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.

```
## Init Rcpp
```

Step 0: specify directories.

Step 1: set up controls for evaluation experiments.

In this chunk, ,we have a set of controls for the evaluation experiments for all four models

- (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`, Random Forest with different PCA_Threshold + Number of trees, Neural Network with different Number of hidden layer, CNN with different Number of rounds () (numver of iteration)+ batch size(50)nunber of obs use each round + learn rate (0.01) (smaller - longer time to learn)

Step 2: construct visual feature

- In addition to 5000 sift features, we used Histogram of Oriented Gradients (HOG) method to generate 448 extra new features

- HOG + a feature descriptor used in computer vision and image processing for the purpose of object detection + it counts occurrences of gradient orientation in localized portions of an image + it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy

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

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

Call the train model and test model from library.

`train.R` and `test.R` are wrappers for all of our 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")
```

Step 3: Baseline GBM

```
sift_hog<-fread("../output/hog_feature+sift.csv")
sift<-fread("../output/sift_features/sift_features.csv",header=TRUE)
sift<-unlist(t(sift))


label <- read.table("../data/labels.csv",header=T)
label <- unlist(label)
label_train<-label
dat_train<-sift
hog_train<-sift_hog

# Train the model and tune parameters
library("gbm")
depth_values<-c(1,3,5,7,9)
err_cv<-matrix(nrow=length(depth_values), ncol=2)

K=5
for(k in 1:length(depth_values)){
  err_cv[k,] <- gbm_cv(dat_train, label_train, K=K,depth=depth_values[k])
}

write.csv(err_cv, file="../output/err_cv_gbm.csv")
```

```r
plot(depth_values, err_cv[,1], xlab="Interaction Depth", ylab="CV Error",
     main="Cross Validation Error",type="l",ylim=c(0,0.4))

depth_best1 <- depth_values[which.min(err_cv[,1])]

# best depth is 5
fit_train_gbm<-gbm_train(dat_train, label_train,depth=depth_best1)
pred_test1<-as.numeric(gbm_test(fit_train_gbm,dat_train)>0.5)


########################################################
# Train the model and tune parameters with hog features
err_cv_hog<-matrix(nrow=length(depth_values), ncol=2)

K=5
for(k in 1:length(depth_values)){
  err_cv_hog[k,] <- gbm_cv(hog_train, label_train, K=K,depth=depth_values[k])
}



write.csv(err_cv_hog , file="../../output/err_cv_gbm_hog.csv")

plot(depth_values, err_cv[,1], xlab="Interaction Depth", ylab="CV Error",
     main="Cross Validation Error",type="l",ylim=c(0,0.4))

depth_best2 <- depth_values[which.min(err_cv[,1])]
# best depth = 9

# Use the optimal model to fit the whole training data set and test the model


fit_train_gbm_hog<-gbm_train(hog_train, label_train,depth=depth_best2)
pred_test2<-as.numeric(gbm_test(fit_train_gbm_hog,hog_train)>0.5)

# Error rate
mean(pred_test1!=label_train)
mean(pred_test2!=label_train)
# 0.2355 & 0.098
```

Step 4: Random Forest

```r
# Load functions
source("../lib/Random_Forest_PCA/random forest_train_test_cv.R")


# Load features and label
feature <- fread("../output/hog_feature+sift.csv", header = TRUE)
label <- fread("../data/labels.csv")
label <- c(t(label))
feature <- tbl_df(feature)
```

```
######### Tuning parameters #########

# Tune parameter for random forest: ntree
ntree <- seq(10, 400, by=90)


err_cv_rf <- c()
err_sd_rf <- c()


for (j in 1:length(ntree)){
  cat("j=", j, "\n")
  result <- rf_cv(dat_train = feature, label_train = label, K = 5, ntree = ntree[j])
  err_cv_rf[j] <- result[1]
  err_sd_rf[j] <- result[2]
}


# Save results
save(err_cv_rf, file="../output/err_cv_rf.RData")
save(err_sd_rf, file="../output/err_sd_rf.RData")

# Visualize CV results
png(filename=paste("../figs/cv_result_rf.png"))
plot(x=ntree, y=err_cv_rf, type="l", ylab="error rate",main="Random Forest")
dev.off()

# Choose the best parameter value from visualization
best_ntree <- 300


############# Retrain model with tuned parameters ##############

# train the model with the entire training set
tm_train_rf <- system.time(fit_train_rf <- rf_train(dat_train=feature, label_train=label, ntree=best_nt
save(fit_train_rf, file="../output/fit_train_rf.RData")


### Make prediction
tm_test_rf <- system.time(pred_test_rf <- rf_test(fit_train = fit_train_rf, dat_test=feature))
save(pred_test_rf, file="../output/pred_test_rf.RData")
```

Step 5: Nerual Network

```
# Load features and label
feature <- fread("../output/hog_feature+sift.csv", header = TRUE)
label <- fread("../data/labels.csv")
label <- c(t(label))

######### Tuning parameters #########

#### Ignore this section if optimal training parameter for hidden layers already known
```

```r
#### hiddenLayers_origFeat <- 5
#### hiddenLayers_newFeat <- 3
#### As found in our tuning shown below

# Tune parameter number of hidden layers

layers <- c(1,2,5,10,20,40,100)

cv <- vector("list", length(layers))
i <- 1
impr <- TRUE

while (i < length(cv)) {
  cv[[i]] <- nn_cv(feature, label, K=5, hiddenLayers=layers[i])
  i = i+1
}

q <- unlist(cv)
q2 <- q[c(TRUE,FALSE)]
plot(q2, type='l')


# Visualize CV results
q <- unlist(cv)
q2 <- q[c(TRUE,FALSE)]
plot(y=q2, x=layers[1:6], type='l', xlab="Number of Neurons in Hidden Layer", ylab="5-Fold Avg CV Error
png(filename=paste("../figs/cv_result_nn.png"))
dev.off()

####
#### Begin here if known training parameter
####

# Choose the best parameter value from visualization

hiddenLayers_origFeat <- 5
hiddenLayers_newFeat <- 3

# train the model with the entire training set
fit_train_nn <- nn_train(train = feature, y = label, hiddenLayers = hiddenLayers_newFeat)
save(fit_train_nn, file="../output/fit_train_nn.RData")

# qq <- nn_cv(feature, label, K=5, hiddenLayers=3)


### Make prediction
# ?? fit_train_nn <- file("../../output/fit_train_nn.RData")
pred_test_nn <- nn_test(fit_train_nn, testData)
save(pred_test_nn, file="../output/pred_test_nn.RData")
```

Prediction

```r
test(dat_test)
```