# Project 3 - Dogs, Fried Chicken or Blueberry Muffins?

*Group6: Sijian Xuan, Xinyao Guo, Siyi Wang, Xiaoyu Zhou, Pinren Chen*

*Oct 10, 2017*

## Install necessary packages.

## Set the working directory to the image folder.

## Read Data

```
sift.feature=read.csv("../data/sift_feature.csv", header = T)
lbp.feature=read.csv("../data/lbp_feature.csv", header = F)
hog.feature = read.csv("../data/hog_feature.csv")
label=read.csv("../data/trainlabel.csv")
```

## SIFT feature

### Make dataset

```
sift_data=data.frame(cbind(label,sift.feature[,-1]))
test.index=sample(1:3000,500,replace=F)
colnames(sift_data)[2]="y"
sift_data = sift_data[,-1]
test.sift=sift_data[test.index,]
test.x.sift=test.sift[,-1]
train.sift=sift_data[-test.index,]
```

## Baseline model: GBM + SIFT

### Tune parameters:n.trees = 250, shrinkage = 0.1

```
y<-label[,2]
X<-test.x.sift
source("../lib/tune gbm.r")
#These lines of code takes a long like crazy time so I just wrote the results as a csv file,and we read
for(k in 1:length(depths)){
  cat("k=", k, "\n")
  err_cv[k,] <- cv.function(X, y, depths[k], shrinkage=shrinkages, K=5)  #K=5
}
colnames(err_cv) = c("mean of cv.error","sd of cv.error")
rownames(err_cv) = c("depth = 3", "depth = 5", "depth = 7", "depth = 9","depth = 11")
print(err_cv)
write.csv(err_cv,file = "../output/err_cv_for_baseline.csv")
```

```
err_cv_for_baseline = read.csv("../output/err_cv_for_baseline.csv")
print(err_cv_for_baseline)
```

```
##              mean.of.cv.error.sd.of.cv.error
## 1 depth = 3         0.6666667     0.01160699
## 2 depth = 5         0.6666667     0.01457738
## 3 depth = 7         0.6666667     0.01711887
## 4 depth = 9         0.6666667     0.02838231
## 5 depth = 11        0.6666667     0.01443376
```

## Other models + SIFT

**The 5000-dimensional SIFT feature takes a long time to get the results. If PCA is used to do dimension reduction, the accuracy become really low. It makes sense because doing PCA dimension reduction means losing information. As we are pursuing higher accuracy and shorter time at the same time, we started to use other feature extraction methods. With Zhilin's suggestion, we use Local Binary Patterns(LBP), Histogram of oriented gradients(HoG) and Convolutional Neural Network(CNN) to extract features.**

## Local Binary Patterns(LBP)

**Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings.**

**A useful extension to the original operator is the so-called uniform pattern, which can be used to reduce the length of the feature vector and implement a simple rotation invariant descriptor. This idea is motivated by the fact that some binary patterns occur more commonly in texture images than others. A local binary pattern is called uniform if the binary pattern contains at most two 0-1 or 1-0 transitions. For example, 00010000(2 transitions) is a uniform pattern, 01010100(6 transitions) is not. In the computation of the LBP histogram, the histogram has a separate bin for every uniform pattern, and all non-uniform patterns are assigned to a single bin. Using uniform patterns, the length of the feature vector for a single cell reduces from 256 to 59. The 58 uniform binary patterns correspond to the integers 0, 1, 2, 3, 4, 6, 7, 8, 12, 14, 15, 16, 24, 28, 30, 31, 32, 48, 56, 60, 62, 63, 64, 96, 112, 120, 124, 126, 127, 128, 129, 131, 135, 143, 159, 191, 192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254 and 255.**

**We used MATLAB to extract LBP features(adapted codes from Zhilin's work, I added a filter for color image and grayscale image). The column dimension is 59, which is much less than 5000. So it is reasonable that we expect a decreased time usage. The time use is 569.281s.**

**Make LBP dataset**

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

```
lbpdata = data.frame(cbind(label,lbp.feature))
colnames(lbpdata)[2] = "y"
lbpdata = lbpdata[,-1]
test.lbp = lbpdata[test.index,]
test.x.lbp = test.lbp[,-1]
train.lbp = lbpdata[-test.index,]
```

# GBM + LBP

```
#These lines also take long time to run so I just run it once and save it into a csv file and read it t
X = train.lbp[,-1]
y = train.lbp[,1]
for(k in 1:length(depths)){
  cat("k=", k, "\n")
  err_cv[k,] <- cv.function(X, y, depths[k], shrinkage=shrinkages, K=5)  #K=5
}
colnames(err_cv) = c("mean of cv.error","sd of cv.error")
rownames(err_cv) = c("depth = 3", "depth = 5", "depth = 7", "depth = 9","depth = 11")
print(err_cv)
write.csv(err_cv,file = "../output/err_cv_for_GBM+LBP.csv")
```

```
err_cv_for_GBM_LBP = read.csv("../output/err_cv_for_GBM+LBP.csv")
print(err_cv_for_GBM_LBP)
```

```
##            X mean.of.cv.error sd.of.cv.error
## 1  depth = 3        0.6666667    0.030345236
## 2  depth = 5        0.6666667    0.007993053
## 3  depth = 7        0.6666667    0.022882550
## 4  depth = 9        0.6666667    0.015365907
## 5 depth = 11        0.6666667    0.012416387
```

# Some advanced models + LBP

## BPNN

**Tune parameters:size = 1, decay = 0.01**

```
bp.model=train.bp(train.lbp)
bp.pre=test.bp(bp.model,test.x.lbp)
table(bp.pre,test.lbp$y)
```

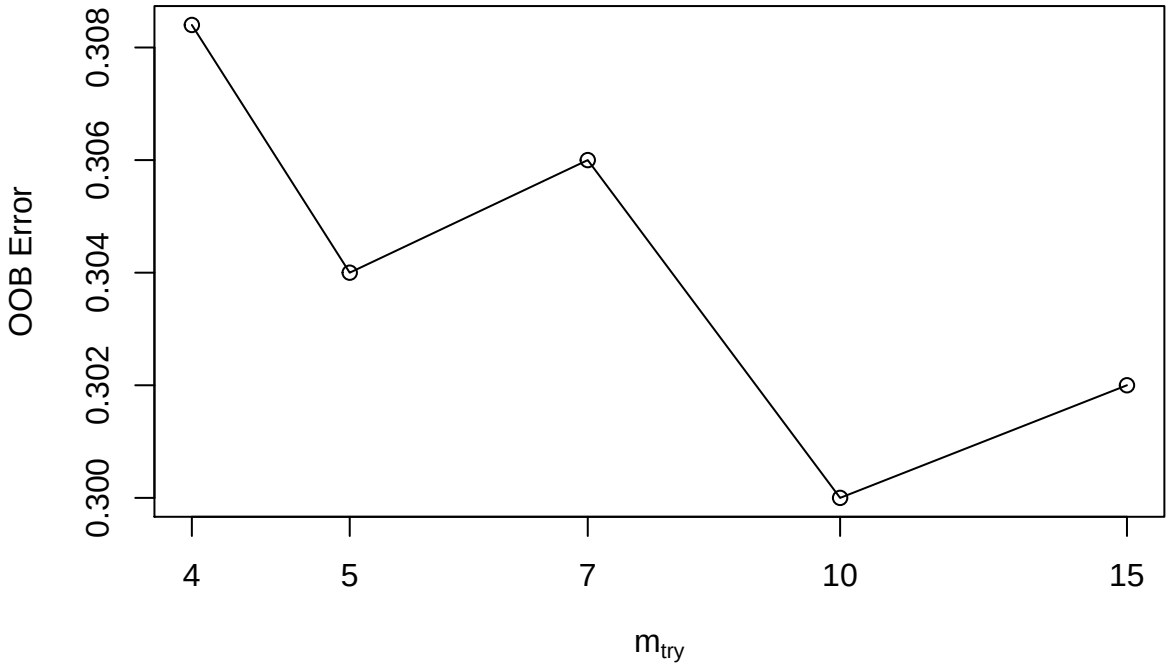
```
##
## bp.pre   0   1   2
##      0 105  47   2
##      1  46  85  34
##      2  11  32 138
```

## Random Forest + LBP

**Tune Parameter: m.try=15**

```
rf.model <- train.rf(train.lbp)
```

```
## mtry = 7  OOB error = 30.6%
## Searching left ...
## mtry = 5     OOB error = 30.4%
## 0.006535948 1e-05
## mtry = 4     OOB error = 30.84%
## -0.01447368 1e-05
## Searching right ...
## mtry = 10    OOB error = 30%
## 0.01315789 1e-05
## mtry = 15    OOB error = 30.2%
## -0.006666667 1e-05
```




```
rf.pre=test.rf(rf.model,test.x.lbp)
table(rf.pre,test.lbp$y)
```

```
##
## rf.pre   0   1   2
##      0 126  42  14
##      1  23  98  35
##      2  13  24 125
```

## SVM + LBP

**Tune Parameters: cost=10, gamma=0.01**

```
svm.model <- train.svm(train.lbp)
svm.pre=test.svm(svm.model,test.x.lbp)
table(svm.pre,test.lbp$y)
```

```
##
## svm.pre   0   1   2
##       0 138  18   8
##       1  17 128  13
##       2   7  18 153
```

## Logistic Regression + LBP

```
log.model <- train.log(train.lbp)
log.pre=test.log(log.model,test.x.lbp)
table(log.pre, test.lbp$y)
```

```
##
## log.pre   0   1   2
##       0 117  22   9
##       1  29 117  22
##       2  16  25 143
```

## Xgboost

### Tune model and choose parameters

```
#These lines also take long time to run; we print the result instead of running it every time.
trainnn<-as.matrix(train.hog)
testtt<-as.matrix(test.hog)
dtrain=xgb.DMatrix(data=trainnn[,-1],label=trainnn[,1])

NROUNDS = c(500,1000)
ETA = c(0.3)
MAX_DEPTH = c(3,4,5,6)

cv.xgb <- function(X.train, y.train, K, NROUNDS, ETA, MAX_DEPTH){
  for (nround in NROUNDS){
    for (eta in ETA){
      for (max_depth in MAX_DEPTH){
        n <- length(y.train)
        n.fold <- floor(n/K)
        s <- sample(rep(1:K, c(rep(n.fold, K-1), n-(K-1)*n.fold)))
        cv.acc <- rep(NA, K)

        for (i in 1:K){
          train.data <- X.train[s != i,]
          train.label <- y.train[s != i]
          test.data <- X.train[s == i,]
          test.label <- y.train[s == i]

          param <- list("objective" = "multi:softmax",
                        "eval_metric" = "mlogloss",
                        "num_class" = 3, 'eta' = eta, 'max_depth' = max_depth)

          dtrain=xgb.DMatrix(data=train.data,label=train.label)
```

```
          bst <- xgb.train(data = dtrain,  param = param, nrounds = nround)
          pred <- predict(bst, newdata = test.data)

          cv.acc[i] <- mean(pred == test.label)
        }
        print(paste("------Mean 5-fold cv accuracy for nround=",nround,",eta=",eta,",max_depth=",max_de
                    "------",mean(cv.acc)))
        key = c(nround,eta,max_depth)
        CV_ERRORS[key] = mean(cv.acc)

      }
    }
  }
}

CV_ERRORS = list()
cv.xgb(trainnn[,-1], trainnn[,1], 5, NROUNDS, ETA, MAX_DEPTH)
#Results
#[1] "------Mean 5-fold cv accuracy for nround= 500 ,eta= 0.3 ,max_depth= 3 ------ 0.8044"
#[1] "------Mean 5-fold cv accuracy for nround= 500 ,eta= 0.3 ,max_depth= 4 ------ 0.7836"
#[1] "------Mean 5-fold cv accuracy for nround= 500 ,eta= 0.3 ,max_depth= 5 ------ 0.8"
#[1] "------Mean 5-fold cv accuracy for nround= 500 ,eta= 0.3 ,max_depth= 6 ------ 0.7936"
#[1] "------Mean 5-fold cv accuracy for nround= 1000 ,eta= 0.3 ,max_depth= 3 ------ 0.7928"
#[1] "------Mean 5-fold cv accuracy for nround= 1000 ,eta= 0.3 ,max_depth= 4 ------ 0.7984"
#[1] "------Mean 5-fold cv accuracy for nround= 1000 ,eta= 0.3 ,max_depth= 5 ------ 0.796"
#[1] "------Mean 5-fold cv accuracy for nround= 1000 ,eta= 0.3 ,max_depth= 6 ------ 0.7952"
```

**Tuned xgboost model**

```
xgboost.model = train.xgboost(train.lbp)
xgboost.pre = test.xgboost(xgboost.model,test.lbp)
table(xgboost.pre, test.lbp$y)
```

```
##
## xgboost.pre   0   1   2
##           0 131  27  11
##           1  19 108  30
##           2  12  29 133
```

# Histograms of Orientation Gradients

**Algorithm Overview**

**Local shape information often well described by the distribution of intensity gradients or edge directions even without precise information about the location of the edges themselves.**

**Divide image into small sub-images: "cells" Cells can be rectangular (R-HOG) or circular (C-HOG)**

**Accumulate a histogram of edge orientations within that cell**

**The combined histogram entries are used as the feature vector describing the object**

**To provide better illumination invariance (lighting, shadows, etc.) normalize the cells across larger regions incorporating multiple cells: "blocks"**

**Why HOG?**

**Capture edge or gradient structure that is very characteristic of local shape**

**Relatively invariant to local geometric and photometric transformations**

Within cell rotations and translations do not affect the HOG values

Illumination invariance achieved through normalization

**The spatial and orientation sampling densities can be tuned for different applications**

For human detection (Dalal and Triggs) coarse spatial sampling and fine orientation sampling works best

For hand gesture recognition (Fernandez-Llorca and Lacey) finer spatial sampling and orientation sampling is required

**We extracted HoG feature using R, it can be found in "./lib/hog_feature_extraction.r".**

# Some advanced models + HoG

## Note we are not using GBM anymore because it takes so long time to run.

**GBM, wish you well in the future.**

**Make HoG dataset**

```
hogdata = data.frame(cbind(label,hog.feature[,-1]))
colnames(hogdata)[2] = "y"
hogdata = hogdata[,-1]
test.hog = hogdata[test.index,]
test.x.hog = test.hog[,-1]
train.hog = hogdata[-test.index,]
```

## BPNN + HoG

**Tune parameters:size = 1, decay = 0.01**

```
bp.model=train.bp(train.hog)
bp.pre=test.bp(bp.model,test.x.hog)
table(bp.pre,test.hog$y)
```

```
##
## bp.pre   0   1   2
##      0 148  23  13
##      1  14  94  51
```

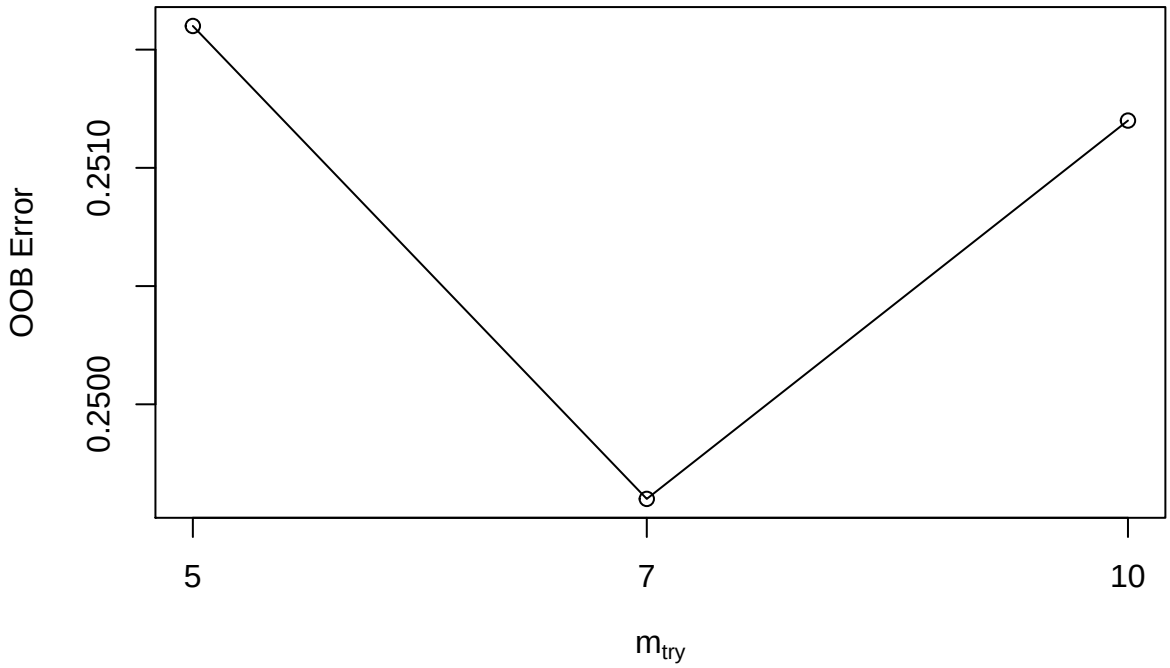
```
##       2   0  47 110
```

## Random Forest + HoG

**Tune Parameter: m.try=15**

```
rf.model <- train.rf(train.hog)
```

```
## mtry = 7  OOB error = 24.96%
## Searching left ...
## mtry = 5     OOB error = 25.16%
## -0.008012821 1e-05
## Searching right ...
## mtry = 10    OOB error = 25.12%
## -0.006410256 1e-05
```




```
rf.pre=test.rf(rf.model,test.x.hog)
table(rf.pre,test.hog$y)
```

```
##
## rf.pre   0   1   2
##      0 141  37  23
##      1  18 110  30
##      2   3  17 121
```

## SVM + HoG

**Tune Parameters: cost=10, gamma=0.01**

```
svm.model <- train.svm(train.hog)
svm.pre=test.svm(svm.model,test.x.hog)
table(svm.pre,test.hog$y)
```

```
##
## svm.pre   0   1   2
##       0 148  13  15
##       1   9 123  28
##       2   5  28 131
```

### Logistic Regression + HoG

**Tune logistic regression model tuning result is using the default interation 100, we tried 500 but accuracy is getting worse.**

### LR model

```
log.model <- train.log(train.hog)
log.pre=test.log(log.model,test.x.hog)
table(log.pre, test.hog$y)
```

```
##
## log.pre   0   1   2
##       0 145  15  19
##       1  12 128  24
##       2   5  21 131
```

### Xgboost model

```
xgboost.model = train.xgboost(train.hog)
xgboost.pre = test.xgboost(xgboost.model,test.hog)
table(xgboost.pre, test.hog$y)
```

```
##
## xgboost.pre   0   1   2
##           0 142  20  14
##           1  13 126  29
##           2   7  18 131
```

## Cross Validation Error Rate

```
#These lines also take long time to run so I just run it once and save it into a csv file and read it t
source("../lib/cross_validation.R")
cv.error.lbp =cv.function(lbpdata,5)
cv.error.hog = cv.function(hogdata,5)
print (cv.error.lbp)
print(cv.error.hog)
write.csv(cv.error.lbp,"../output/cv.error.lbp.csv")
write.csv(cv.error.hog,"../output/cv.error.hog.csv")
```

```
cv.error.lbp = read.csv("../output/cv.error.lbp.csv")
cv.error.hog = read.csv("../output/cv.error.hog.csv")
print(cv.error.lbp)
```

```
##   X        bp        rf       svm  logistic   xgboost
## 1 1 0.3506667 0.3016667 0.1956667 0.2393333 0.2453333
```

```
print(cv.error.hog)
```

```
##   X        bp        rf       svm  logistic   xgboost
## 1 1 0.3143333 0.2546667 0.2106667 0.2203333 0.1993333
```

## Final Train & Time

```
c=system.time(bp <- train.bp(lbpdata))
d=system.time(rf <- train.rf(lbpdata))
e=system.time(svm <- train.svm(lbpdata))
f=system.time(logistic <- train.log(lbpdata))
g = system.time(xgboost <- train.xgboost(lbpdata))
time_lbp=list(bp=c,rf=d,svm=e,logistic=f,xgboost = g)

c=system.time(bp <- train.bp(hogdata))
d=system.time(rf <- train.rf(hogdata))
e=system.time(svm <- train.svm(hogdata))
f=system.time(logistic <- train.log(hogdata))
g = system.time(xgboost <- train.xgboost(hogdata))
time_hog=list(bp=c,rf=d,svm=e,logistic=f,xgboost = g)

print(time_lbp)
#$bp
#   user  system elapsed
#  0.383   0.002   0.384

#$rf
#   user  system elapsed
# 59.177   0.151  59.439

#$svm
#   user  system elapsed
#  1.752   0.007   1.760

#$logistic
#   user  system elapsed
#  0.304   0.003   0.307

#$xgboost
#   user  system elapsed
# 28.457   0.061  28.636
print(time_hog)
#$bp
#   user  system elapsed
#  0.266   0.001   0.266

#$rf
#   user  system elapsed
# 49.142   0.172  49.444
```

```
#$svm
#   user  system elapsed
#  1.518   0.010   1.528

#$logistic
#   user  system elapsed
#  0.271   0.003   0.274

#$xgboost
#   user  system elapsed
# 22.785   0.079  22.976
```

## Final Model

**We choose SVM + LBP and xgboost + HoG as our final mdoel. They both have high accuracy as well as short time usage.**