Code

Project 3 - Main Script (Baseline Model - GBM)

Group 8

This is the report for our baseline model. In this `main.Rmd`, we use Boosted Decision Stumps to predict neighborhood pixels in order to enhance the resolution of blurry and low-resolution images.

Note: please use Mac OS system to reproduce this file on your computer successfully.

Hide

```
# load required packages
if(!require("EBImage")){
  source("https://bioconductor.org/biocLite.R")
  biocLite("EBImage")
}
if(!require("gbm")){
  install.packages("gbm")
}
if(!require("plyr")){
  install.packages("plyr")
}
if(!require("doMC")){
  install.packages("doMC")
}
library("EBImage")
library("gbm")
library("plyr")
library("doMC")
```

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 obtain reproducible results, `set.seed()` whenever randomization is used.

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="")
```

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="")
```

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

```
model_values <- seq(1, 4, 1)
#model_values <- 3
model_labels = paste("XGBoost with depth =", model_values)
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications. In this file, we use GBM with depth one and compare between different values of `shrinkage` and `number of trees`. In the following chunk, we list, in a vector, setups corresponding to models that we will compare - three `shrinkage` values and three `numbers of trees` values. Therefore, we will compare $3 \times 3 = 9$ models in the cross validation part.

Hide

```
# shrinkage
shrinkage_values <- c(0.001, 0.01, 0.1)
# number of trees
ntrees_values <- c(100,200,300)
# nine model labels
model_labels <- c()
for (shrink in shrinkage_values){
    model_labels <- c(model_labels,
                      paste("gbm with shrinkage rate=", shrink,
                            "number of trees=", ntrees_values))
    }
```

Step 2: 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")
save(tm_feature_train, file="../output/tm_feature_train.RData")
```

Step 3: Train a classification model with training images

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.

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 `shrinkage` and `number of trees` for GBM in this file.

Hide

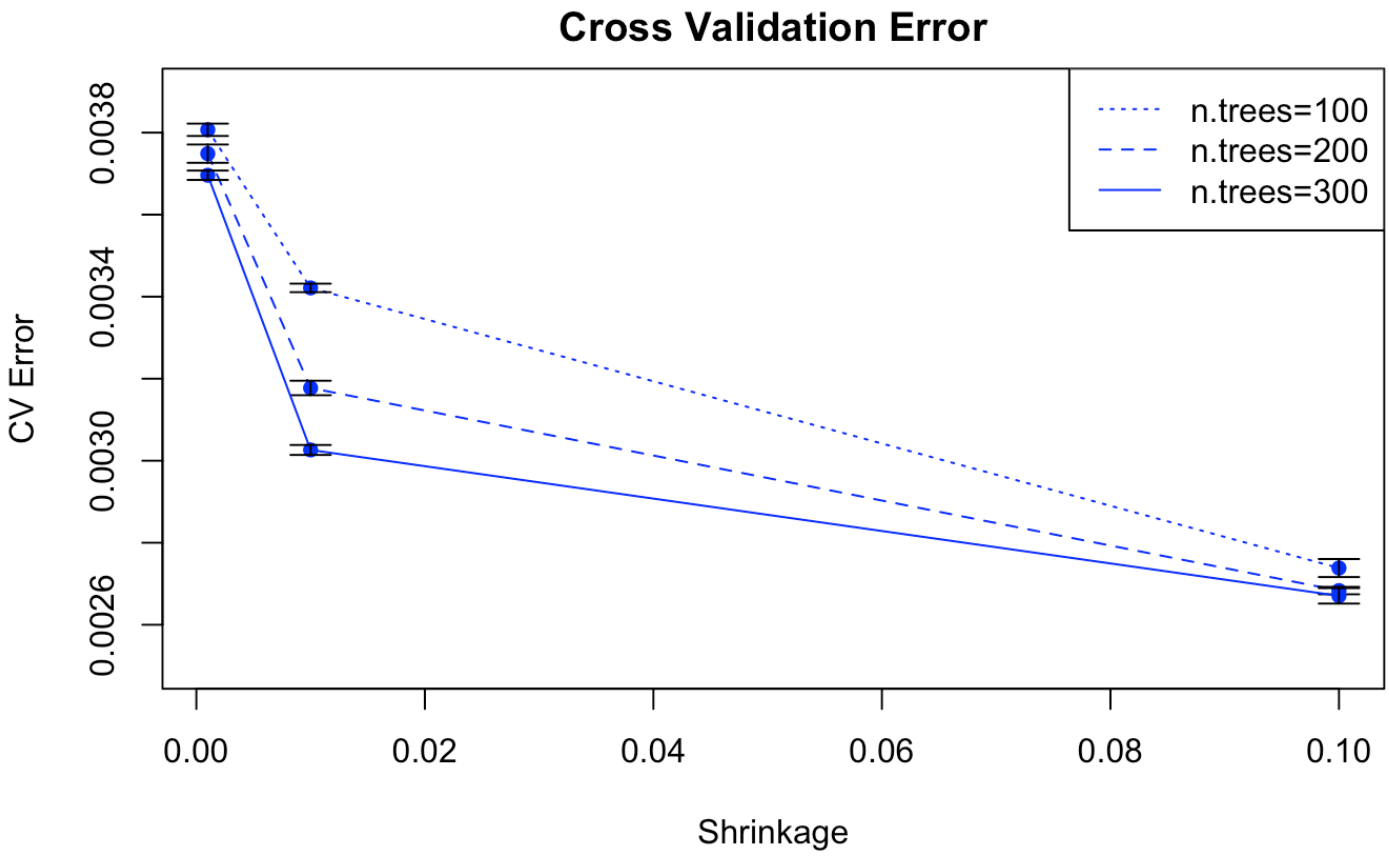
```
set.seed(2018)
source("../lib/cross_validation.R")
load(file="../output/feature_train.RData")
feat_train <- dat_train$feature
label_train <- dat_train$label
# Ramdomly choose 500 images for a K-fold cross validation
img_cv <- sort(sample(1500, 500, replace = FALSE)) # full model
#img_cv <- sort(sample(1200, 500, replace = FALSE)) # local test
ind_cv <- unlist(lapply(img_cv, function(x) {((x-1)*1000+1) : (x*1000)}))
feat_train_cv <- feat_train[ind_cv, ,]
label_train_cv <- label_train[ind_cv, ,]
if(run.cv){
  # create a list to store the cross validation time
  tm_cv <- list()
  # create an array to store the cross valifation results - mse and standard error
  err_cv <- array(dim=c(length(shrinkage_values)*length(ntrees_values), 2))
  # iterate through 9 values for tunning parameters
  for(k in 1:length(shrinkage_values)){
    for (j in 1:length(ntrees_values)) {
      cat("shrinkage = ", shrinkage_values[k], "ntrees = ", ntrees_values[j], "\n")
      tm_cv[[(k-1)*3+j]] <- system.time(err_cv[(k-1)*3+j,] <-
                                          cv.function(feat_train, label_train,
                                          par=list(shrink=shrinkage_values[k],
                                                     n.trees=ntrees_values[j]),
                                          K))
        cat("time for current cv session = ", tm_cv[[(k-1)*3+j]][3], "\n")
     }
    }
  save(err_cv, file="../output/err_cv.RData")
}
```

```
shrinkage =  0.001 ntrees =  100
time for current cv session =  3143.986
shrinkage =  0.001 ntrees =  200
time for current cv session =  6335.012
shrinkage =  0.001 ntrees =  300
time for current cv session =  8291.618
shrinkage =  0.01 ntrees =  100
time for current cv session =  2842.561
shrinkage =  0.01 ntrees =  200
time for current cv session =  5578.975
shrinkage =  0.01 ntrees =  300
time for current cv session =  8312.135
shrinkage =  0.1 ntrees =  100
time for current cv session =  3178.171
shrinkage =  0.1 ntrees =  200
time for current cv session =  5830.398
shrinkage =  0.1 ntrees =  300
time for current cv session =  10778.83
```

Visualize cross-validation results.

Hide

```
if(run.cv){
  load("../output/err_cv.RData")
  plot(shrinkage_values, err_cv[c(1,5,9),1], xlab="Shrinkage", ylab="CV Error",
       main="Cross Validation Error", type="n", ylim=c(0.0025, 0.0039))
  for (i in c(2,1,0)) {
    points(shrinkage_values, err_cv[((1:3)*3-i),1], col="blue", pch=16)
    lines(shrinkage_values, err_cv[((1:3)*3-i),1], col="blue", lty=i+1)
    arrows(shrinkage_values, err_cv[((1:3)*3-i),1]-err_cv[((1:3)*3-i),2], shrinkage_v
alues,
           err_cv[((1:3)*3-i),1]+err_cv[((1:3)*3-i),2], length=0.1, angle=90, code=3)
  }
  legend("topright", legend=c("n.trees=100","n.trees=200","n.trees=300"),
         lty=c(3,2,1), col="blue")
}
```



- Choose the "best" parameter value according to one standard error rule

Hide

```
#model_best=model_values[1]
# if(run.cv){
#   best_ind <- which.min(err_cv[,1])
# }
# according to one stand error rule
best_ind <- 8
best_ntrees_ind <- (best_ind-1)%%length(shrinkage_values)+1
best_shrink_ind <- (best_ind-best_ntrees_ind)%/%length(shrinkage_values)+1
par_best <- list(shrink=shrinkage_values[best_shrink_ind],
                 n.trees=ntrees_values[best_ntrees_ind])
print(par_best)
```

```
$shrink
[1] 0.1

$n.trees
[1] 200
```

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

Hide

```
set.seed(2018)
tm_train=NA
tm_train <- system.time(fit_train <- train(feat_train, label_train, par_best))
save(fit_train, file="../output/fit_train.RData")
save(tm_train, file="../output/tm_train.RData")
```

Step 4: 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){
  load(file="../output/fit_train.RData")
  tm_test <- system.time(superResolution(test_LR_dir, test_HR_dir, fit_train))
}
save(tm_test, file="../output/tm_test.RData")
```

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("../output/tm_feature_train.RData")
cat("Time for constructing training features=", tm_feature_train[3], "s \n")
```

```
Time for constructing training features= 204.918 s
```

Hide

```
load("../output/tm_train.RData")
cat("Time for training model=", tm_train[3], "s \n")
```

```
Time for training model= 1612.68 s
```

Hide

```
load("../output/tm_test.RData")
cat("Time for super-resolution=", tm_test[3], "s \n")
```