Project 3 - Example Main Script

Code

Chao Yin

Loading Libraries

Hide

```
if(!require('EBImage')){
  source('https://bioconductor.org/biocLite.R')
  biocLite('EBImage')
}
if(!require('gbm')){
  install.packages('gbm')
}
if(!require('foreach')){
  install.packages('foreach')
}
if(!require('doParallel')){
  install.packages('doParallel')
}
if(!require('abind')){
  install.packages('abind')
}
if(!require('tidyverse')){
  install.packages('tidyverse')
}
library('EBImage')
library('gbm')
library('foreach')
library('doParallel')
library('abind')
library('tidyverse')
```

Step 0: specify directories.

Hide

```
set.seed(2018)
# setwd("../doc")
# here replace it with your own path or manually set it in RStudio to where this rmd file is loc
ated.
# use relative path for reproducibility
```

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=FALSE # run cross-validation on the training set
K <- 10  # number of CV folds
run.feature.train=FALSE # 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 10-fold-cross-validation, we compare the performance of models with different specifications. We use GBM with `depth` of 3, 6, 9, 12 and 15, and `learning rate` of 1e-1, 1e-2, 1e-3, 1e-4. In the following chunk, we list, in a vector, setups (`depth` and `learning rate`) corresponding to models that we will compare.

Hide

```
model_values <- list(depth = seq(3, 15, 3), lr = 10^c(-1:-4))
model_labels = paste('GBM with depth =', rep(model_values$depth, rep(4,5)), ', learning rate =',
rep(model_values$lr, 5))
```

Step 2: import training images class labels.

We provide extra information of image label: car (0), flower (1), market (2).

Hide

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

Step 3: construct features and responses

- `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")
if(!file.exists('../output/feature_train.RData')){
  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')
}else{
  dat_train <- get(load('../output/feature_train.RData'))
  feat_train <- dat_train$feature
  label_train <- dat_train$label
}
```

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

Call the train model and test model from library.

- `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.

Hide

```
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, that is, the interaction depth for GBM in this example. (This part runs on GCP)

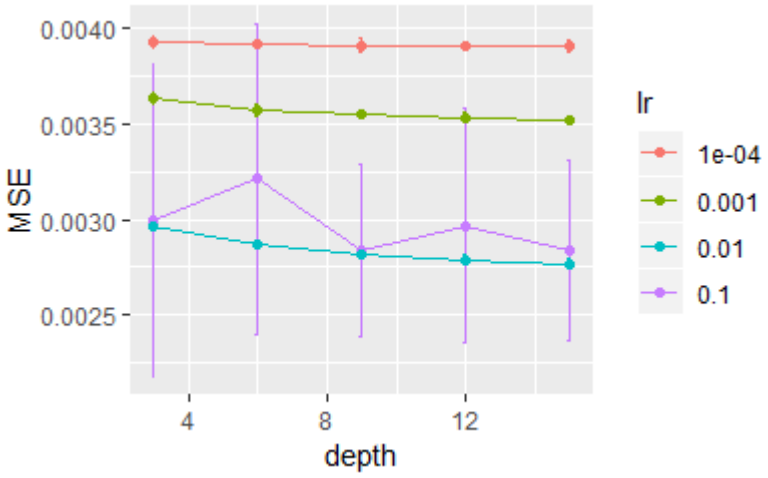
Hide

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

if(run.cv){
  err_cv <- array(dim=c(length(model_values[[1]])*length(model_values[[2]]), 3))
  for(k in 1:length(model_values[[1]])){
    cat("depth=", model_values[[1]][k], "\n")
    for(j in 1:length(model_values[[2]])){
      cat("learning rate=", 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_train, par,
 K)
    }
  }
  save(err_cv, file="../output/err_cv.RData")
}
```

Visualize cross-validation results.

Hide

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

  cv_Err <- data.frame(depth = rep(model_values$depth, rep(4,5)), lr = factor(rep(model_values$l
r, 5)), MSE = err_cv[,1], SE = err_cv[,2])
  cv_Err %>%
  ggplot(aes(x=depth, y=MSE, colour = lr, group=lr)) +
    geom_errorbar(aes(ymin=MSE-SE, ymax=MSE+SE), width=.1) +
    geom_line() +
    geom_point()
}
```



- Choose the “best”" parameter value

Hide

```
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. (This part runs on GCP)

Hide

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

Step 5: Super-resolution for test images

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){
  fit_train <- readRDS(file="../output/fit_train.RDS")
  tm_test <- system.time(superResolution(test_LR_dir, test_HR_dir, fit_train))
}
```

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

```
load(file='../output/tm_feature_train.RData')
load(file='../output/tm_train.RData')
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
```

```
Time for constructing training features= 51.77 s
```

Hide

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

```
Time for training model= 27.526 s
```

Hide

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