# 5243_proj3_grp6_demo

*Dejian Wang, dw2726*

*3/23/2017*

In project 3, we are asked to construct an image classifier to distinguish grayscale images of poodle from fried chickens. The holding storage and memory cost, the test running time cost, together with the accuracy rate, are of great concern.

**Set up controls for evaluation experiments.**

**Import training images class labels.**

**General idea of conducting the experiment**

We hope to discover the best "model + feature" combination with carefully consideration of accuracy and computational cost trade-off. Specifically, we tried each combination among each model in a list of "candidate" models and a list of "candidate" features, recorded their error rate and computational time, then select the best combination. Besides, we also perform Principle Component Anlysis (PCA) methods onto features, train models, hoping to imporve performance.

**Implement the baseline.**

First, we implemented the baseline classifier: a GBM with Scale-invariant feature transform (SIFT) feature. We adopted the 'xgboost' package, an efficient implementation of the gradient boosting framework.

**load the SIFT feature** We are given .csv file of SIFT features. Read them and save them in .Rdata form.

```
load('../output/feature_sift.Rdata')
# feature_sift = read.csv('../data/train/features/feature_sift.csv', header = T)
# feature_sift = t(feature_sift)
# save(feature_sift, file="../output/feature_sift.RData")
```

**Parameter tuning with cross-validation.** Do model selection by choosing among different values of training model parameters. The paramaters we have choosen to tune are: `colsample_bytree`, `max_depth`, `nrounds`, `sub_sample` (subsample ratio of the training instance), and `eta` (learning rate) for training a GBM model, using xgb.train(). Here we display the procedure of tuning `max_depth`, which is the maximum depth of a tree, as a demonstration of our methods for tuning parameters.

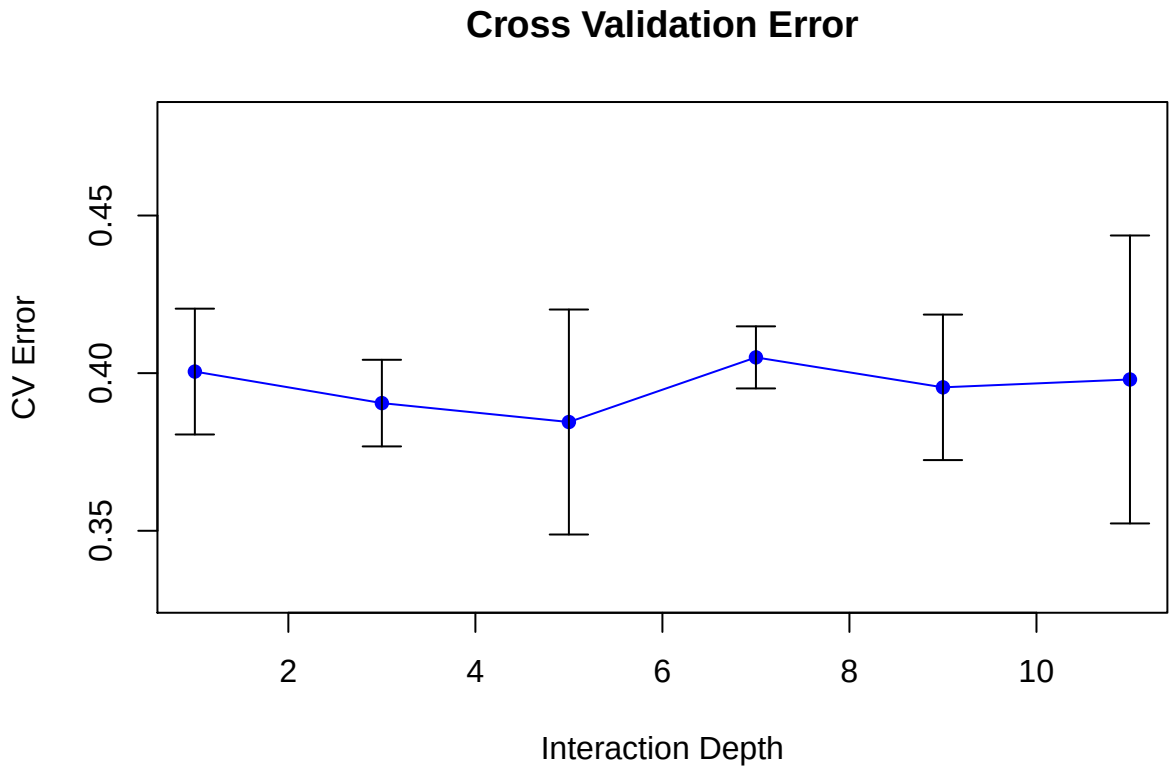
```
source("../lib/cross_validation.R")
source("../lib/train.R")
source("../lib/test.R")

model_values <- seq(1, 11, 2)
model_labels = paste("GBM with depth =", model_values)

# 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(data_sift, label_train, model_values[k], K, 'sift')
#   }
#   save(err_cv, file="../output/err_cv.RData")
# }
```

Visualize cross-validation error rate for different chosen value of parameter.




Choose the "best" parameter value that gives the smallest error rate.

```
model_best=model_values[1]
if(run.cv){
  cat('best value for depth: ', which.min(err_cv[,1]))
  model_best <- model_values[which.min(err_cv[,1])]
}
```

```
## best value for depth:  3
```

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

**Train the baseline model with best parameters** Finally we got the best baseline model with parameters carefully tuned.

```
tm_train=NA
tm_train <- system.time(fit_baseline <- train(feature_sift, label_train, par_best, 'sift'))
save(fit_baseline, file="../output/fit_baseline.RData")
```

### Select different features & models

Next, we select lists of candidate models and candidate features. For models, besides GBM, we will try additional two classic supervised learning algorithms, logistic regression and random forest. As for features (or descriptors), in addition to SIFT, we will also try local binary pattern (LBP), which is Gray-Scale Invariant and efficient, and Histogram of Oriented Gradient (HOG) as our feature.

With default setting, the number of dimemsions of candidate features are 5000, 13275, and 7056. Noticing that the dimemsions are large, we also perform principle component analysis (PCA) methods onto all features to reduce dimensions of features, hoping to speed up the computational process.

### Construct visual features

Construct LBP and HOG features by calling the Matlab functions (separately run hogfeature.m and lbpfeature.m with MATLAB in lib folder). Then load constructed features.

```
load('../output/feature_lbp.RData')
# feature_lbp = read.csv('../data/train/features/feature_lbp.csv', header = F)
# save(feature_lbp, file="../output/feature_lbp.RData")
load('../output/feature_hog.RData')
# feature_hog = read.csv('../data/train/features/feature_hog.csv', header = F)
# save(feature_hog, file="../output/feature_hog.RData")
```

After parameter tuning for each combination of feature and model, use the best parameters to train models. Build up and display the table below indicating the cross-validated error rate for each combination. We find that the one with the lowest error rate is "LBP + GBM (xgboost)", with a error rate of 7%.

```
err_table <- matrix(NA, nrow = 6, ncol = 3)
colnames(err_table) = c('GBM', 'Logistic Regression', 'Random Forest')
rownames(err_table) = c('SIFT', 'LBP', 'HOG', 'PCA + SIFT', 'PCA + LBP', 'PCA + HOG')
```

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

```
tm_train=NA
tm_train <- system.time(fit_advanced <- train(feature_lbp, label_train, par_best, 'lbp'))
save(fit_advanced, file="../output/fit_advanced.RData")
```

### 5. Make prediction

Feed the final training model with the completely holdout testing data. Construct lbp feature with lbpeature.m file (Modification to run: change the first line from: cd 'spr2017-proj3-group6/data/train', into cd 'spr2017-proj3-group6/data/test', to create the features for the test set) in Matlab first, then make prediction based on thoes features.

```
# tm_test=NA
# if(run.test){
#   # feature_lbp_test <- read.csv('../data/test/features/feature_lbp.csv')
#   # save(feature_lbp_test, file = "../output/feature_lbp_test.Rdata")
#   load(file=paste0("../output/feature_lbp_test.Rdata"))
#
#   load(file="../output/fit_advanced.RData")
#   tm_test <- system.time(pred_test <- test(fit_advanced, as.matrix(feature_lbp_test)))
```

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

### 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=", tm_feature_train[1], "s \n")
# # 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")
```