Project 3-Dogs, Fried Chicken or Blueberry Muffins?

Group7: Vassily Carantino, Xin Gao, Lin Han, Yijia Li, Qian Shi

Nov 1, 2017

Step 0: Prepare all needed packages

```
# list.of.packages <- c("gbm","caret","randomForest","EBImage","xgboost","OpenImageR"
, "dplyr")
# new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Pac
kage"])]
# if(length(new.packages))
#   {
#    install.packages(new.packages)
#    source("https://bioconductor.org/biocLite.R")
#    biocLite("EBImage")
#   }
library("gbm")
```

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

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

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

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

```
## Loaded gbm 2.1.3
```

```
library("ggplot2")
library("caret")
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:survival':
##
##     cluster
```

```
library("randomForest")
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library("EBImage")
library("xgboost")
library("OpenImageR")
```

```
##
## Attaching package: 'OpenImageR'
```

```
## The following objects are masked from 'package:EBImage':
##
##     readImage, writeImage
```

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:xgboost':
##
##     slice
```

```
## The following object is masked from 'package:EBImage':
##
##     combine
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

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) process features for test set

```
run.feature = TRUE
run.cv = TRUE
run.feature.train = TRUE
run.feature.test = TRUE
```

Step 2: Construct new visual features

```
# source("../lib/feature.R")
# set.seed(1)
#
# img_dir<- "../data/training_set/images/"
# tm_feature<- c()
# if(run.feature)
# {
#   tm_feature <- system.time(rgb_feature <- feature(img_dir, export=TRUE))
# }
# cat("Time for constructing features is ", tm_feature[3], "s \n")
#
# label_train<- read.csv("../data/training_set/label_train.csv")[, 2]
# names(label_train)<- "y"
# rgb_feature$y<- label_train
#
# training<- sample_frac(rgb_feature, 0.7, replace=FALSE)
# testing<- setdiff(rgb_feature, training, 'rows')
#
# write.csv(training, file="../output/rgb_new_training.csv")
# write.csv(testing, file="../output/rgb_new_testing.csv")
```

Step 3: Models Training and Parameters Selection

Training Gradient Boosting Model (baseline) and XGboost model on original SIFT features by cross validation.

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

Baseline Model: GBM

Parameters training on Original 5000 SIFT features

- Load SIFT features

```
train_SIFT<- readRDS('../output/training.RData')
X_trainSIFT<- train_SIFT[, -5001]
y_trainSIFT<- train_SIFT$y
```

- Choice for parameter values To find the best parameters for Gradient Boosting Model (GBM), we set interaction depth fixed to 3, trees numbers from 200 to 600, and shrinkage values from 0.01 to 0.07.

```
trees_range<- seq(200, 600, 200)
shrinks_range<- seq(0.01, 0.07, 0.02)
```

- Find the best parameters by cross validation

```
# K<- 3
# if(run.cv){
#   best_paras<- train_gbm_para(X_trainSIFT, y_trainSIFT, shrinkage_range, trees_rang
e, K)
# }
```

- The best parameters

```
# if(run.cv){
#   best_shrink<- best_paras$shrink
#   best_trees<- best_paras$ntrees
# }
best_shrink<- 0.07
best_trees<- 600
```

- Visualize the outcomes of CV

```
# if(run.cv){
#   acu_mat<- best_paras$acu_mat
#
#   plot(shrinks_range, acu_mat[, 1],
#        xlab='Shrinkage', ylab="Accuracy", main="Cross Validation Accuracy for GBM",
#        type="b", pch=1, ylim=c(0.76, 0.84))
#
#   for(i in 2:length(trees_range)){
#      lines(shrinks_range, acu_mat[, i], type='b', col=i, pch=i)
#   }
#
#   legend("bottomright", paste("Trees = ", trees_range),
#          col=1:length(trees_range), pch=1:length(trees_range), lty=1, cex=0.7)
# }
```

From the above plot we can see that, the best parameters for GBM model is shrinkage = 0.01 and n.trees = 600. However, this accuracy is greater than 80%. In order to avoid overfitting, we choose parameters corresponding to the second largest accuracy, that is shrinkage = 0.07 and n.trees = 600. Next, we use all kinds of features in this GBM model, like SIFT, HOG, HSV, RGB, using k folds cross-validation to get the average accuarcy. The resuls are showing below: CV_Acu RGB1 89.7% HSV 87.5% HOG 82.2% SIFT 81.0% We can see that GBM with RGB1 features is the best combination.

Fit the best GBM on entire training SIFT and RGB features separately

- GBM & Original 5000 SIFT features

```
train_SIFT<- readRDS('../output/training.RData')
X_trainSIFT<- train_SIFT[, -5001]
y_trainSIFT<- train_SIFT$y
tm_SIFT<- c()

if(run.feature.train){
  tm_SIFT<- system.time(train_SIFT_gbm<- gbm_train(X_trainSIFT, y_trainSIFT, best_shr
ink, best_trees))
}
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 2344: V2344 has no variation.
```

```
cat("Train the best GBM on entire training SIFT features is ", tm_SIFT[3], "s \n")
```

```
## Train the best GBM on entire training SIFT features is  601.381 s
```

```
#saveRDS(train_SIFT_gbm, file="../output/train_SIFT_gbm.RData")
```

- GBM & 1440 RGB features

```
train_RGB<- read.csv('../output/rgb_training1.csv')
X_trainRGB<- train_RGB[, -1441]
y_trainRGB<- train_RGB$y
tm_RGB<- c()

if(run.feature.train){
  tm_RGB<- system.time(train_RGB_gbm<- gbm_train(X_trainRGB, y_trainRGB, best_shrink,
best_trees))
}
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 91: V91 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 101: V101 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 102: V102 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 103: V103 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 111: V111 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 112: V112 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 113: V113 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 114: V114 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 221: V221 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 222: V222 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 231: V231 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 232: V232 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 233: V233 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 234: V234 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 341: V341 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 342: V342 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 351: V351 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 352: V352 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 353: V353 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 354: V354 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 461: V461 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 462: V462 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 463: V463 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 471: V471 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 472: V472 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 473: V473 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 474: V474 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 582: V582 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 591: V591 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 592: V592 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 593: V593 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 594: V594 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 711: V711 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 712: V712 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 832: V832 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 833: V833 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 834: V834 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 952: V952 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 953: V953 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 965: V965 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1083: V1083 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1085: V1085 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1086: V1086 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1202: V1202 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1203: V1203 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1205: V1205 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1206: V1206 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1216: V1216 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1322: V1322 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1323: V1323 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1324: V1324 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1325: V1325 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1326: V1326 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1327: V1327 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1335: V1335 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1336: V1336 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1337: V1337 has no variation.
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 1348: V1348 has no variation.
```

```
cat("Train the best GBM on entire training RGB features is ", tm_RGB[3], "s \n")
```

```
## Train the best GBM on entire training RGB features is  166.071 s
```

```
#saveRDS(train_RGB_gbm, file="../output/train_RGB_gbm.RData")
```

Prefer Model: Xgboost

Parameters training on 1440 RGB features

- Load RGB features

```
train_RGB<- read.csv('../output/rgb_feature1.csv')
X_trainRGB<- train_RGB[, -1441]
y_trainRGB<- train_RGB$y
```

- Find the best parameters by cross validation

```
# K<- 5
# if(run.cv){
#   xgbcv_result <- xgb_para(X_trainRGB, y_trainRGB, K)
# }
```

- The best parameters

```
# if(run.cv){
#   best_par <- xgbcv_result[2]
#   par_best <- list(max_depth = best_par[[1]]$max_depth, eta = best_par[[1]]$eta,
#                    nrounds = 100, gamma = 0,
#                    nthread = 2, subsample = 0.5,
#                    objective = "multi:softprob", num_class = 3)
# }
par_best <- list(max_depth = 3, eta = 0.3,
                 nrounds = 100, gamma = 0,
                 nthread = 2, subsample = 0.5,
                 objective = "multi:softprob", num_class = 3)
```

- Visualize the outcomes of CV

```
# if(run.cv){
#   error_mat <- 1-xgbcv_result[[1]]
#   plot(c(0.1,0.3,0.5), error_mat[1,],
#        xlab='eta', ylab='CV Accuracy', main='Cross Validation Accuracy',
#        type='b', pch=1, ylim=c(0.85,0.95))
#   for(i in 2:3){
#     lines(c(0.1,0.3,0.5), error_mat[i,], type='b', col=i, pch=i)
#   }
#   legend('bottomright', paste('max_depth=', c(3,5,7)),
#          col=1:3, pch=1:3, lty=1, cex=0.6)
# }
```

For Xgboost Model, we tested parameters first, max_depth from (3,5,7), and eta from (0.1,0.3,0.5). From the above plot, we can get the best parameters are, max_depth = 3 and eta = 0.3, since it has the highest accuracy. Next, we used all kinds of features into this Xgboost model, like SIFT, HOG, HSV, RGB, using k folds cross-validation to get the accuarcy. The resuls are showing below: CV_Acu RGB1 90.7% HSV 89.3% HOG 84.8% SIFT 79.8% We can see that Xgboost Model with RGB1 features is the best combination.

Fit the best Xgboost model on entire training RGB features

```
train_RGB<- read.csv('../output/rgb_training1.csv')
X_trainRGB<- train_RGB[, -1441]
y_trainRGB<- train_RGB$y

tm_train<- c()

if(run.feature.train){
  tm_train <- system.time(train_RGB_xgb <- xgb_train(X_trainRGB, y_trainRGB, par_best
))
}

cat("Train the best Xgboost on entire training RGB features is ", tm_train[3], "s \n"
)
```

```
## Train the best Xgboost on entire training RGB features is  24.318 s
```

```
#saveRDS(train_RGB_xgb, file="../output/train_RGB_xgb.RData")
```

Step 4: Make prediction on testing datasets

Fit the best GBM model on testing SIFT and RGB features separately

- GBM & Original 5000 SIFT features

```
tm.SIFT_test_gbm<- c()
if(run.feature.test){
  test_SIFT<- readRDS('../output/testing.RData')
  X_test<- test_SIFT[, -5001]
  y_test<- test_SIFT$y

  #load(file="../output/train_SIFT_gbm.RData")

  tm.SIFT_test_gbm<- system.time(pred_SIFT_gbm<- gbm_test(train_SIFT_gbm, X_test))
  acu_SIFT_gbm<- sum(pred_SIFT_gbm == y_test) / length(y_test)

  cat("GBM accuracy for original SIFT features is ", acu_SIFT_gbm, "\n")
  cat("Time for fitting GBM on testing SIFT features is ", tm.SIFT_test_gbm[3], "s \n
")

  #saveRDS(pred_SIFT_gbm, file="../output/pred_SIFT_gbm.RData")
}
```

```
## GBM accuracy for original SIFT features is  0.8055556
## Time for fitting GBM on testing SIFT features is  6.926 s
```

- GBM & 1440 RGB features

```
tm.RGB_test_gbm<- c()
if(run.feature.test){
   test_RGB<- read.csv('../output/rgb_testing1.csv')
   X_test<- test_RGB[, -1441]
   y_test<- test_RGB$y

   #load(file="../output/train_RGB_gbm.RData")

   tm.RGB_test_gbm<- system.time(pred_RGB_gbm<- gbm_test(train_RGB_gbm, X_test))
   acu_RGB_gbm<- sum(pred_RGB_gbm == y_test) / length(y_test)

   cat("GBM accuracy for RGB features is ", acu_RGB_gbm, "\n")
   cat("Time for fitting GBM on testing RGB features is ", tm.RGB_test_gbm[3], "s \n"
)

   #saveRDS(pred_RGB_gbm, file="../output/pred_RGB_gbm.RData")
}
```

```
## GBM accuracy for RGB features is  0.8788889
## Time for fitting GBM on testing RGB features is  0.33 s
```

Fit the best Xgboost model on testing RGB features

```
tm_test<- c()
if(run.feature.test){
  test_RGB<- read.csv('../output/rgb_testing1.csv')
  X_test<- test_RGB[, -1441]
  y_test<- test_RGB$y

  #load(file="../output/train_RGB_xgb.RData")
  tm_test <- system.time(pred_RGB_xgb <- xgb_test(train_RGB_xgb, X_test))
  cat("Time for fitting Xgboost on testing RGB features is ", tm_test[3], "s \n")

  error <- sum(pred_RGB_xgb != y_test)/length(y_test)
  cat("Xgboost accuracy for RGB features is ", 1-error, "\n")

  #saveRDS(pred_RGB_xgb, file="../output/pred_RGB_xgb.RData")
}
```

```
## Time for fitting Xgboost on testing RGB features is  0.076 s
## Xgboost accuracy for RGB features is  0.8866667
```