Main

March 19, 2018

```
knitr::opts_chunk$set(echo = TRUE)
```

In this project, we carried out model evaluation and selection for predictive analytics on the image data. We created a classification model for images of poodles, fried chickens, and muffins. For feature extraction method, we tried RGB, SIFT, SURF, HoG and ORB. For classification methods, we tried GBM, Xgboost, SVM with linear and RBF kernels and Random Forest. Moreover, we also tried to combine some dimension reduction methods (PCA, tsne, LLE and ISOMAP) with feature extraction methods to decrease running time. In the end, we chose RGB and Xgboost as our final (advanced) model.

```
packages.used=c("EBImage","xgboost","plyr","gbm")
packages.needed=setdiff(packages.used, intersect(installed.packages()[,1],
                                                 packages.used))
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}

library(gbm)
library(EBImage)
library(xgboost)
library(plyr)
```

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

```
img_train_dir <- '..//data/train/'  #this is for train features
img_test_dir <- '..//data/test/'  #this is for test set images
```

Note: While doing the project, we use random sample to split the data into training and test sets rather than put them into different files. To ensure the results are reproducible, set.seed() is used in this project.

Step 1: set up controls for evaluation experiments.

In this chunk, ,we have a set of controls for the evaluation experiments.

- (T/F) run advance model(T=advance model; F=baseline model)
- (T/F) cross-validation on the training set
- (number) k, the number of CV folds
- (number) seed, the seed used to ensure results to be reproducible
- (number) train_p, the proportion of training data, preferable to be between 0.6-0.8
- (T/F) process RGB features for training set(T=RGB; F=SIFT)
- (T/F) run evaluation on an independent test set
- (T/F) process RGB features for test set(T=RGB; F=SIFT)

```
run.adv=TRUE #run advance model
run.cv=TRUE # run cross-validation on the training set
```

```
k <- 5  # number of CV folds
seed <- 123  #random seed
train_p <- 0.8 #the proportion of training data
run.feature.train=TRUE # process RGB features for training set
run.test=FALSE # run evaluation on an independent test set
run.feature.test=FALSE # process RGB features for test set
```

Step 2: import training images class labels.

```
if (run.adv){
  label<-read.csv(paste(img_train_dir,'label_train.csv',sep=''),header=T,sep=',')
  y<-label$label-1  #xgboost label from 0 to num.class
} else{
  label<-read.csv(paste(img_train_dir,'label_train.csv',sep=''),header=T,sep=',')
  y<-as.factor(label$label)
}
```

Step 3: Preparation for Training Model

```
source("../lib/feature.R")
```

Step 3.1: Extract features

```
if(run.feature.train){
  train.data <- read.csv(paste('..//output/',"rgb_feature.csv",sep = ""),
                 header=T, sep=',')[,-1]
}else{
  train.data<- read.csv(paste(img_train_dir,"SIFT_train.csv",sep = ""),
                  header=F, sep=',')[,-1]
}


if(run.test){
  if(run.feature.test){
  tm_feature_test <- system.time(
    new.data <-RGB_feature(paste(img_test_dir,"images/",sep=""),export=T))
} else {
  new.data<-read.csv(paste(img_test_dir,"SIFT_test.csv",sep = ""),
                     header=T, sep='')[,-1]
  }
}
```

Step 3.2: Random split the data to training and testing set

```
train<-cbind(y,train.data)
set.seed(seed)
samp<-sample(1:nrow(train),nrow(train)*train_p)
train.data<-train[samp,]
test.data<-train[-samp,]
```

Step 4: Train a classification model with training images

Call the train model and test model from library.

```
source("../lib/train.R")
source("../lib/test.R")
```

Model selection with cross-validation

- Get the best parameters and train the model with the entire training set.

```
if (!run.adv){
  cv.error<-tuned$cv.error
  best.paras<-data.frame(ntree=tuned$ntree,
                         shrinkage=tuned$shrinkage,
                         interaction.depth=tuned$interaction.depth)
  tm_train<-system.time(
    opt<-gbm_model(train.data,tuned$ntree,tuned$shrinkage,
                   tuned$interaction.depth))
  test.error<-gbm_test(opt,test.data)
}else{
  cv.error<-tuned$cv.error
  best.paras<-data.frame(eta=tuned$eta,
                         max_depth=tuned$max_depth,
                         min_child_weight=tuned$min_child_weight)
  tm_train<-system.time(opt<-xgboost_model(train.data,best.paras))
  test.error<-xgboost_test(opt,test.data)
}
```

```
## [1]  train-merror:0.184167
## [2]  train-merror:0.152917
## [3]  train-merror:0.147500
## [4]  train-merror:0.127083
## [5]  train-merror:0.117500
```

Step 5: Evaluation Model

Step 5.1: Get the CV error and test error based on the given 3000 data

```
cv.error;test.error
```

```
## [1] 0.1675
```

```
## [1] 0.1783333
```

Step 5.2: Make prediction for new testing set

```
if (run.test){
  if (!run.adv){
  best.iter=gbm.perf(opt,method="OOB", plot.it = FALSE)
  tm_test<-system.time(pred<-predict(opt,new.data,best.iter,type='response'))
  pred.new=apply(pred,1,which.max)
  }else{
  tm_test<-system.time(pred.new<-predict(opt,as.matrix(new.data))+1)
  }
```

```
  save(pred.new, file="..//output/pred_new.RData")
}
```

Summary

Running Time

```
if(run.test){
  cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
  cat("Time for training model=", tm_train[1], "s \n")
  cat("Time for making prediction=", tm_test[1], "s \n")
}else{
  cat("Time for training model=", tm_train[1], "s \n")
}
```

```
## Time for training model= 0.96 s
```

Error Rate

```
cat("CV error rate for given images=",cv.error,'\n')
```

```
## CV error rate for given images= 0.1675
```

```
cat("Test error rate for given images= ",test.error)
```

```
## Test error rate for given images=  0.1783333
```