Project 3 - Labradoodle v.s. Fried Chicken

Group 8

March 24, 2017

Team Members

- Ken Chew
- Yue Jin
- Yifei Lin
- Sean Reddy
- Yini Zhang

Project Summary

- In this project, we implemented the Gradient Boosting Machine (GBM), Random Forest, Neural Network and Convol to generate a classification engine for grayscale images of poodles versus images of fried chickens.
- To further improve the prediction performance, besides the provided SIFT descriptors, we also used Histogram of Oriented Gradients descriptors to train the model.


```
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
```

```
## Warning: package 'gbm' was built under R version 3.3.3
```

```
## Loading required package: survival
```

```
## Warning: package 'survival' was built under R version 3.3.3
```

```
## Loading required package: lattice
```

```
## Loading required package: splines
```

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

```
## Loaded gbm 2.1.1
```

```
library("EBImage")
library("gbm")
```

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.

```
setwd("./ads_spr2017_proj3")
# here replace it with your own path or manually set it in RStudio to where this rmd file is located.
```

Provide directories for raw images. Training set and test set should be in different subfolders.

```
experiment_dir <- "../data/zipcode/" # This will be modified for different data sets.
img_train_dir <- paste(experiment_dir, "train/", sep="")
img_test_dir <- paste(experiment_dir, "test/", 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) run evaluation on an independent test set

```
run.cv=TRUE # 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
```

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.

```
model_values <- seq(3, 11, 2)
model_labels = paste("GBM with depth =", model_values)
```

Step 2: import training images class labels.

For the example of zip code digits, we code digit 9 as "1" and digit 7 as "0" for binary classification.

```
label_train <- read.table(paste(experiment_dir, "train_label.txt", sep=""),
                          header=F)
label_train <- as.numeric(unlist(label_train) == "9")
```

Step 3: 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.

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

tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(img_train_dir,
                                                       "train",
                                                       data_name="zip",
                                                       export=TRUE))
}

tm_feature_test <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature(img_test_dir,
                                                     "test",
                                                     data_name="zip",
                                                     export=TRUE))
}

#save(dat_train, file="./output/feature_train.RData")
#save(dat_test, file="./output/feature_test.RData")
```

Step 4: 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: a path that points to the training set features. + Input: an R object of training sample labels. + 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 classifiers. + Output: an R object of class label predictions on the test set. If there are multiple classifiers under evaluation, there should be multiple sets of label predictions.

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

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.

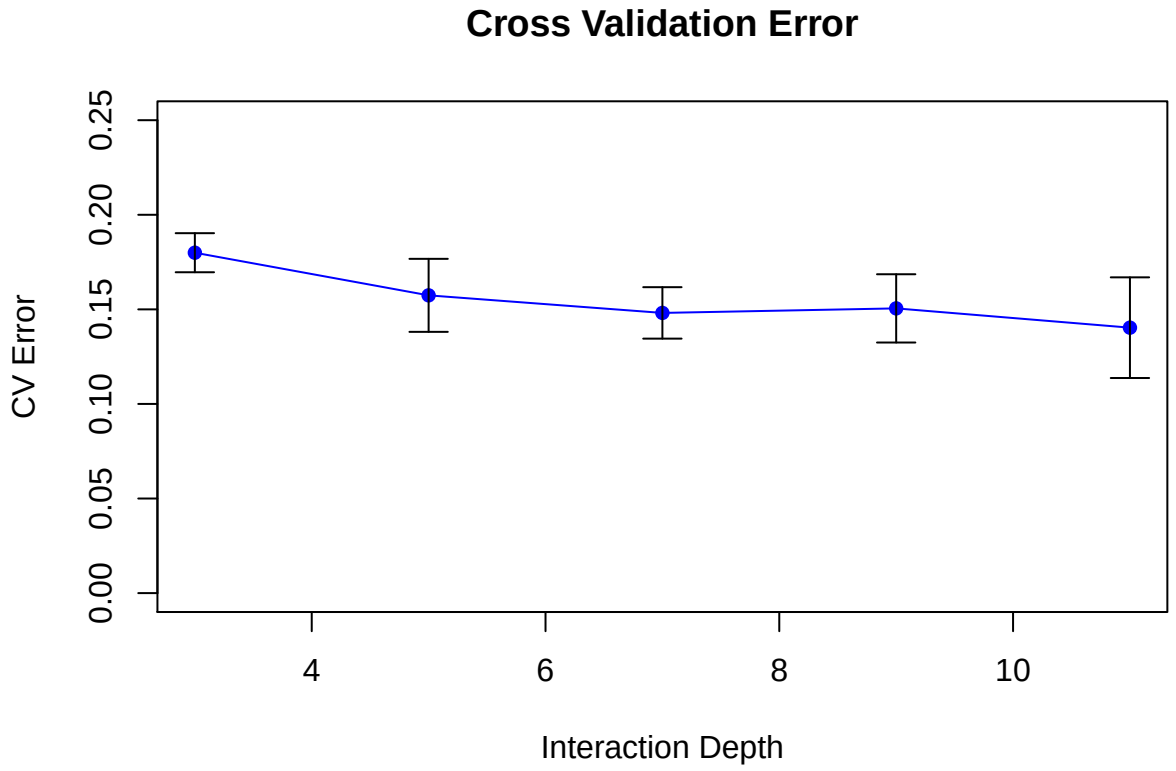
```
source("../lib/cross_validation.R")

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

```
## k= 1
## k= 2
## k= 3
## k= 4
## k= 5
```

Visualize cross-validation results.

```
if(run.cv){
  load("../output/err_cv.RData")
  #pdf("../fig/cv_results.pdf", width=7, height=5)
  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)
  #dev.off()
}
```



- Choose the "best"" parameter value

```
model_best=model_values[1]
if(run.cv){
  model_best <- model_values[which.min(err_cv[,1])]
}

par_best <- 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(dat_train, label_train, par_best))
```

```
## Warning in gbm.perf(fit_gbm, method = "OOB", plot.it = FALSE): OOB
## generally underestimates the optimal number of iterations although
## predictive performance is reasonably competitive. Using cv.folds>0 when
## calling gbm usually results in improved predictive performance.
```

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

Step 5: Make prediction

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

```
tm_test=NA
if(run.test){
  load(file=paste0("../output/feature_", "zip", "_", "test", ".RData"))
  load(file="../output/fit_train.RData")
  tm_test <- system.time(pred_test <- test(fit_train, dat_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")
```

```
## Time for constructing training features= 1.36 s
```

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

```
## Time for constructing testing features= 0.31 s
```

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

```
## Time for training model= 6.73 s
```

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

```
## Time for making prediction= 0.04 s
```