

Project 3 - Example Main Script

Chengliang Tang, Tian Zheng

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

This file is currently a template for running evaluation experiments of image analysis (or any predictive modeling). You should update it according to your codes but following precisely the same structure.

```
if(!require("EBImage")){  
  source("https://bioconductor.org/biocLite.R")  
  biocLite("EBImage")  
}
```

```
## Loading required package: EBImage
```

```
if(!require("gbm")){  
  install.packages("gbm")  
}
```

```
## Loading required package: gbm
```

```
## Loaded gbm 2.1.5
```

```
library("EBImage")  
library("gbm")
```

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.

```
set.seed(2018)  
# 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.

```
train_dir <- "../data/train/" # 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

```
run.cv=FALSE # run cross-validation on the training set
K <- 5 # number of CV folds
run.feature.train=T # 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.

```
model_values <- seq(3, 11, 2)
model_labels = paste("GBM with depth =", model_values)
```

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.

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

Step2.5 split the training/test set according to lables

```
ts<-c()#index of test set
l<-unique(extra_label$Label)
label<-extra_label$Label
for(i in l){
  train_sub<-which(label==i)
  ts<-c(ts,sample(train_sub,length(train_sub)/5))
}
train_ind<-setdiff(1:1500,ts)#index of training set
```

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

```
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,index=train_ind))
  feat_train <- dat_train$feature
  label_train <- dat_train$label
  save(dat_train, file="../output/feature_train.RData")
}

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 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.

```
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.

```
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=", k, "\n")
    err_cv[k,] <- cv.function(feat_train, label_train, model_values[k], K)
  }
  save(err_cv, file="../output/err_cv.RData")
}
```

Visualize cross-validation results.

```
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(0.0034, 0.0036))
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)
```



- Choose the “best” parameter value

```
model_best=model_values[1]
if(run.cv){
  model_best <- model_values[which.min(err_cv[,1])]
}
#Using the one standard error rule, the best depth is 7 instead of 11
par_best <- list(depth=7)
```

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

```
tm_train=NA
tm_train <- system.time(fit_train <- train(feet_train, label_train, par_best))
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
```

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

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.

```
source("../lib/superResolution.R")
test_dir <- "../data/train/" # 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(performance<-superResolution(train_LR_dir, train_HR_dir,fit_train,index=ts))
}
```

Summarize the test MSE and PSNR

```
#test mse
performance[1]

## [1] 0.003390449

#test psnr
performance[2]

## [1] 24.69743
```

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.

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

## Time for constructing training features= 70.87 s
cat("Time for training model=", tm_train[1], "s \n")

## Time for training model= 5680.271 s
cat("Time for super-resolution=", tm_test[1], "s \n")

## Time for super-resolution= 1481.813 s
```