Code

Project 3 - Group 6 Main Script (Xgboost Model + Baseline)

Jingwen Wang and Ziyi Liao (credit to Chengliang Tang, Tian Zheng)

This file contains all complete steps and models that we have for this project. It includes baseline, baseline improvement and Xgboost model.

Hide

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

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(2018)
#setwd("./ads_fall2018_proj3")
# 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 = F # run cross-validation on the training set
run.cv.imp = F # run improved cross-validation
run.xgb = T # run xgboost model
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
if(run.xgb){
  library("xgboost")
}
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications.

Our previous test using GBM with different `depth` (k = 3, 5, 7, 9, 11). In the following chunk, we list, in a vector, setups (in this case, `depth`) corresponding to models that we will compare.

In the Baseline Improvement Model, we also tune parameter `shrinkage` and we try `depth` (k = 9,11). Ideally, we should have tried deeper depth (i.e. 13, 15, 17 etc).

In the Xgboost Model, we use parameter `depth` (k = 6,7,8) and `shrinkage` (eta = 0.6, 0.7, 0.8, 0.9).

Model Setup: Baseline Model:

Hide

```
if(run.cv){
  model_values <- seq(3, 11, 2)
  model_labels = paste("GBM with depth =", model_values)
}
```

Model Setup: Baseline Improvement:

Hide

```
if(run.cv.imp){
  model_values <- list(depth=seq(9,11,2), lr=c(.1, .01, .001))
  model_labels = paste("GBM with depth =", rep(model_values$depth, rep(3,2)), ', shri
nkage =', rep(model_values$lr, 4))

}
```

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

Hide

```
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$feature
  label_train <- dat_train$label
}

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

Feature Train Time

Hide

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

```
Time for constructing training features= 170.869 s
```

Hide

```
# Time for constructing training features= 131.276 s
```

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

Call the train model and test model from library.

`train_base.R` and `test_base.R` should be wrappers for all your model training steps and your classification/prediction steps. This two functions are used for baseline models.

- `train_base_imp.R`
- Input: a path that points to the training set features and responses.
- Output: an RData (or RDS) file that contains trained classifiers in the forms of R objects: models/settings/links to external trained configurations.
- `test_base.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.

Notes: `+ train_base_imp` considers about `depth` and `shrinkage` , and we set `n.tree = 1000` instead of `200` . We tried to improve the efficiency by paralleling the algorithm.

Hide

```
if(run.cv){
  source("../lib/train_base.R")
  source("../lib/test_base.R")
}
```

Hide

```
if(run.cv.imp){
  source("../lib/train_base_imp.R")
  source("../lib/test_base.R")
}
```

Hide

```
# Xgboost Function
if(run.xgb){
  source("../lib/xgb_train.R")
  source("../lib/test_xgb.R")
}
```

Model selection with cross-validation:

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

Hide

```
if(run.cv){
  source("../lib/cross_validation_base.R")}
if(run.cv.imp){
  source("../lib/cross_validation_base.R")}
if(run.xgb){
  source("../lib/xgb_cv.R")
}
```

Model 1: Baseline 0 GBM + Cross-validation:

Hide

```
# Initial Baseline
if(run.cv){
  err_cv <- array(dim=c(length(model_values), 4))
  for(k in 1:length(model_values)){
    cat("k=", k, "\n")
    err_cv[k,] <- cv.function(feat_train, label_train, model_values[k], K)
  }
  save(err_cv, file="../output/err_cv.RData")
}
```

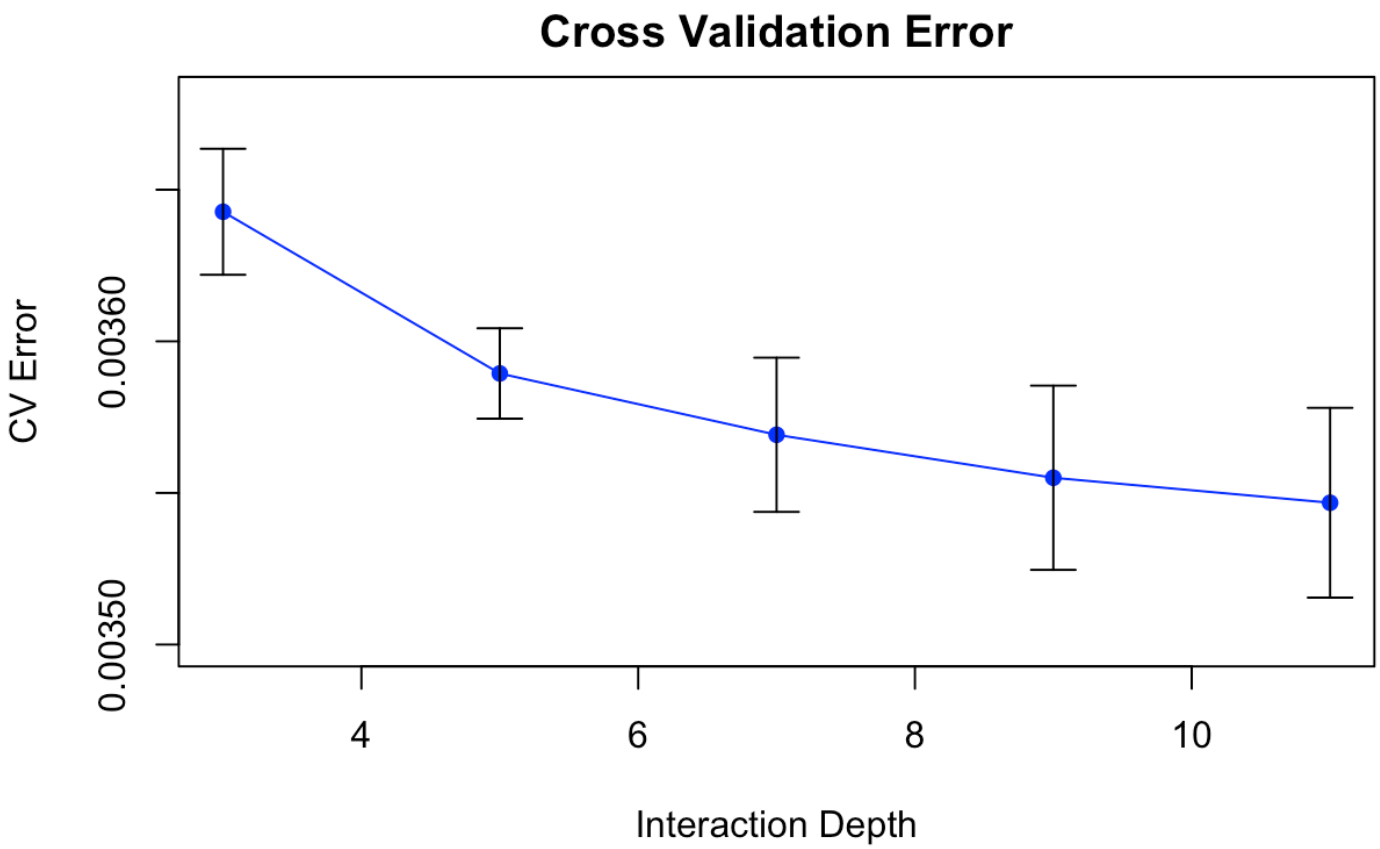
Hide

```
# Improved Baseline
if(run.cv.imp){
  err_cv <- array(dim=c(length(model_values[[1]])* length(model_values[[2]]), 4))
  for(k in 1:length(model_values[[1]])){
    cat('k=', k, '\n')
    for(j in 1:length(model_values[[2]])) {
      cat('shrinkage=', model_values[[2]][j], '\n')
      par <- list(d=model_values[[1]][k], lr=model_values[[2]][j])
      err_cv[(k-1)*length(model_values[[2]]) + j,] <- cv.function(feat_train, label_t
rain, par, K)
    }
  }
  save(err_cv, file="../output/err_cv_imp.RData")
}
```

Visualize cross-validation results of inital Baseline Model.

Hide

```
if(run.cv){
  load("../output/err_cv.RData")
  plot(model_values, err_cv[,1], xlab="Interaction Depth", ylab="CV Error",
       main="Cross Validation Error", type="n", ylim=c(0.0035, 0.00368))
  points(model_values, err_cv[,1], col="blue", pch=16)
  lines(model_values, err_cv[,1], col="blue")
  arrows(model_values, err_cv[,1]-err_cv[,2], model_values, err_cv[,1]+err_cv[,2],
         length=0.1, angle=90, code=3)
}
```



Hide

```
# Printed As Table
colnames(err_cv) <- c("cv train error","cv error std", "PSNR", "PSNR std")
rownames(err_cv) <- c("k=3","k=5","k=7","k=9","k=11")
err_cv
```

```
     cv train error cv error std     PSNR   PSNR std
k=3     0.003642728 2.077155e-05 24.38579 0.02476166
k=5     0.003589409 1.491194e-05 24.44980 0.01804309
k=7     0.003569186 2.542561e-05 24.47440 0.03081050
k=9     0.003555009 3.037937e-05 24.49172 0.03717331
k=11    0.003546770 3.128800e-05 24.50180 0.03816503
```

- Choose the “best”" parameter value by initial baseline model:

Hide

```
if(run.cv){
  model_best=model_values[1]
  model_best <- model_values[which.min(err_cv[,1])]
}
par_best <- list(depth=model_best) # depth at 11
```

Result from Improved Baseline Model: Lower Error Rate

Hide

```
if(run.cv.imp){
  load("../output/err_cv_lr.RData")
  # Depth = 9
  dep.9 <- err_cv[1:3,]
  rownames(dep.9) <- c("lr=0.1","lr=0.01","lr=0.001")
  colnames(dep.9) <- c("error mean","error std", "PSNR mean", "PSNR std")
  print(dep.9)
  # Depth = 11
    dep.11 <- err_cv[4:6,]
    rownames(dep.11) <- c("lr=0.1","lr=0.01","lr=0.001")
    colnames(dep.11) <- c("error mean","error std", "PSNR mean", "PSNR std")
    print(dep.11)
}
```

```
          error mean    error std PSNR mean    PSNR std
lr=0.1   0.002744741 8.648087e-05  25.61669 0.135704996
lr=0.01  0.002654278 1.468446e-05  25.76059 0.023969417
lr=0.001 0.002974204 4.599424e-06  25.26630 0.006715872
          error mean    error std PSNR mean    PSNR std
lr=0.1   0.003367471 1.565176e-03  25.01098 1.627894458
lr=0.01  0.002646186 1.250848e-05  25.77384 0.020569086
lr=0.001 0.002954702 6.194176e-06  25.29487 0.009095478
```

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

Hide

```
if(run.cv){
  tm_train=NA
  tm_train <- system.time(fit_train <- train(feat_train, label_train, par_best))
  saveRDS(fit_train, file="../output/fit_train_0.RDS")
}
```

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

Hide

```
if(run.cv.imp){
  par_best <- list(depth = 11,lr=0.01) # Parameter From Improved Baseline
  tm_train=NA
  tm_train <- system.time(fit_train <- train(feat_train, label_train, par_best))
  saveRDS(fit_train, file="../output/fit_train.RDS")
}
```

Hide

```
cat("Time for initial training model=", tm_train[1], "s \n")
# Time for training improved baseline model= 57.077 s
# (we run this on Google Cloud)
```

Model 2: Xgboost Model:

Hide

```
if(run.xgb){
  xgb.param.all <- xgb.parameters(feat_train,label_train,5)
  save(xgb.param.all,file="../output/xgb_par.RData")
}
```

** Xgboost Results Demonstration**:

Hide

```
if(run.xgb){
  load("../output/xgb_par.RData")
}
# Best Parameters
cat("Best eta = ",xgb.param.all[[1]]$eta,"\n")
```

```
Best eta =  0.9
```

Hide

```
cat("Best depth = ",xgb.param.all[[1]]$depth,"\n")
```

```
Best depth =  8
```

Hide

```
# MSE Results
cat("MSE =",mean(xgb.param.all[[2]][,1]),"\n")
```

```
MSE = 0.05251038
```

Hide

```
MSE <- mean(xgb.param.all[[2]][,1])
cat("PSNR =",-10*log10(MSE),"\n")
```

```
PSNR = 12.79755
```

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

Hide

```
if(run.xgb){
  tm_train=NA
  par_best <- xgb.param.all$best.param
  tm_train <- system.time(fit_train <- train.xgboost(feat_train, label_train,par_best
))
  save(fit_train, file="../output/xgb_fit.RData")
}
```

Step 5: Super-resolution for test images:

In this file, we are using **Xgboost** model to do the super resolution due to the time efficiency.

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

```
source("../lib/superResolution.R")
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/xgb_fit.RData")
  tm_test <- system.time(superResolution(test_LR_dir, test_HR_dir, fit_train))
}
```

Hide

```
if(run.xgb){
  cat("Time for super-resolution=", tm_test[1], "s \n")}
```

```
Time for super-resolution= 3.23 s
```

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")
cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
cat("Time for training model=", tm_train[1], "s \n")
cat("Time for super-resolution=", tm_test[1], "s \n")
```