Project 3 - Group9 Main Script

Liangbin Chen, Yaqin Li, Tongyue Liu, Imroze Shaheen, Zheren Tang

March 24, 2017

Step 0: Environment Preparation.

Check out for the environment of the classification engine, we have `train.R`,`test.R`, `cross_validation.R`, `feature.R` contained the function we need in each step. Also, set all packages needed before the engine start to classify. In our classification, we also try some `python` and `Matlab` program in our feature extraction.

```
#setwd("~/GitHub/spr2017-proj3-group-9")
if(!require("gbm")){
  install.packages("gbm")
}
library(ggplot2)
library(EBImage)
library(gbm)
library(adabag)
library(data.table)
library(dplyr)
library(randomForest)
library(parallel)
library(OpenImageR)
library(e1071)
source("../lib/cross_validation.R")
source("../lib/train.R")
source("../lib/test.R")
source("../lib/feature.R")
```

Step 1: Construct Baseline

In our project we use `SIFT feature` and `GBM model` as our baseline model. Since the baseline model runs really slow, so I simply include our model in the `Basicmodel.RData`. In this final report, I just Use the result of the two models comparison.

```
#baseline
# k<-3
# cl <- makeCluster(getOption("cl.cores", 8))
# err<-parLapply(cl, 1:16, cv.function,X.train=sift, y.train=y, K=k)
# plot(err)
# d<-which.min(err)
# gbmbase <- gbm.fit(x=sift, y=y,
#                        n.trees=1000,
#                        distribution="bernoulli",
#                        interaction.depth=d,
#                        bag.fraction = 0.5,
#                        verbose=FALSE)
# save(gbmbase,file="Basicmodel.RData")
```

The error rate for our baseline model is about 27.32%. Based on the raw SIFT features, we have tried Xgboost, Random Forest, GBM, KNN, SVM models. After checking the error rate we find that the most

effective model is the Random Forest.

Step 2: Construct Visual Feature

For the visual Feature part, we have tried CNN, PCA, Logistic, Transform Distance, HOG, and some personalized features of the picture. After trained model and check the error rate on our selected model, we find that the best combination of features are made by HOG+Personalized+Improved SIFT features. We got 56 features(=54 HOG features+1 personalized feature+1 sgn(SIFT) feature)in our final classification engine.

```
t1<-Sys.time()
fea<-feature("../data/training_data/training_data/raw_images","image")
t2<-Sys.time()
feature.time<-t2-t1

#test data
# t1<-Sys.time()
# fea.test<-feature("../data/training_data/training_data/test_images","image")
# t2<-Sys.time()
# feature.time.test<-t2-t1
```

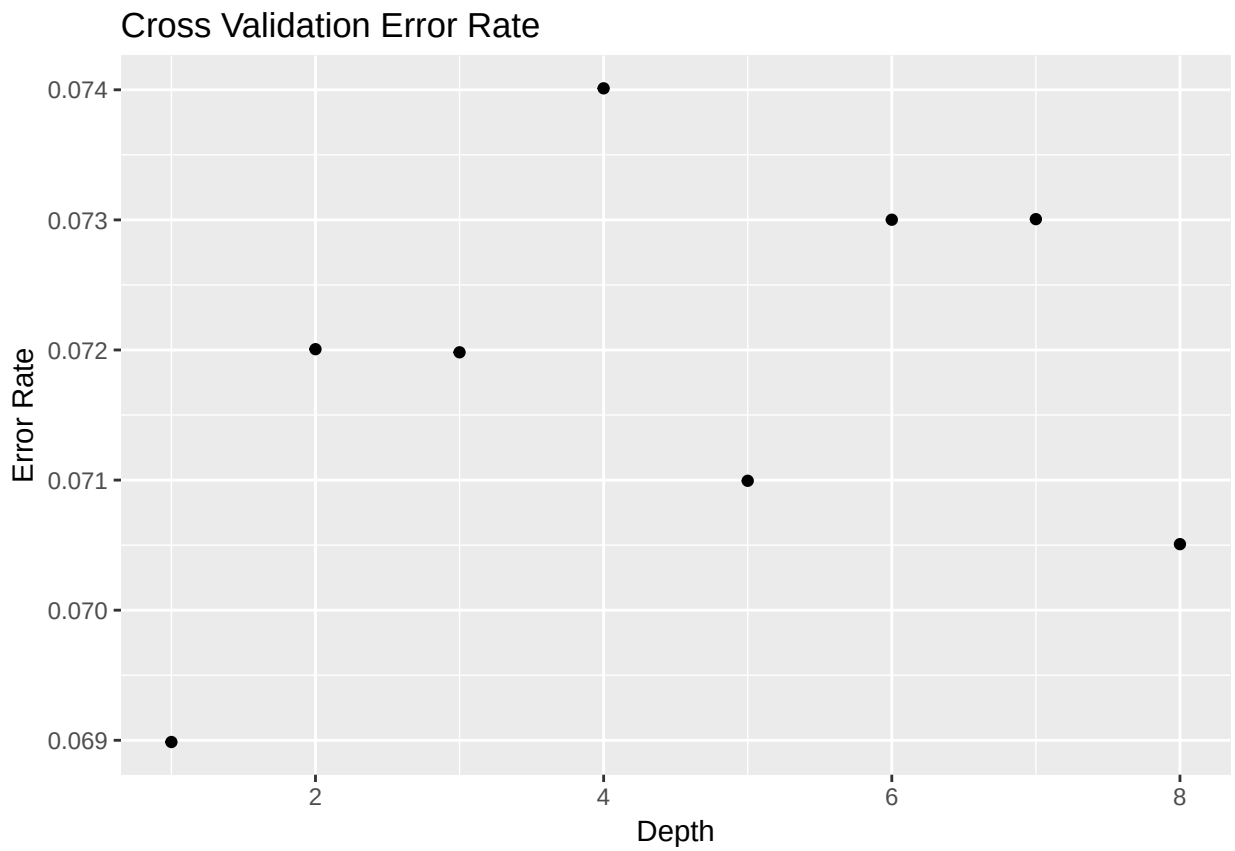
Explaination of our personalized features.

dog1 dog2 chicken1 chicken2 We can easily find that most of the dogs pictures contain the dog in the middle of the whole picture, and if we clean the pixel less than 4 pixels of white, we can discovered that dog pictures almost have a larger area of the black near the outline of the image.

Step 3: Model selection with cross-validation

Using cross-validation or independent test set evaluation, we compare the performance of different classifiers or classifiers with different specifications. We use Random Forest with different `depth` from 1 to 8. In the following chunk, you can see our error rate through the plot.

```
#cross validation CV
cl <- makeCluster(getOption("cl.cores", 8))
y<-c(rep(0,1000),rep(1,1000))
k<-3
rd_err <- parLapply(cl, 1:8,  rf.cv.function,X.train=fea, y.train=y,ntree=1000, K=k)
rd_err<-unlist(rd_err)
depth<-seq(1,8,1)
class<-rep(c("rd_err","Random Forest"),c(8,8))
error1<-data.frame(depth,rd_err)
ggplot(data = error1) +
  geom_point(mapping = aes(x = depth, y = rd_err))+
  labs(title = "Cross Validation Error Rate", x = "Depth", y = "Error Rate")
```




```
# rd_err.test <- parLapply(cl, 1:8,  rf.cv.function,X.train=fea.test, y.train=y,ntree=1000, K=k)
# rd_err<-unlist(rd_err.test)
# err<-c(rd_err,rd_err.test)
# error.combine<-data.frame(depth,rd_err,class)
#ggplot(data = error.combine) +
  #geom_point(mapping = aes(x = depth, y = err,fill=class))
  #labs(title = "Cross Validation Error Rate", x = "Depth", y = "Error Rate")
```

Step 4: Summarize Running Time

Following chunk contains the code of running time. Because of the

```
t1<-Sys.time()
rf_fit <- randomForest(x=fea,y=as.factor(y),
                       importance=TRUE,
                       ntree=1000,
                       nodesize=3)
save(rf_fit,file="Advancemodel.RData")
t2<-Sys.time()
train.time<-t2-t1
```

Step5: Make prediction

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

```
# t1<-Sys.time()
# pred<-rf_test(rf_fit,fea.test)
# test.err<-1-mean(pred==y)
# t2<-Sys.time()
# pred.time<-t2-t1
```

Step6: Summarize Running Time

Prediction performance matters, do 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=", feature.time, "s \n")
```

```
## Time for constructing training features= 1.021262 s
```

```
#cat("Time for constructing testing features=", feature.test.time, "s \n")
cat("Time for training model=", train.time, "s \n")
```

```
## Time for training model= 14.07559 s
```

```
#cat("Time for making prediction=", pre.time, "s \n")
```

Attention! CNN deep learning part

CNN About this project, we have tried some new methods through Python. One of our teammates, Imroze is really good at deep learning part. So we built an CNN model together and find that the prediction result is really good. But since as the professor mentioned that this project does not focus on the deep learning part, and we also find some difficulties to get the output result in less than half an hour, so we didn't combine that part in our R models. It turns a really good result based on the trainning image.