Project 3

Mar 1, 2018

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

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

```
packages.used=c("gbm", "xgboost")
# check packages that need to be installed.
packages.needed=setdiff(packages.used,
                        intersect(installed.packages()[,1],
                                  packages.used))
# install additional packages
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE,
                   repos='http://cran.us.r-project.org')
}
## Loading packages
library("EBImage")
library("gbm")
```

```
## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

```
library("xgboost")
```

Step 0: Set local working directory.

```
# setwd("/Users/JingyiWang/Google Drive/5243 Data/Porject 3/train")
```

###Set relative path for training and test dataset.

```
experiment_dir <- "../data/"
img_train_dir <- paste(experiment_dir, "train/images/", sep="")
img_test_dir <- paste(experiment_dir, "test/", sep="")
```

###Set seed and import functions

```
set.seed(1234)
source("../lib/train.R")
source("../lib/test.R")
source("../lib/feature.R")
source("../lib/cross_validation.R")
source("../lib/tune.R")
```

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=TRUE # run cross-validation on the training set
K <- 5  # number of CV folds
run.test=FALSE # run evaluation on an independent test set
run.tune = FALSE
```

####Boolean variables for Feature Extraction procedures

```
run.feature.train = FALSE
run.feature.test = FALSE
```

####Boolean variables indicating which model to run

```
run.gbm = TRUE   # if choose gbm, then we are running baseline model
run.xgboost = FALSE
run.svm = FALSE

run.merge = FALSE
run.baseline.retrain = FALSE
run.SIFT.test = FALSE
```

Using cross-validation or independent test set evaluation, we compare the performance of different classifiers or classifiers 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 maybe comparing very different classifiers. You can assign them numerical IDs and labels specific to your project.

```
shrinks_range <- c(0.01,0.05,0.1,0.15,0.2) #for gbm
trees_range  <- c(40,50,60,70,100) #for gbm
mfinal <- c(50, 75, 100, 125) # for adaboost
max_depth_values<-seq(3,9,2) #for xgboost
min_child_weight_values <- seq(1,6,2) #for xgboost
```

Step 2: import training images class labels.

```
label_train <- c(rep(0,1000),rep(1,1000),rep(2,1000))
```

Step 3: baseline model

Import SIFT feature

```
if(run.gbm == T){
  # import SIFT feature
  SIFT_feature <- read.csv('../data/train/SIFT_train.csv',header = F)[,-1]
  # import SIFT test feature
  if(run.SIFT.test == T){
    SIFT_feature_test <- read.csv('../data/train/SIFT_test.csv',header = F)[,-1]
  }
}
```

Tune parameter

```
# decide whether tune parameters
if(run.tune == T){
   par_best_baseline <- tune(dat_train ,label_train,
               run.xgboost = T,
               run.gbm = F,
               run.adaboost = F)
save(par_best_baseline,file = "par_best_baseline.RData")
print("Best parameter is:", par_best_baseline)
}

# load best parameter for baseline model
load("../output/par_best_baseline.RData")
```

Whether retrain baseline model

```
if(run.baseline.retrain == T){
  tm_train_baseline <- system.time(
    fit_train_baseline <- train(SIFT_feature, label_train, par_best_baseline)
    )
  save(fit_train_baseline, file="../output/fit_train_baseline.RData")
}
load("../output/fit_train_baseline.RData")
```

Whether run SIFT test

```
if(run.SIFT.test){
  test_baseline_pred <- test(SIFT_feature_test, dat_test,
                         run.xgboost = F, run.gbm = T,
                         run.adaboost = F, par=NULL)
  save(test_baseline_pred, file = "../output/test_baseline_pred.RData")
  write.csv(test_baseline_pred, file = "../output/test_baseline_pred.csv")
}
```

Step 4: construct visual feature

For this simple example, we use the row averages of raw pixel values as the visual features. Note that this strategy **only** works for images with the same number of rows. For some other image datasets, the feature function should be able to handle heterogeneous input images. Save the constructed features to the output subfolder. `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 that are required by all the models you are going to evaluate later.

```
if(run.feature.train){
  rgb_feature <- feature(img_dir = img_train_dir, par = NULL)
  save(rgb_feature, file="../output/feature_train.RData")
}
if(run.feature.test){
  rgb_test <- feature(img_dir = img_test_dir, par = NULL)
  save(rgb_test, file="../output/feature_test.RData")
}
```

Step 5: 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: + dat_train - processed features from images + label_train - class labels for training images + run.xxxxxx - select which model to fit + par - specified parameters, otherwise use default + Output: + fitted model models/settings/links to external trained configurations. + `test.R` + Input: + dat_test - processed features from testing images + run.xxxxxx - select which model to fit + par - specified parameters, otherwise use default + Output: + model prediction

```
load("../output/feature_train.RData")
if(run.test == T) load("../output/feature_test.RData")
```

decide whether to merge feature

```
if(run.merge == T){
  gist_train <- read.csv('./data/gist_feature/gistfea512.csv',header = F)
  dat_train = cbind(rgb_feature, gist_train)
  if(run.test == T){
    gist_test <- read.csv('../data/gist_feature.csv',header = F)
    dat_test = cbind(rgb_test, gist_test)
  }
}else{
  dat_train = rgb_feature
  if(run.test == T) dat_test = rgb_test
}
```

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.

```
if(run.cv){
err_cv <- cv(dat_train ,label_train,
             run.xgboost = T,
             run.gbm = F,
             run.adaboost = F,
             K = 5,
             par = NULL)$error

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

Visualize cross-validation results.

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

- Choose the “best”" parameter value

```
if(run.tune){

  par_best <- tune(dat_train ,label_train,
                 run.xgboost = T,
                 run.gbm = F,
                 run.adaboost = F)
  save(par_best,file = "../output/par_best.RData")
  print("Best parameter is:", par_best)
  }
```

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

```
tm_train=NA
load("../output/par_best.RData")
tm_train <- system.time(fit_train <- train(dat_train, label_train, par_best))
```

```
## Warning in if (run.xgboost == T) {: the condition has length > 1 and only
## the first element will be used
```

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

Step 6: Make prediction

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

```
tm_test=NA
if(run.test){
  test_pred <- test(fit_train, dat_test,
                    run.xgboost = T, run.gbm = F,
                    run.adaboost = F, par=NULL)
  save(test_pred,file = "..output/test_pred.RData")
  write.csv(test_pred,file = "../output/test_pred.csv")
  }
```

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_train[1], "s \n")
```

```
## Time for constructing training features= 67.14 s
```

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

```
## Time for constructing testing features= NA s
```

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

```
## Time for training model= 67.14 s
```

```
cat("Time for making prediction=", tm_test[1], "s \n")
```

```
## Time for making prediction= NA s
```