Code

Project 3 - Main Script (Improved Model - XGBoost)

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("xgboost")
}
if(!require("doMC")) {
  install.packages("doMC")
}
library("EBImage")
library("xgboost")
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

```
# set work directory
set.seed(2018)
setwd("../doc")
```

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 file, we use XGBoost with different `max_depth` and `min_child_weight` (i.e., minimum sum of instance weight needed in a child). In the following chunk, we list, in a vector, setups corresponding to models that we will compare - three `max_depth` values and three `min_child_weight` values. Therefore, we will compare $3 \times 3 = 9$ models in the cross validation part.

Hide

```
depth_values <- c(1, 3, 5)
child_weight_values <- c(1, 3, 5)
model_labels <- c()
for (d in depth_values) {
  model_labels <- c(model_labels,
                    paste("xgboost with maximum depth = ", d,
                          "minimum child weight = ", child_weight_values))
  }
```

Step 2: import training images class labels.

We utilize extra information of image label: car (0), flower (1), market (2) in the XGBoost model.

Hide

```
extra_label <- read.csv(train_label_path, colClasses=c("NULL", NA, NA))
train_file_0 <- which(extra_label$Label==0)
train_file_1 <- which(extra_label$Label==1)
train_file_2 <- which(extra_label$Label==2)
list_file <- list(file0=train_file_0, file1=train_file_1, file2=train_file_2)
```

Step 3: construct features and responses

`feature_keypoint.R` is the wrapper for feature engineering functions and options. To improve sampling efficiency, we use a 3*3 Laplacian filter to detect the keypoints in images, and then give more sampling weight to points with high rate of color change.

- `feature_keypoint.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_keypoint.R")
tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(keypoint_train <- feature_keypoint(train_LR_dir, tr
ain_HR_dir))
  feat_train <- keypoint_train$feature
  label_train <- keypoint_train$label
}
save(keypoint_train, file="../output/keypoint_feature_train.RData")
save(tm_feature_train, file="../output/tm_keypoint_feature_train.RData")
```

Step 4: Train a classification model with training images

Call the train model and test model from library.

`train_xg.R` and `test_xg.R` should be wrappers for all your model training steps and your classification/prediction steps.

- `train_xg.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_xg.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_xg.R")
source("../lib/test_xg.R")
```

Model selection with cross-validation

** Do model selection by choosing among different values of training model parameters, that is, the `max_depth` and `min_child_weight` for XGBoost in this file.

Hide

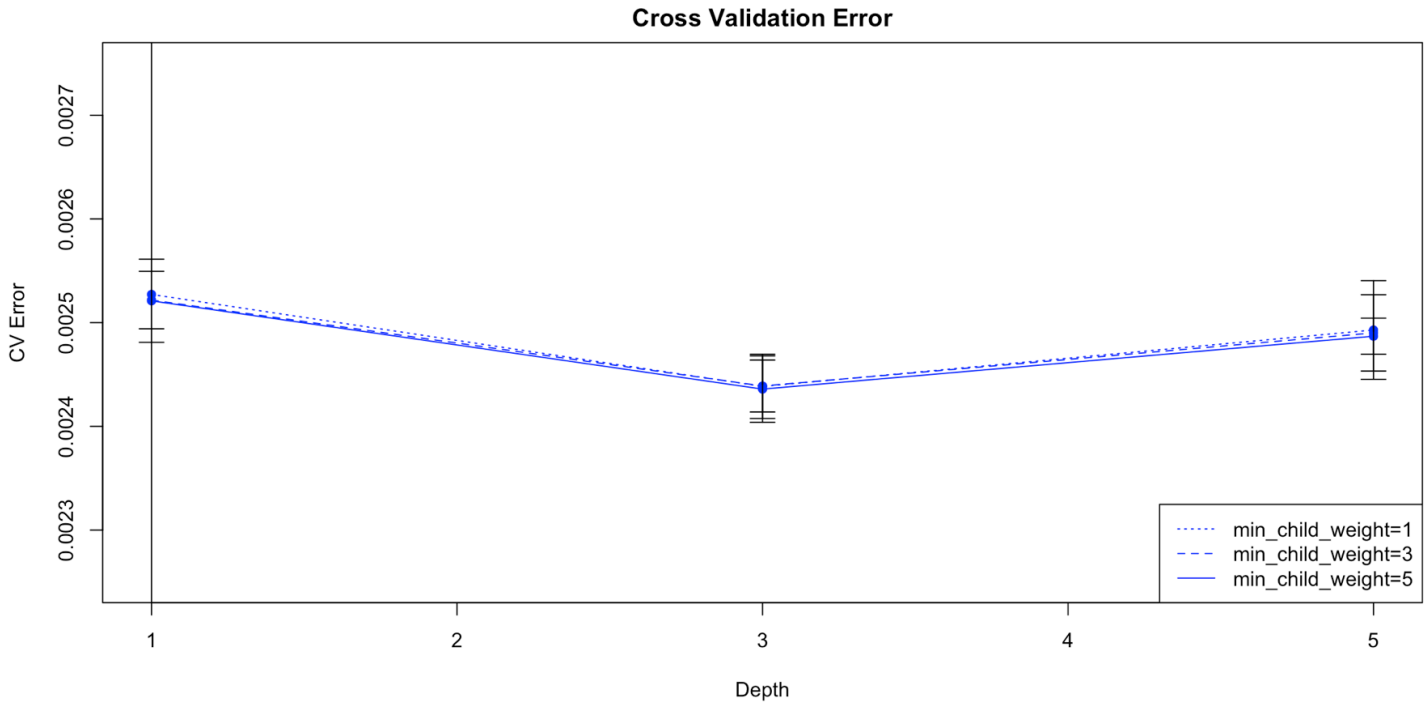
```
set.seed(2018)
source("../lib/cross_validation.R")
load(file="../output/keypoint_feature_train.RData")
feat_train <- keypoint_train$feature
label_train <- keypoint_train$label
if(run.cv){
  list_err_cv <- list()
  for(i in 1:3){
    cat("*** Class = ", (i-1), "***\n")
    ind_cv <- unlist(lapply(list_file[[i]], function(x) {((x-1)*1000+1) : (x*1000)}))
    feat_train_cv <- feat_train[ind_cv, ,]
    label_train_cv <- label_train[ind_cv, ,]

    tm_cv <- list()
    err_cv <- array(dim = c(length(depth_values)*length(child_weight_values), 2))
    for(k in 1:length(depth_values)){
      cat("max_depth = ", depth_values[k], "\n")
      for (j in 1:length(child_weight_values)) {
        cat("min_child_weight = ", child_weight_values[j], " ")
        tm_cv[[(k-1)*3+j]] <- system.time(
          err_cv[(k-1)*3+j,] <- cv.function(feat_train_cv, label_train_cv,
                                            par = list(depth = depth_values[k],
                                                       min.weight = child_weight_valu
es[j]),
                                            K))
        cat("time for current cv session = ", tm_cv[[(k-1)*3+j]][3], "\n")
      }
    }
    list_err_cv[[i]] <- err_cv
  }
  save(list_err_cv, file="../output/err_cv_xgboost.RData")
}
```

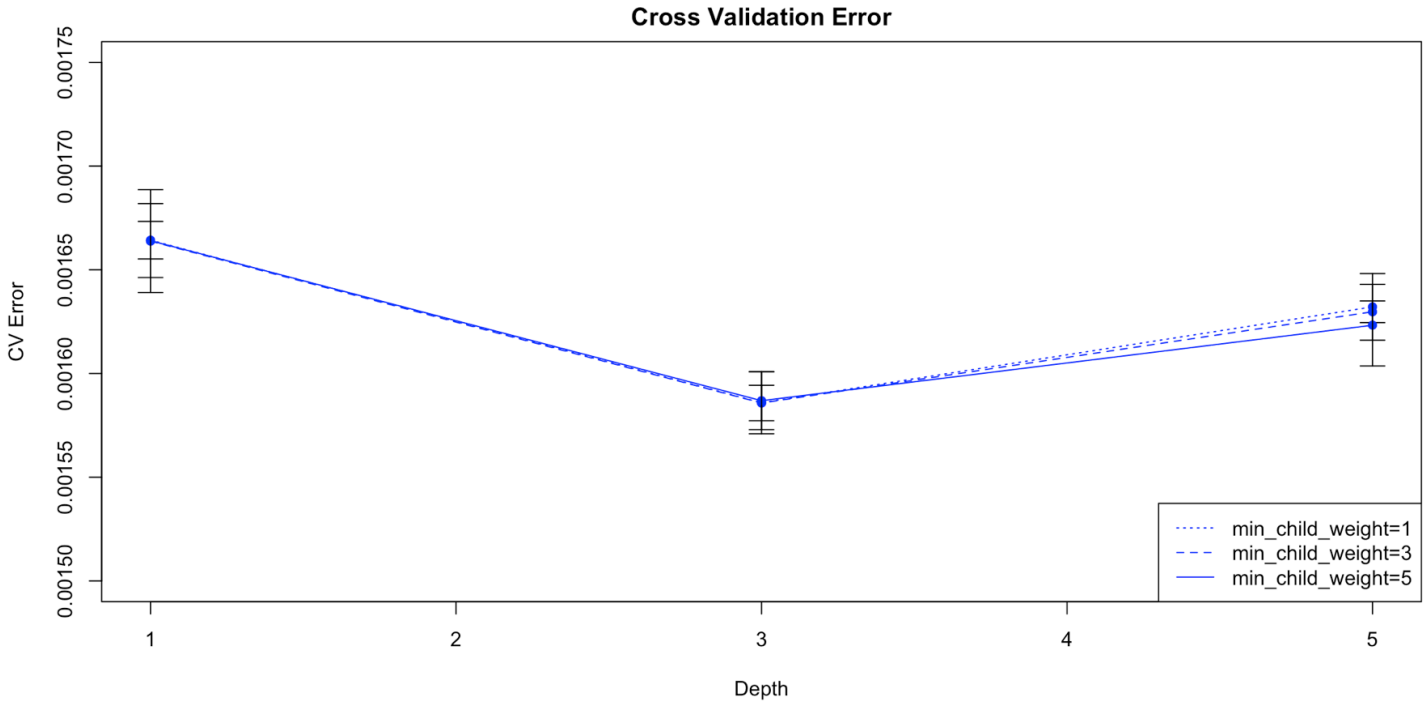
```
*** Class =  0 ***
max_depth =  1
min_child_weight =  1  time for current cv session =  761.086
min_child_weight =  3  time for current cv session =  740.745
min_child_weight =  5  time for current cv session =  721.389
max_depth =  3
min_child_weight =  1  time for current cv session =  2076.568
min_child_weight =  3  time for current cv session =  2030.559
min_child_weight =  5  time for current cv session =  1988.801
max_depth =  5
min_child_weight =  1  time for current cv session =  3788.385
min_child_weight =  3  time for current cv session =  3193.676
min_child_weight =  5  time for current cv session =  3511.487
*** Class =  1 ***
max_depth =  1
min_child_weight =  1  time for current cv session =  835.604
min_child_weight =  3  time for current cv session =  733.143
min_child_weight =  5  time for current cv session =  792.41
max_depth =  3
min_child_weight =  1  time for current cv session =  2155.828
min_child_weight =  3  time for current cv session =  2118.375
min_child_weight =  5  time for current cv session =  2238.64
max_depth =  5
min_child_weight =  1  time for current cv session =  3181.185
min_child_weight =  3  time for current cv session =  3296.55
min_child_weight =  5  time for current cv session =  3245.057
*** Class =  2 ***
max_depth =  1
min_child_weight =  1  time for current cv session =  782.721
min_child_weight =  3  time for current cv session =  780.841
min_child_weight =  5  time for current cv session =  784.608
max_depth =  3
min_child_weight =  1  time for current cv session =  2086.721
min_child_weight =  3  time for current cv session =  2206.748
min_child_weight =  5  time for current cv session =  2075.829
max_depth =  5
min_child_weight =  1  time for current cv session =  3447.046
min_child_weight =  3  time for current cv session =  3435.197
min_child_weight =  5  time for current cv session =  3420.122
```

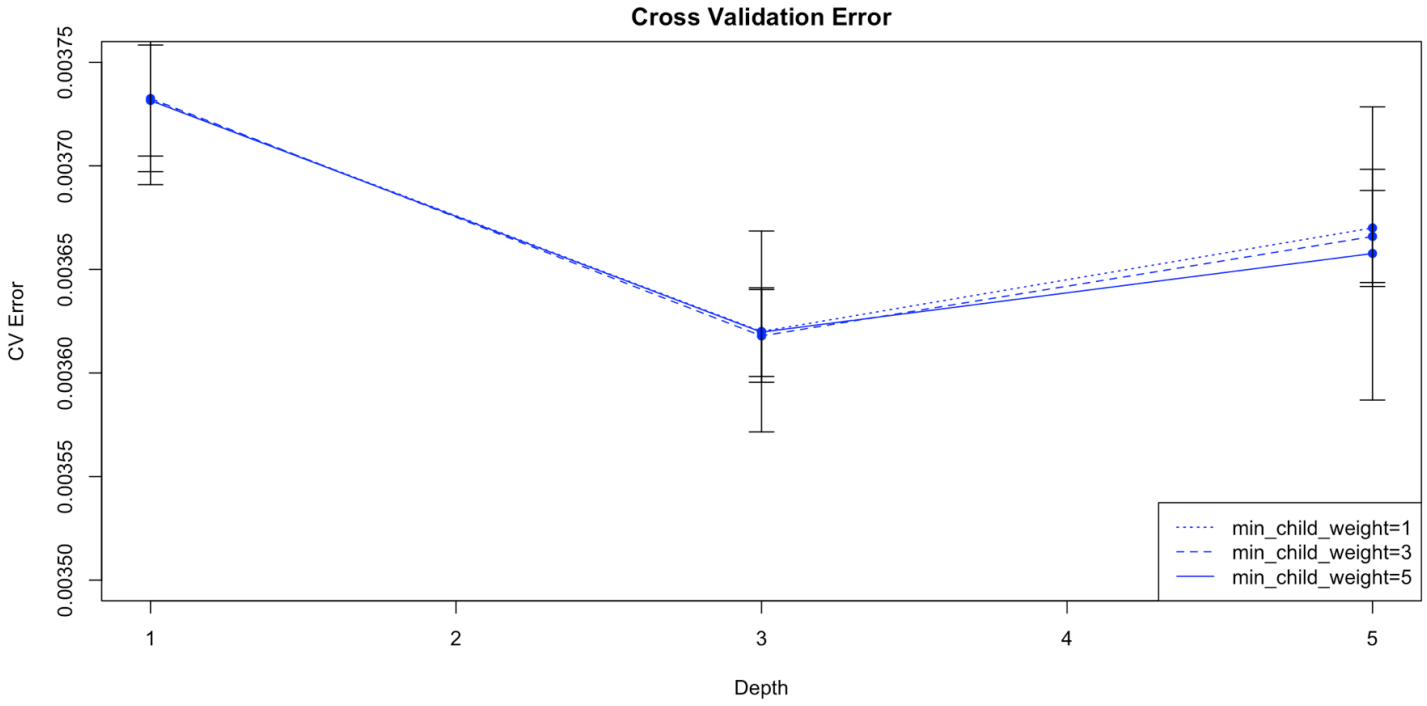
Visualize cross-validation results.

Hide

```
if(run.cv){
  load("../output/err_cv_xgboost.RData")
  for(j in list_err_cv){
    err_cv <- j
    plot(depth_values, err_cv[c(1,5,9),1], xlab="Depth", ylab="CV Error",
         main="Cross Validation Error", type="n", ylim=c(floor(min(err_cv[,1])*4000)/
4000,
                                                         ceiling(max(err_cv[,1])*4000
)/4000))
    for (i in c(2,1,0)) {
      points(depth_values, err_cv[((1:3)*3-i),1], col="blue", pch=16)
      lines(depth_values, err_cv[((1:3)*3-i),1], col="blue", lty=i+1)
      arrows(depth_values, err_cv[((1:3)*3-i),1]-err_cv[((1:3)*3-i),2], depth_values,
             err_cv[((1:3)*3-i),1]+err_cv[((1:3)*3-i),2], length=0.1, angle=90, code=
3)
    }
    legend("bottomright", legend=c("min_child_weight=1","min_child_weight=3","min_chi
ld_weight=5"),
           lty=c(3,2,1), col="blue")
  }
}
```







- Choose the “best” parameter value

Hide

```
list_par_best <- list()
for(i in 1:3){
  err_cv <- list_err_cv[[i]]
  best_ind <- which.min(err_cv[,1])
  best_weight_ind <- (best_ind-1)%%length(depth_values) + 1
  best_depth_ind <- (best_ind-best_weight_ind)%/%length(depth_values) + 1
  par_best <- list(depth=depth_values[best_depth_ind],
                   min.weight=child_weight_values[best_weight_ind])
  list_par_best[[i]] <- par_best
  print(par_best)
}
```

```
$depth
[1] 3

$min.weight
[1] 5

$depth
[1] 3

$min.weight
[1] 3

$depth
[1] 3

$min.weight
[1] 3
```

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

Hide

```
set.seed(2018)
list_tm_train <- list()
list_fit_train <- list()
for(i in 1:3){
  cat("*** Class = ", (i-1), "***\n")
  ind_train <- unlist(lapply(list_file[[i]], function(x) {((x-1)*1000+1) : (x*1000)})
)
  feat_train_i <- feat_train[ind_train, ,]
  label_train_i <- label_train[ind_train, ,]
  list_tm_train[[i]] <- system.time(list_fit_train[[i]] <-
                                      train(feat_train_i, label_train_i, list_par_bes
t[[i]]))
}
```

```
*** Class =  0 ***
*** Class =  1 ***
*** Class =  2 ***
```

Hide

```
save(list_fit_train, file="../output/fit_train_xgboost.RData")
save(list_tm_train, file="../output/tm_train_xgboost.RData")
```

Step 5: Super-resolution for test images

Feed the final training model with the completely holdout testing data.

- `superResolution_xg.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_xg.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="")
test_label_path <- paste(test_dir, "label.csv", sep="")
test_label <- read.csv(test_label_path, colClasses=c("NULL", NA, NA))
test_file_0 <- which(test_label$Label==0)
test_file_1 <- which(test_label$Label==1)
test_file_2 <- which(test_label$Label==2)
test_file <- list(file0=test_file_0, file1=test_file_1, file2=test_file_2)
list_tm_test <- list()
if(run.test){
  load(file="../output/fit_train_xgboost.RData")
  for(i in 1:3){
    cat("*** Class = ", (i-1), "***\n")
    list_tm_test[[i]] <- system.time(superResolution(test_LR_dir, test_HR_dir,
                                                     list_fit_train[[i]], test_file[[
i]][401:500]))
  }
}
```

```
*** Class =  0 ***
*** Class =  1 ***
*** Class =  2 ***
```

Hide

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

```
Time for constructing training features= 377.383 s
```

Hide

```
load("../output/tm_train_xgboost.RData")
cat("Time for training model=", sum(sapply(list_tm_train, function(x){ x[3] })), "s \
n")
```

```
Time for training model= 1578.32 s
```

Hide

```
load("../output/tm_test_xgboost.RData")
cat("Time for super-resolution=", sum(sapply(list_tm_test, function(x){ x[3] })), "s
\n")
```