Baseline GBM

Group 11

```
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
```

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

```
## Loading required package: foreach
```

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

```
## Loading required package: parallel
```

```
library("EBImage")
library("gbm")
library("foreach")
library("parallel")
```

Step 0: specify directories.

```
# use relative path for reproducibility
set.seed(2019)
#setwd("~/Desktop/GR5243/Project3/Spring2019-Proj3-grp11/")
# here replace it with own file path
```

```
train_dir <- "C:/Users/xtxwq/Desktop/72 dragon/Spring2019-Proj3-spring2019-proj
3-grp11-master/data/train_set/"
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
- (T/F) run training set

```
run.cv=F # 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
run.train=TRUE # run training 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(1, 5, 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))
```

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("C:/Users/xtxwq/Desktop/72 dragon/Spring2019-Proj3-spring2019-proj3-grp1
1-master/lib/feature_new.R") #using feature_new

tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(train_LR_dir, train_HR_d
ir))
  feat_train <- dat_train$feature
  label_train <- dat_train$label
}

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

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("C:/Users/xtxwq/Desktop/72 dragon/Spring2019-Proj3-spring2019-proj3-grp1
1-master/lib/train.R")
test <- function(modelList, dat_test){

  ### Fit the classfication model with testing data

  ### Input:
  ###  - the fitted classification model list using training data
  ###  - processed features from testing images
  ### Output: training model specification

  ### load libraries
  library("gbm")

  predArr <- array(NA, c(dim(dat_test)[1], 4, 3))

  for (i in 1:12){
    fit_train <- modelList[[i]]
    ### calculate column and channel
    c1 <- (i-1) %% 4 + 1
    c2 <- (i-c1) %/% 4 + 1
    featMat <- dat_test[, , c2]
    ### make predictions
    predArr[, c1, c2] <- predict(fit_train$fit, newdata=featMat,
                    n.trees=fit_train$iter, type="response")
  }
  return(as.numeric(predArr))
}
```

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("C:/Users/xtxwq/Desktop/72 dragon/Spring2019-Proj3-spring2019-proj3-grp1
1-master/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="err_cv.RData")
}
```

Visualize cross-validation results.

```
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, 0.25))
  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])]
}

par_best <- NULL #list(depth=model_best)
```

- 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(feat_train, label_train, par_best))
```

```
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predi
ctive performance is reasonably competitive. Using cv_folds>1 when calling gbm
usually results in improved predictive performance.
```

```
save(fit_train, file="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("C:/Users/xtxwq/Desktop/72 dragon/Spring2019-Proj3-spring2019-proj3-grp1
1-master/lib/superResolution_new.R")
#using superresolution_new
test_dir <- "C:/Users/xtxwq/Desktop/72 dragon/Spring2019-Proj3-spring2019-proj3
-grp11-master/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="fit_train.RData")
  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.

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

```
## Time for constructing training features= 1.83 s
```

```
cat("Time for training model=", tm_train[1], "s \n")
```

```
## Time for training model= 5.15 s
```

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

```
## Time for super-resolution= 11.25 s
```