0.4174985 0.014086877 0.8035957 0.014111349
## 83  0.10  0.005     5      200   0.4144201 0.015494709 0.8052435 0.015479767
## 84  0.20  0.005     5      200   0.4115338 0.016405299 0.8017982 0.016251374
## 85  0.30  0.005     5      200   0.4027665 0.021082382 0.8034172 0.010682815
## 86  0.01  0.010     5      200   0.4615955 0.009160897 0.7787671 0.015849051
## 87  0.05  0.010     5      200   0.4174985 0.014086877 0.8042510 0.013930606
## 88  0.10  0.010     5      200   0.4156444 0.011930234 0.8044954 0.015731661
## 89  0.20  0.010     5      200   0.4065506 0.014654309 0.8016072 0.016248020
## 90  0.30  0.010     5      200   0.4011374 0.020653877 0.7984039 0.015230475
## 91  0.01  0.050     5      200   0.4615955 0.009160897 0.7786848 0.016080235
## 92  0.05  0.050     5      200   0.4166843 0.014804206 0.8032928 0.014654747
## 93  0.10  0.050     5      200   0.4143769 0.017366280 0.8050385 0.014988197
## 94  0.20  0.050     5      200   0.4050338 0.014409292 0.8028591 0.010353641
## 95  0.30  0.050     5      200   0.4057419 0.020123531 0.8003049 0.012840485
## 96  0.01  0.100     5      200   0.4615955 0.009160897 0.7788813 0.016026083
## 97  0.05  0.100     5      200   0.4174375 0.010785276 0.8035164 0.015512558
## 98  0.10  0.100     5      200   0.4137265 0.015457852 0.8058696 0.015549654
## 99  0.20  0.100     5      200   0.4061859 0.012503112 0.8036462 0.007516747
## 100 0.30  0.100     5      200   0.4048464 0.017908749 0.7960404 0.012265871
## 101 0.01  0.001     0      600   0.4437645 0.013777979 0.7883131 0.015401787
## 102 0.05  0.001     0      600   0.3835403 0.022916992 0.8057435 0.019116530
## 103 0.10  0.001     0      600   0.3654317 0.013245676 0.7986175 0.025228027
## 104 0.20  0.001     0      600   0.3486151 0.021180715 0.7942583 0.025609927
## 105 0.30  0.001     0      600   0.3330139 0.024322477 0.7974005 0.023020208
## 106 0.01  0.005     0      600   0.4437645 0.013777979 0.7883814 0.015458859
## 107 0.05  0.005     0      600   0.3867539 0.021166860 0.8058633 0.018717227
```

```
## 108 0.10  0.005     0     600  0.3660274 0.014305945 0.7970936 0.023052354
## 109 0.20  0.005     0     600  0.3431298 0.022724907 0.7945000 0.023678039
## 110 0.30  0.005     0     600  0.3410718 0.023715061 0.8003460 0.024720129
## 111 0.01  0.010     0     600  0.4437645 0.013777979 0.7883371 0.015501306
## 112 0.05  0.010     0     600  0.3881028 0.023279016 0.8057542 0.019197069
## 113 0.10  0.010     0     600  0.3677815 0.012072210 0.7980814 0.025349084
## 114 0.20  0.010     0     600  0.3516864 0.016395512 0.7929924 0.024112589
## 115 0.30  0.010     0     600  0.3354363 0.021478487 0.7968150 0.021017381
## 116 0.01  0.050     0     600  0.4437645 0.013777979 0.7887439 0.015342614
## 117 0.05  0.050     0     600  0.3860203 0.021352695 0.8058191 0.019154772
## 118 0.10  0.050     0     600  0.3613560 0.016238318 0.7986472 0.025248653
## 119 0.20  0.050     0     600  0.3429626 0.020871134 0.7939703 0.023545448
## 120 0.30  0.050     0     600  0.3435116 0.022652894 0.7973209 0.020972939
## 121 0.01  0.100     0     600  0.4437645 0.013777979 0.7888414 0.015331579
## 122 0.05  0.100     0     600  0.3859067 0.022195282 0.8064627 0.019040327
## 123 0.10  0.100     0     600  0.3626835 0.016047646 0.7996072 0.022850309
## 124 0.20  0.100     0     600  0.3470822 0.011875493 0.7961901 0.022430643
## 125 0.30  0.100     0     600  0.3375910 0.018733751 0.7965280 0.019744586
## 126 0.01  0.001     5     600  0.4437645 0.013777979 0.7883131 0.015401787
## 127 0.05  0.001     5     600  0.4185294 0.015184471 0.8035309 0.013806411
## 128 0.10  0.001     5     600  0.4154400 0.012458015 0.8051049 0.014138614
## 129 0.20  0.001     5     600  0.4107942 0.014726956 0.8033595 0.016394417
## 130 0.30  0.001     5     600  0.4046499 0.018198233 0.7979770 0.008939835
## 131 0.01  0.005     5     600  0.4437645 0.013777979 0.7883814 0.015458859
## 132 0.05  0.005     5     600  0.4174985 0.014086877 0.8035957 0.014111349
## 133 0.10  0.005     5     600  0.4144201 0.015494709 0.8052435 0.015479767
## 134 0.20  0.005     5     600  0.4115338 0.016405299 0.8017982 0.016251374
## 135 0.30  0.005     5     600  0.4027665 0.021082382 0.8034172 0.010682815
## 136 0.01  0.010     5     600  0.4437645 0.013777979 0.7883371 0.015501306
## 137 0.05  0.010     5     600  0.4174985 0.014086877 0.8042510 0.013930606
## 138 0.10  0.010     5     600  0.4156444 0.011930234 0.8044954 0.015731661
## 139 0.20  0.010     5     600  0.4065506 0.014654309 0.8016072 0.016248020
## 140 0.30  0.010     5     600  0.4011374 0.020653877 0.7984039 0.015230475
## 141 0.01  0.050     5     600  0.4437645 0.013777979 0.7887439 0.015342614
## 142 0.05  0.050     5     600  0.4166843 0.014804206 0.8032928 0.014654747
## 143 0.10  0.050     5     600  0.4143769 0.017366280 0.8050385 0.014988197
## 144 0.20  0.050     5     600  0.4050338 0.014409292 0.8028591 0.010353641
## 145 0.30  0.050     5     600  0.4057419 0.020123531 0.8003049 0.012840485
## 146 0.01  0.100     5     600  0.4437645 0.013777979 0.7888414 0.015331579
## 147 0.05  0.100     5     600  0.4174375 0.010785276 0.8035164 0.015512558
## 148 0.10  0.100     5     600  0.4137265 0.015457852 0.8058696 0.015549654
## 149 0.20  0.100     5     600  0.4061859 0.012503112 0.8036462 0.007516747
## 150 0.30  0.100     5     600  0.4048464 0.017908749 0.7960404 0.012265871
```

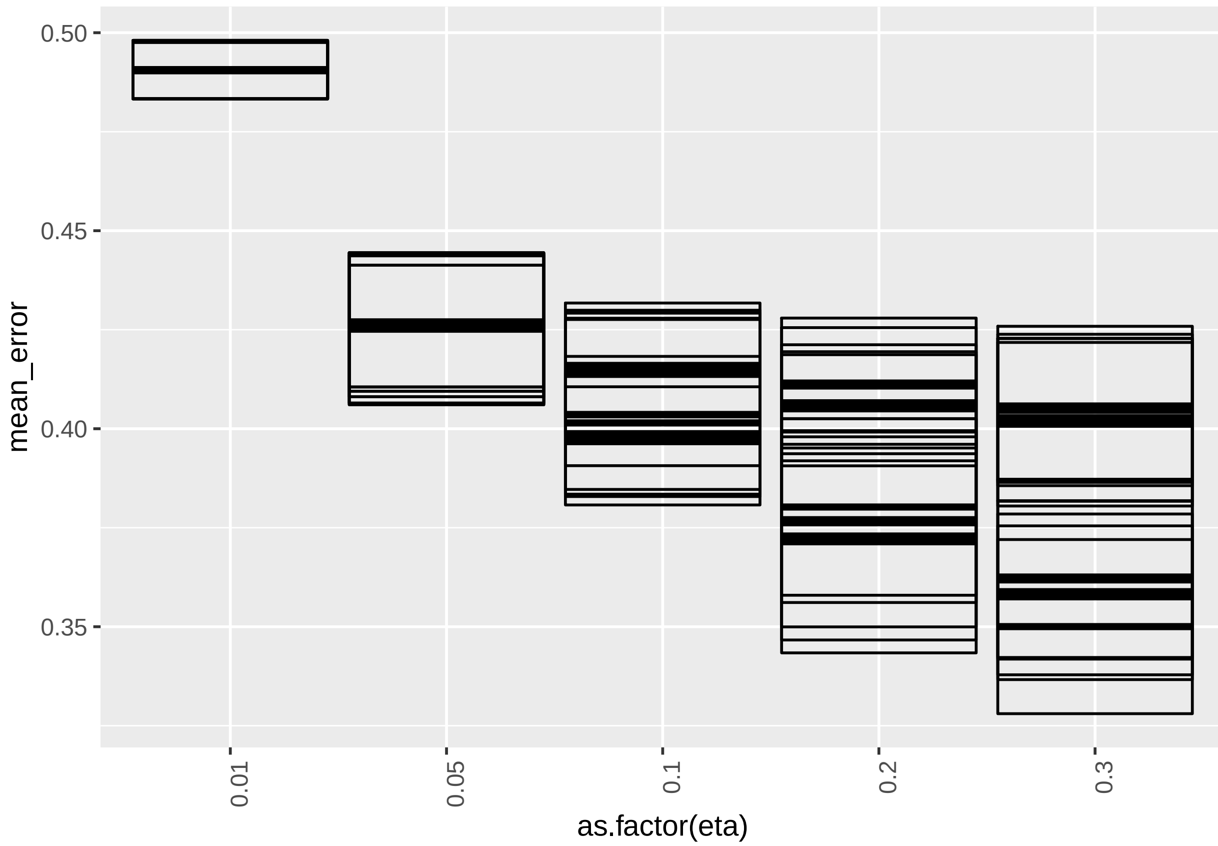
```r
# Mean Error
ggplot(res_cv_xgboost_cv_results, aes(as.factor(nrounds), as.factor(eta), fill = mean_error)) +
  geom_tile()
```



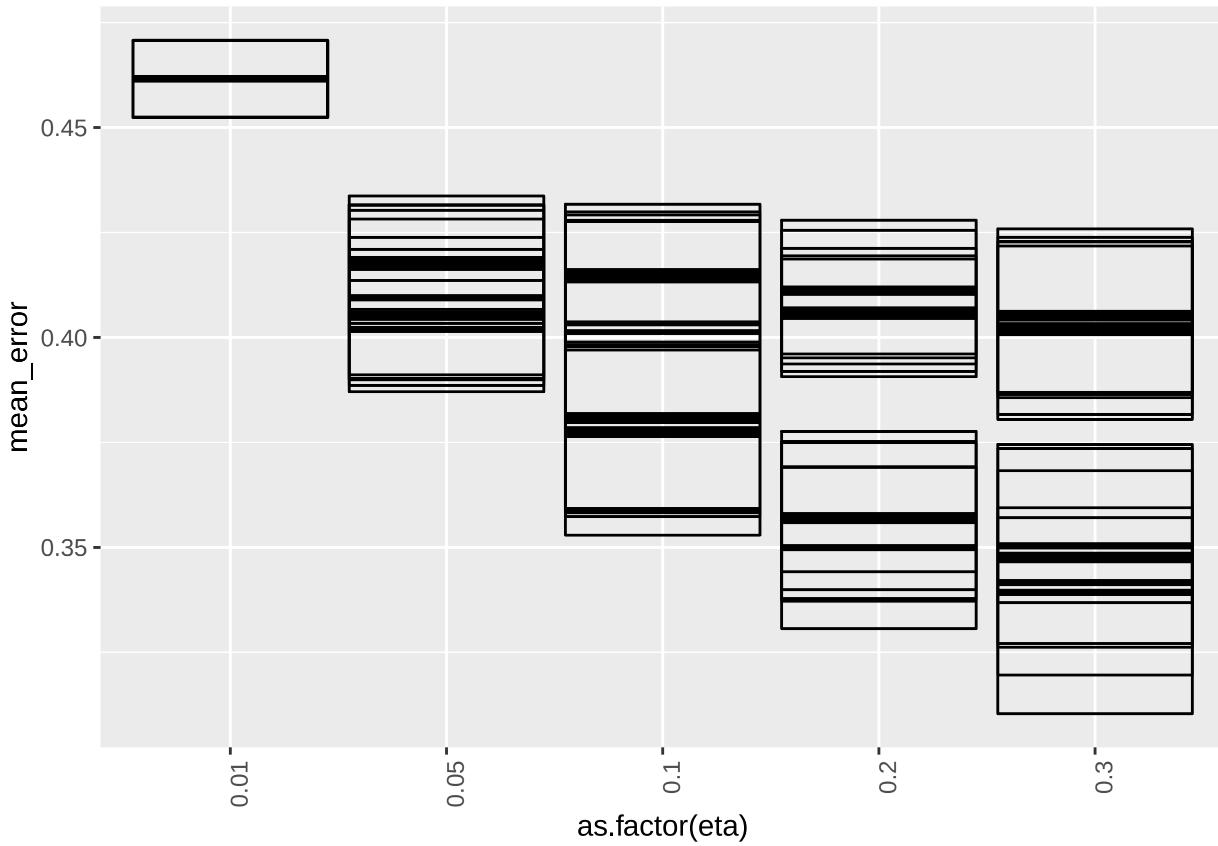

```r
# Mean AUC
ggplot(res_cv_xgboost_cv_results, aes(as.factor(nrounds), as.factor(eta), fill = mean_AUC)) +
  geom_tile()
```



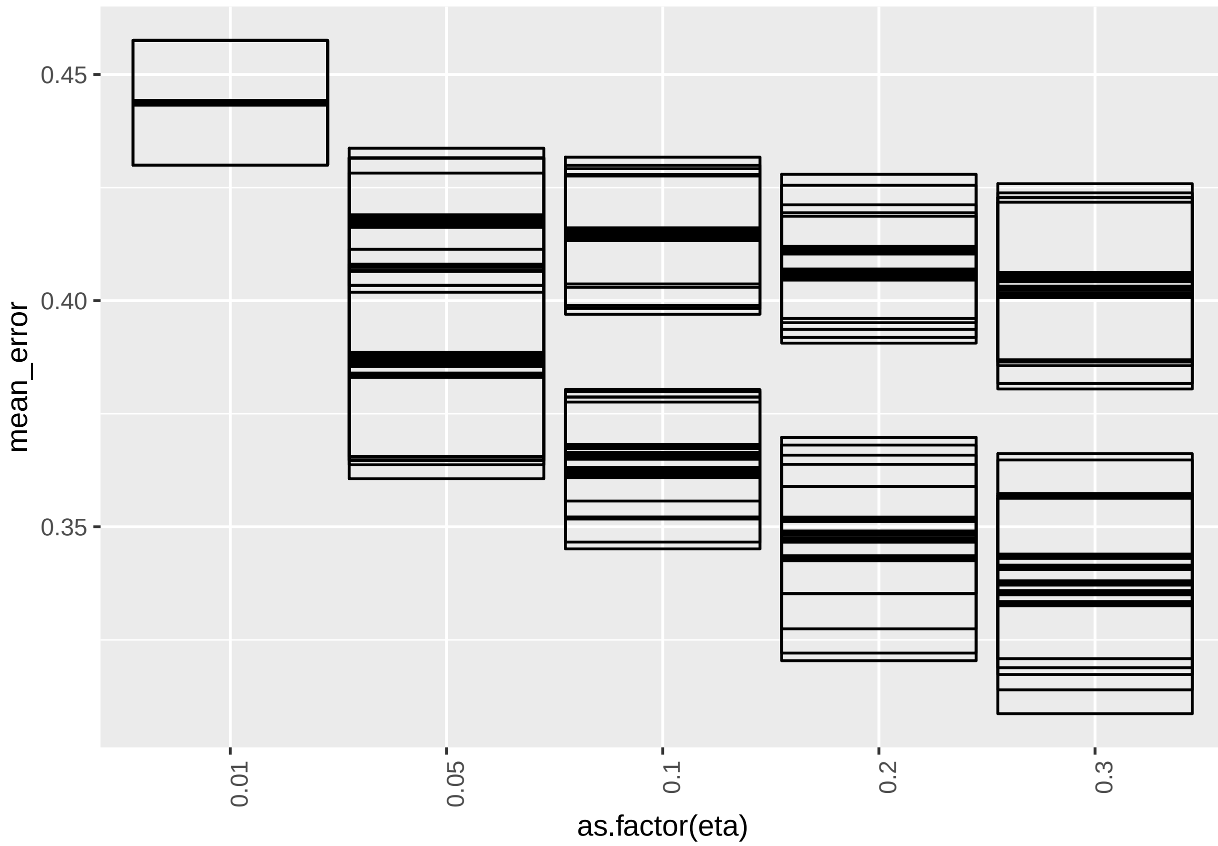

```r
# Mean Error
# nrounds = 100
ggplot(res_cv_xgboost_cv_results[res_cv_xgboost_cv_results$nrounds == 100, ],
       aes(x = as.factor(eta), y = mean_error,
           ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



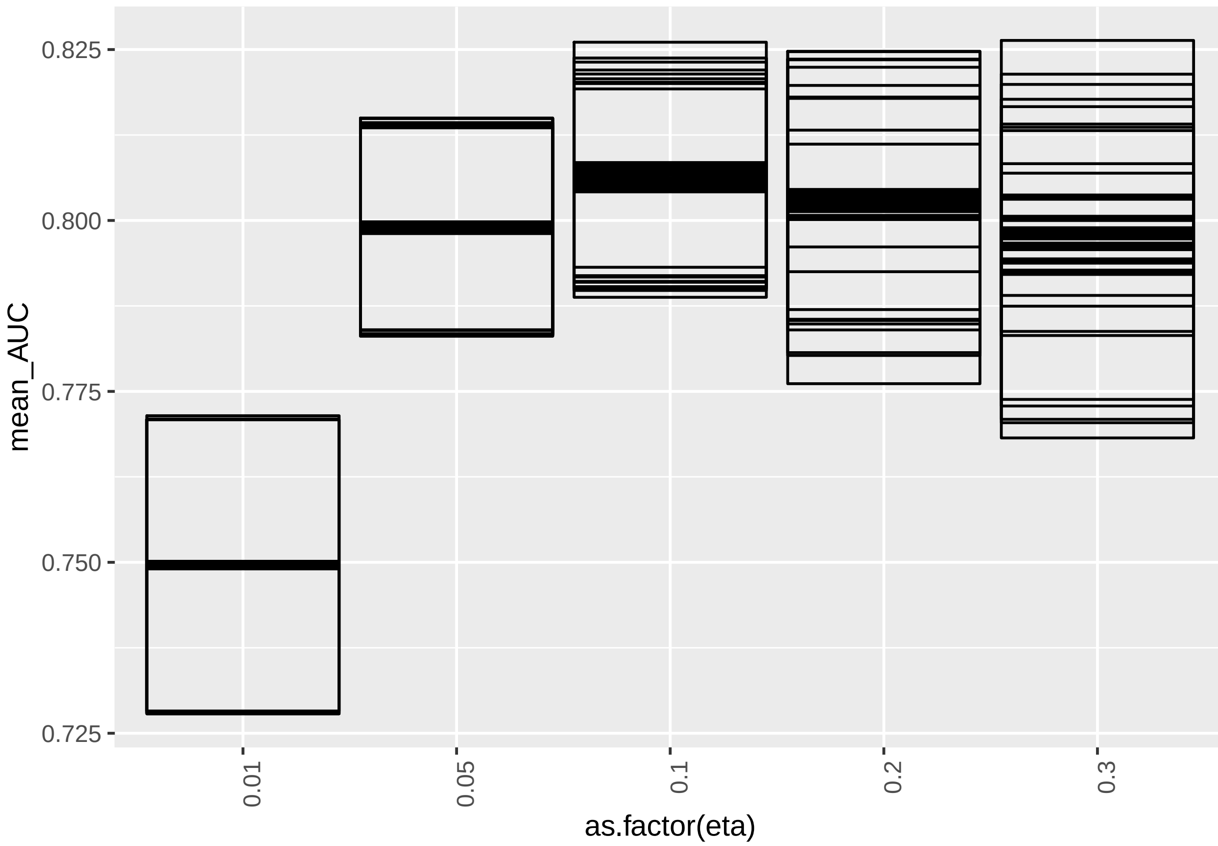

```
# nrounds = 200
ggplot(res_cv_xgboost_cv_results[res_cv_xgboost_cv_results$nrounds == 200, ],
       aes(x = as.factor(eta), y = mean_error,
           ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



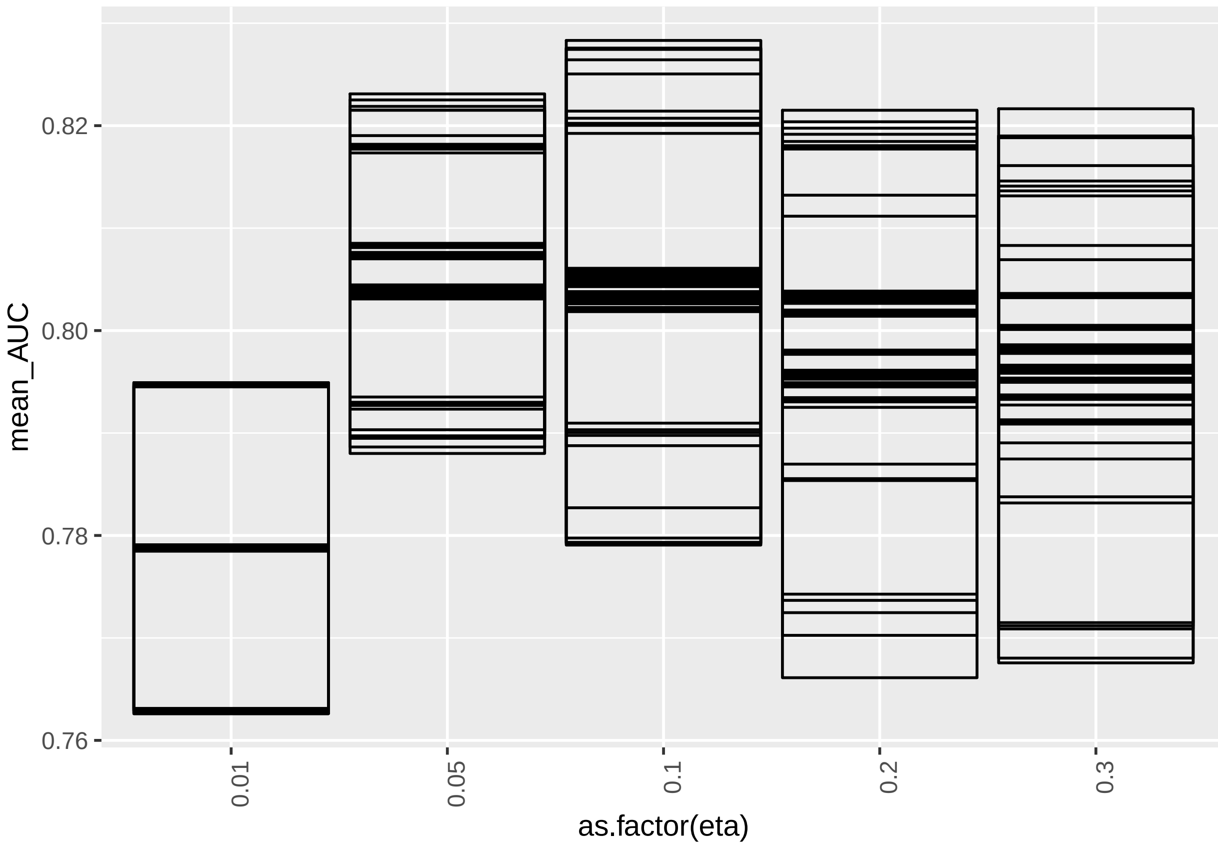

```
# nrounds = 600
ggplot(res_cv_xgboost_cv_results[res_cv_xgboost_cv_results$nrounds == 600, ],
       aes(x = as.factor(eta), y = mean_error,
           ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



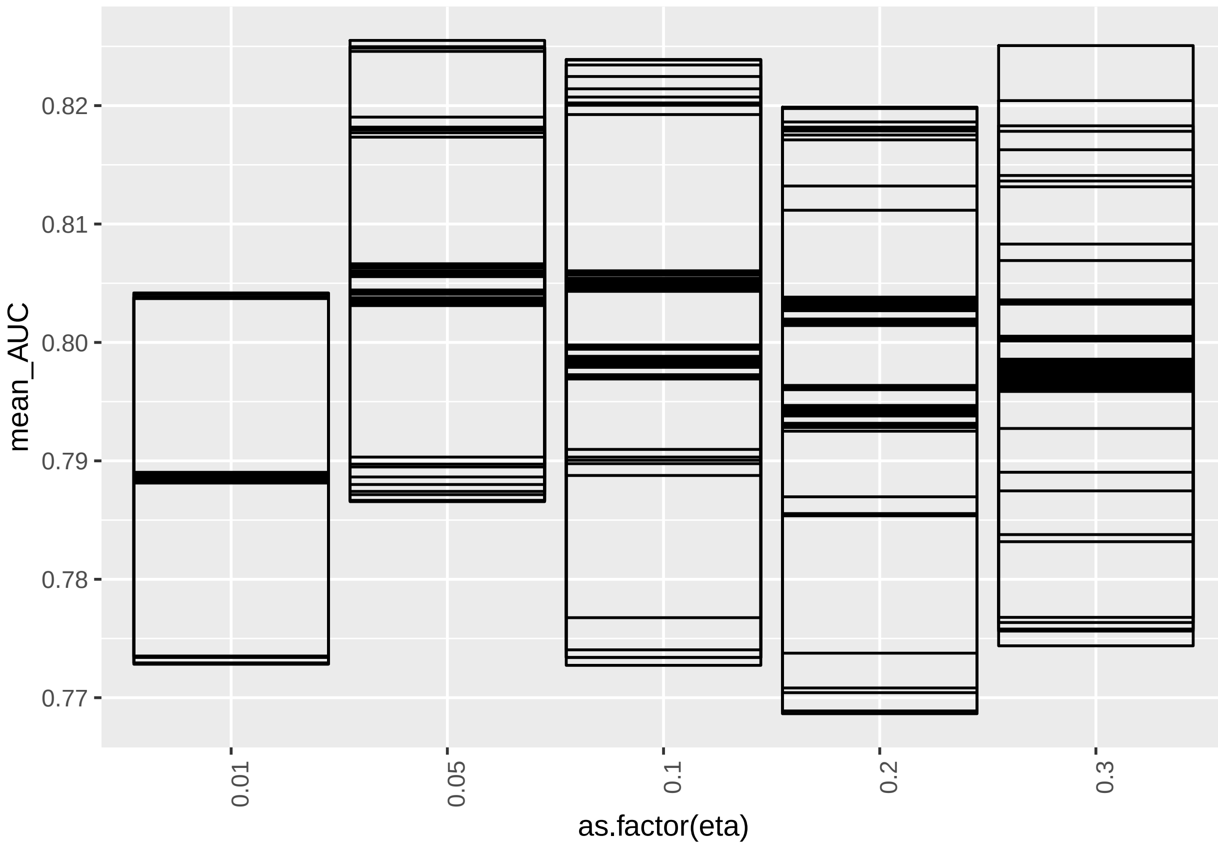

```r
# Mean AUC
# nrounds = 600
ggplot(res_cv_xgboost_cv_results[res_cv_xgboost_cv_results$nrounds == 100, ],
       aes(x = as.factor(eta), y = mean_AUC,
           ymin = mean_AUC - sd_AUC, ymax = mean_AUC + sd_AUC)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```r
# nrounds = 1200
ggplot(res_cv_xgboost_cv_results[res_cv_xgboost_cv_results$nrounds == 200, ],
       aes(x = as.factor(eta), y = mean_AUC,
           ymin = mean_AUC - sd_AUC, ymax = mean_AUC + sd_AUC)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```r
# nrounds = 1800
ggplot(res_cv_xgboost_cv_results[res_cv_xgboost_cv_results$nrounds == 600, ],
       aes(x = as.factor(eta), y = mean_AUC,
           ymin = mean_AUC - sd_AUC, ymax = mean_AUC + sd_AUC)) +
    geom_crossbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



- Choose the "best" parameter value

```
par_best_res_cv_xgboost_cv_results_ind <- which(
  res_cv_xgboost_cv_results$mean_AUC == max(res_cv_xgboost_cv_results$mean_AUC))

par_best_res_cv_xgboost_cv_results_eta <- res_cv_xgboost_cv_results$eta[par_best_res_cv_xgboost_cv_resul
par_best_res_cv_xgboost_cv_results_lambda <- res_cv_xgboost_cv_results$lambda[par_best_res_cv_xgboost_c
par_best_res_cv_xgboost_cv_results_gamma <- res_cv_xgboost_cv_results$gamma[par_best_res_cv_xgboost_cv_
par_best_res_cv_xgboost_cv_results_nrounds <- res_cv_xgboost_cv_results$nrounds[par_best_res_cv_xgboost
```

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

```
if (run.train.xgboost) {
  # training weights
  weight_train <- rep(NA, length(label_train))
  for (v in unique(label_train)){
    weight_train[label_train == v] = 0.5 * length(label_train) / length(label_train[label_train == v])
  }

  if (sample.reweight){
    tm_train_xgboost <- system.time(fit_train_xgboost <- train(features = feature_train, labels = label
                                                                 w =  weight_train,
                                                                 eta_val = par_best_res_cv_xgboost_cv_r
                                                                 lmd = par_best_res_cv_xgboost_cv_resul
```

```r
                                                                    gam = par_best_res_cv_xgboost_cv_resul
                                                                    nr = par_best_res_cv_xgboost_cv_results

  } else {

    tm_train_xgboost <- system.time(fit_train_xgboost <- train(features = feature_train, labels = label_
                                                                    w =  NULL,
                                                                    eta_val = par_best_res_cv_xgboost_cv_r
                                                                    lmd = par_best_res_cv_xgboost_cv_resul
                                                                    gam = par_best_res_cv_xgboost_cv_resul
                                                                    nr = par_best_res_cv_xgboost_cv_results
  }
  save(fit_train_xgboost, tm_train_xgboost, file="../output/fit_train_xgboost.RData")

} else {
  load(file="../output/fit_train_xgboost.RData")
}
```

Step 5: Run test on test images

```r
tm_test_xgboost= NA

if(run.test.xgboost){
  load(file="../output/fit_train_xgboost.RData")
  tm_test_xgboost <- system.time({prob_pred <- predict(fit_train_xgboost, feature_test);
                              label_pred <- ifelse(prob_pred >= 0.5, 1, 0)})
}
```

- Evaluation

```r
## reweight the test data to represent a balanced label distribution

weight_test <- rep(NA, length(label_test))
for (v in unique(label_test)){
  weight_test[label_test == v] = 0.5 * length(label_test) / length(label_test[label_test == v])
}

feature_test <- as.matrix(dat_test[, -6007])
label_test <- as.integer(dat_test$label)
# convert the original 1-2 class into numeric 0s and 1s
label_test <- ifelse(label_test == 2, 0, 1)

accu <- sum(weight_test * (label_pred == label_test)) / sum(weight_test)
tpr.fpr <- WeightedROC(prob_pred, label_test, weight_test)
auc <- WeightedAUC(tpr.fpr)
```

```
## The accuracy of the xgboost model (eta = 0.05, nrounds = 200, lambda = 0.05, gamma = 0) is 71.13684%

## The AUC of the xgboost model (eta = 0.05, nrounds = 200, lambda = 0.05, gamma = 0) is 0.7837474.
```

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.

```
## Time for training xgboost model = 40.903 seconds
```

```
## Time for testing xgboost model = 0.112 seconds
```

Other Models

Principal Components Analysis (PCA) + Support Vector Machines (SVMs)

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
- In this Starter Code, we use logistic regression with LASSO penalty to do classification.

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters.

*Visualize cross-validation results.

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

Step 5: Run test on test images

- Evaluation

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.

Convolutional Neurual Networks

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
- In this Starter Code, we use logistic regression with LASSO penalty to do classification.

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters.

*Visualize cross-validation results.

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

Step 5: Run test on test images

- Evaluation

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.

Random Forests

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
- In this Starter Code, we use logistic regression with LASSO penalty to do classification.

```
library(glmnet)
library(magrittr)
library(WeightedROC)
library(glmnet)
library(ggplot2)
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine

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

```
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.

```r
source("../lib/cross_validation.R")
lmbd = c(1e-3, 5e-3, 1e-2, 5e-2, 1e-1)
feature_train = as.matrix(dat_train[, -6007])
label_train = as.integer(dat_train$label)
if(T){
  res_cv <- matrix(0, nrow = length(lmbd), ncol = 4)
  for(i in 1:length(lmbd)){
    cat("lambda = ", lmbd[i], "\n")
    res_cv[i,] <- cv.function(features = feature_train, labels = label_train, K,
                              l = lmbd[i], reweight = sample.reweight)
  save(res_cv, file="../output/res_cv.RData")
  }
}else{
  load("../output/res_cv.RData")
```

*Visualize cross-validation results.

```r
res_cv <- as.data.frame(res_cv)
colnames(res_cv) <- c("mean_error", "sd_error", "mean_AUC", "sd_AUC")
res_cv$k = as.factor(lmbd)

if(run.cv){
  p1 <- res_cv %>%
    ggplot(aes(x = as.factor(lmbd), y = mean_error,
               ymin = mean_error - sd_error, ymax = mean_error + sd_error)) +
    geom_crossbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

  p2 <- res_cv %>%
    ggplot(aes(x = as.factor(lmbd), y = mean_AUC,
               ymin = mean_AUC - sd_AUC, ymax = mean_AUC + sd_AUC)) +
    geom_crossbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

  print(p1)
  print(p2)
}
```

- Choose the "best" parameter value

```r
par_best <- lmbd[which.min(res_cv$mean_error)] # lmbd[which.max(res_cv$mean_AUC)]
```

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

```r
# training weights
weight_train <- rep(NA, length(label_train))
for (v in unique(label_train)){
  weight_train[label_train == v] = 0.5 * length(label_train) / length(label_train[label_train == v])
}
if (sample.reweight){
  tm_train <- system.time(fit_train <- train(feature_train, label_train, w = weight_train, par_best))
```

```
} else {
  tm_train <- system.time(fit_train <- train(feature_train, label_train, w = NULL, par_best))
}
save(fit_train, file="../output/fit_train.RData")
```

Step 5: Run test on test images

```
tm_test = NA
feature_test <- as.matrix(dat_test[, -6007])
if(run.test){
  load(file="../output/fit_train.RData")
  tm_test <- system.time({label_pred <- as.integer(test(fit_train, feature_test, pred.type = 'class'));
                          prob_pred <- test(fit_train, feature_test, pred.type = 'response')})
}
```

Evaluation

```
## reweight the test data to represent a balanced label distribution
label_test <- as.integer(dat_test$label)
weight_test <- rep(NA, length(label_test))
for (v in unique(label_test)){
  weight_test[label_test == v] = 0.5 * length(label_test) / length(label_test[label_test == v])
}

accu <- sum(weight_test * (label_pred == label_test)) / sum(weight_test)
tpr.fpr <- WeightedROC(prob_pred, label_test, weight_test)
auc <- WeightedAUC(tpr.fpr)


cat("The accuracy of model:", model_labels[which.min(res_cv$mean_error)], "is", accu*100, "%.\n")
cat("The AUC of model:", model_labels[which.min(res_cv$mean_error)], "is", auc, ".\n")
```

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.

Reference(s)

- Du, S., Tao, Y., & Martinez, A. M. (2014). Compound facial expressions of emotion. Proceedings of the National Academy of Sciences, 111(15), E1454-E1462.