Code ▾

Project 3

Spring-2019 Group 10

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

Hide

```
if(!require("EBImage")){
  source("https://bioconductor.org/biocLite.R")
  biocLite("EBImage")
}
if(!require("gbm")){
  install.packages("gbm")
}
if(!require("xgboost")){
  install.packages("xgboost")
}
library("EBImage")
library("gbm")
library(xgboost)
```

Step 0: specify directories.

Set the working directory to the image folder. Specify the training and the testing set. For data without an independent test/validation set, you need to create your own testing data by random subsampling. In order to obain reproducible results, set.seed() whenever randomization is used.

Hide

```
set.seed(2019)
setwd("C:\\Users\\zyang\\Documents\\Spring2019-Proj3-spring2019-proj3-grp10")
# here replace it with your own path or manually set it in RStudio to where this rmd file is located.
# use relative path for reproducibility
```

Provide directories for training images. Low-resolution (LR) image set and High-resolution (HR) image set will be in different subfolders.

Hide

```
train_dir <- "../data/train_set/" # This will be modified for different data sets.
train_LR_dir <- paste(train_dir, "LR/", sep="")
train_HR_dir <- paste(train_dir, "HR/", 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
- (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

Hide

```
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 GBM with different `depth`. In the following chunk, we list, in a vector, setups (in this case, `depth`) 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.

Hide

```
model_values <- seq(3, 11, 2)
model_labels = paste("GBM with depth =", model_values)
model_values_xgboost <- expand.grid(max_depth=c(4,6,8),
                                    eta=c(0.1,0.3,0.5),
                                    subsample=c(0.5,1,1.5),
                                    min_child_weight=c(0.5,1,1.5))
model_labels_xgboost <- paste("XGBoost with depth =",
                              model_values_xgboost$max_depth,
                              "XGBoost with eta =",
                              model_values_xgboost$eta)
##### Tuning paramter: gamma, max_depth, min_child_weight, eta, subsample
```

Step 2: import training images class labels.

We provide extra information of image label: car (0), flower (1), market (2). These labels are not necessary for your model.

Hide

```
extra_label <- read.csv(train_label_path, colClasses=c("NULL", NA, NA))
```

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: a path for low-resolution images. + Input: a path for high-resolution images. + Output: an RData file that contains extracted features and corresponding responses

`r` r source(../lib/feature.R)

tm_feature_train <- NA if(run.feature.train){ tm_feature_train <- system.time(dat_train <- feature(train_LR_dir, train_HR_dir)) feat_train <- dat_train$featurelabel_train < -dat_train$label }

save(dat_train, file=../output/feature_train.RData)

Step 4: Train a regression 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 path that points to the training set features and responses. + 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 classifier. + Output: an R object of response predictions on the test set. If there are multiple classifiers under evaluation, there should be multiple sets of label predictions.

`r` r source(../lib/train.R) source(../lib/test.R) source(../lib/train_xgboost.R)

Model selection with cross-validation

GBM model

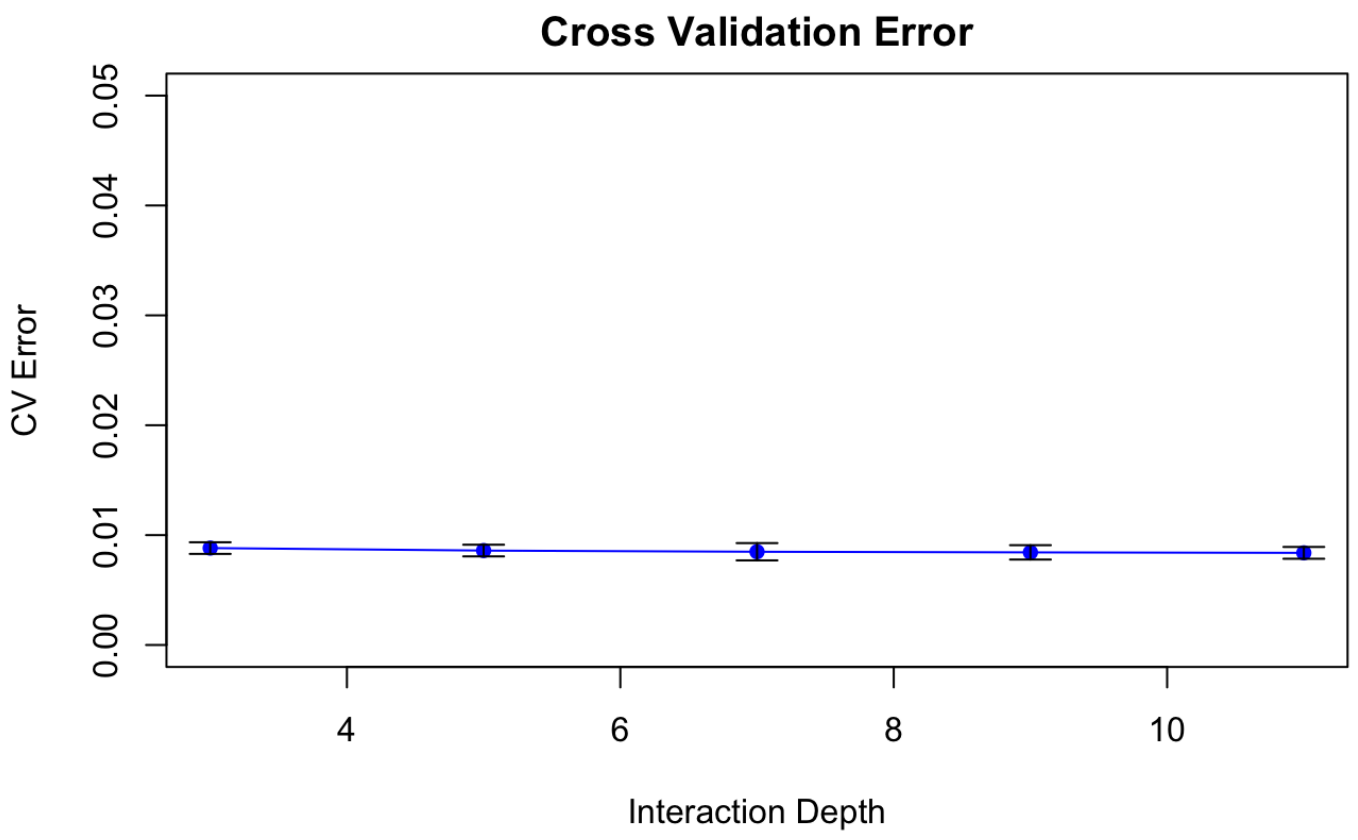
- Do model selection by choosing among different values of training model parameters, that is, the interaction depth for GBM in this example.

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

```
k= 1
k= 2
k= 3
k= 4
k= 5
```

Visualize cross-validation results.

`r` r if(run.cv){ load(../output/err_cv.RData) plot(model_values, err_cv[,1], xlab=Depth, ylab=Error, main=Validation Error, type=, ylim=c(0, 0.05)) points(model_values, err_cv[,1], col=, pch=16) lines(model_values, err_cv[,1], col=) arrows(model_values, err_cv[,1]-err_cv[,2], model_values, err_cv[,1]+err_cv[,2], length=0.1, angle=90, code=3) }



- Choose the “best”" parameter value

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

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

Hide

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

XGBoost model

- Do model selection by choosing among different values of training model parameters, that is, the interaction depth for GBM in this example.

Hide

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

if(run.cv){
  err_cv_xgboost <- array(dim=c(nrow(model_values_xgboost), 2))
  for(k in 1:nrow(model_values_xgboost)){
    cat("k=", k, "\n")
    err_cv_xgboost[k,] <- cv.function_xgboost(feat_train, label_train, model_values_xgboost[k,], K)
  }
  save(err_cv_xgboost, file="../output/err_cv_xgboost.RData")
}
```

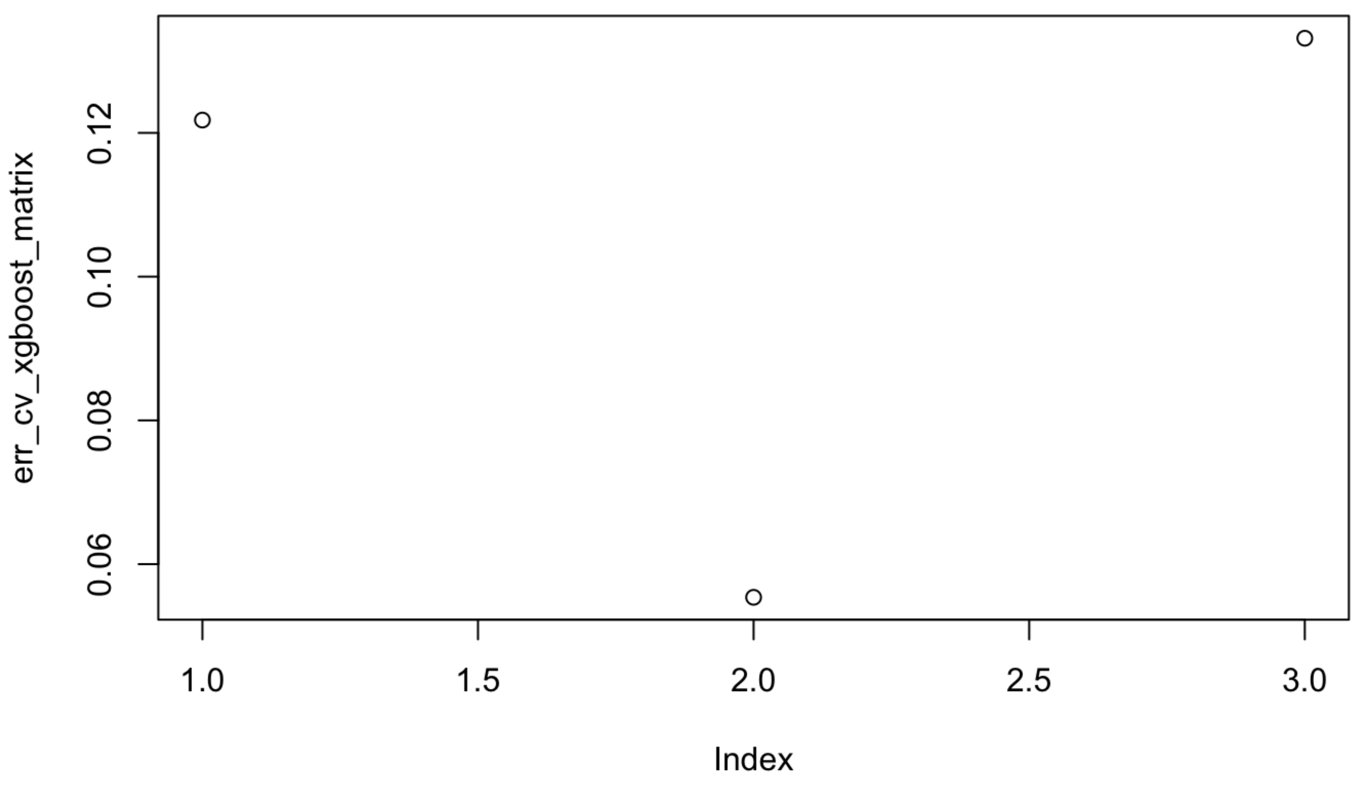
Visualize cross-validation results.

Hide

```
if(run.cv){
  load("../output/err_cv_xgboost.RData")
  err_cv_xgboost_matrix <- matrix(err_cv_xgboost[,1],
                                  ncol=3,
                                  byrow = TRUE)
  colnames(err_cv_xgboost_matrix) <- c(4,6,8)
  err_cv_xgboost_matrix <- apply(err_cv_xgboost_matrix,2,mean)

  #### plot
  plot(err_cv_xgboost_matrix)
}
```

```
data length [5] is not a sub-multiple or multiple of the number of rows [2]
```



- Choose the “best”" parameter value

Hide

```
model_best_xgboost=model_values_xgboost[1]
if(run.cv){
  model_best_xgboost <- model_values_xgboost[which.min(err_cv_xgboost[,1]),]
}

par_best_xgboost <- list(max_depth=model_best_xgboost$max_depth,
                         eta=model_best_xgboost$eta,
                         subsample=model_best_xgboost$subsample,
                         min_child_weight=model_best_xgboost$min_child_weight)
```

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

Hide

```
tm_train_xgboost <- NA
tm_train_xgboost <- system.time(fit_train_xgboost <- train_xgboost(feat_train, label_train, par_best_xgbo
ost))
save(fit_train_xgboost, file="../output/fit_train_xgboost.RData")
```

Step 5: Super-resolution for test images

Hide

```
source("../lib/superResolution.R")
source("../lib/psnr.R")
```

GBM model

Feed the final training model with the completely holdout testing data. + `superResolution.R` + Input: a path that points to the folder of low-resolution test images. + Input: a path that points to the folder (empty) of high-resolution test images. + Input: an R object that contains tuned predictors. + Output: construct high-resolution versions for each low-resolution test image.

Hide

```
test_dir <- "../data/test_set/" # This will be modified for different data sets.
test_LR_dir <- paste(test_dir, "LR/", sep="")
test_HR_dir <- paste(test_dir, "HR/", sep="")

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

XGBoost model

Feed the final training model with the completely holdout testing data. + `superResolution.R` + Input: a path that points to the folder of low-resolution test images. + Input: a path that points to the folder (empty) of high-resolution test images. + Input: an R object that contains tuned predictors. + Output: construct high-resolution versions for each low-resolution test image.

Hide

```
test_dir_xgboost <- "../data/test_set/" # This will be modified for different data sets.
test_LR_dir_xgboost <- paste(test_dir, "LR/", sep="")
test_HR_dir_xgboost <- paste(test_dir, "HR_xgboost/", sep="")

tm_test_xgboost <- NA
if(run.test){
  load(file="../output/fit_train_xgboost.RData")
  tm_test_xgboost <- system.time(superResolution(test_LR_dir_xgboost,
                                                  test_HR_dir_xgboost,
                                                  fit_train_xgboost))
}
```

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.

Hide

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

```
Time for constructing training features= 2842.52 s
```

Hide

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

GBM model

Hide

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

```
Time for training model= 1536.47 s
```

Hide

```
cat("Time for super-resolution=", tm_test[1], "s \n")
```

```
Time for super-resolution= 625.23 s
```

Hide

```
#psnr_gbm <- compute_psnr(test_HR_dir,train_HR_dir)
psnr_gbm
```

```
[1] 23.7681
```

XGBoost model

Hide

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

```
Time for training model= 16.93 s
```

Hide

```
cat("Time for super-resolution=", tm_test_xgboost[1], "s \n")
```

```
Time for super-resolution= 115.53 s
```

Hide

```
#psnr_xgboost <- compute_psnr(test_HR_dir_xgboost,train_HR_dir)
psnr_xgboost
```

```
[1] 22.27632
```