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

```
#training time is kind of long so I write it in csv and
#read it for faster in knit process
features <- train.sift
dim(features)
label_train<-train.sift
dim(label_train)
y<-label_train[,1]
X<-features[,-1]
source("../lib/tune gbm.r")
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")
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)
```

```
##              X mean.of.cv.error sd.of.cv.error
## 1  depth = 3            0.2700     0.01542725
## 2  depth = 5            0.2552     0.02124147
## 3  depth = 7            0.2696     0.02511573
## 4  depth = 9            0.2632     0.02410809
## 5 depth = 11            0.2548     0.02100476
```

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

```r
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,]
```

SVM + LBP (Best one candidate)

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
##       0 142  10   9
##       1  28 124  12
##       2  14  21 140
```

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.

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,]
```

Xgboost model(best model candidate)

```
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
##           0 168  15   7
##           1  12 112  24
##           2   4  28 130
```

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
```

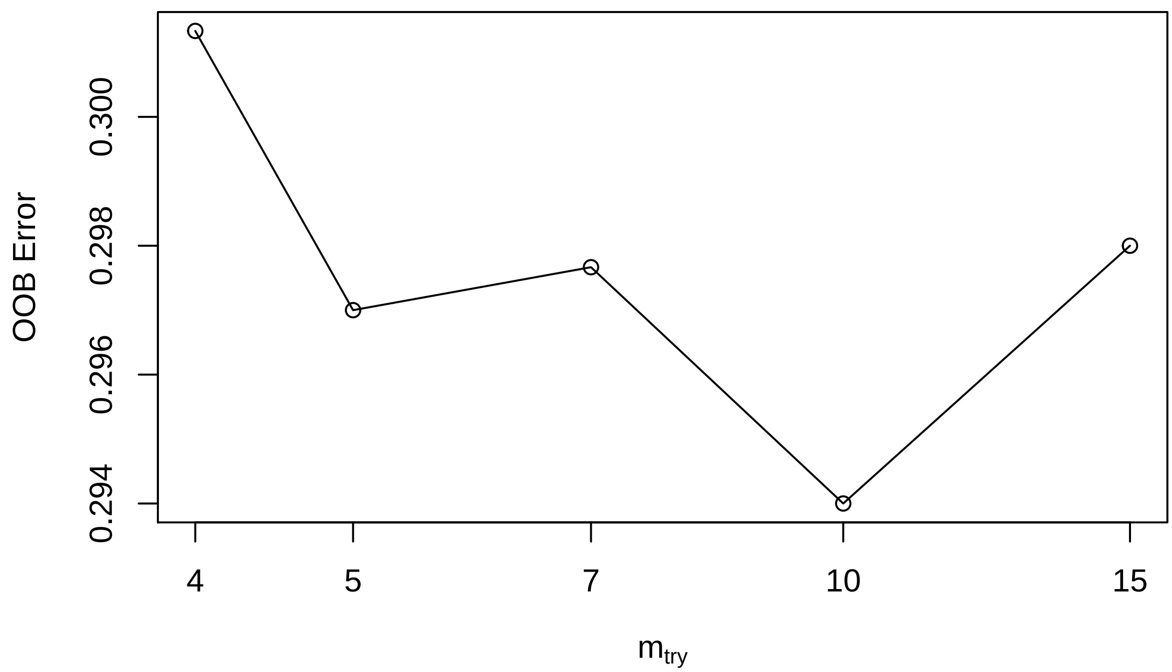
```r
print(cv.error.hog)
```

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

Final Train & Time

```r
c=system.time(bp <- train.bp(lbpdata))
d=system.time(rf <- train.rf(lbpdata))
```

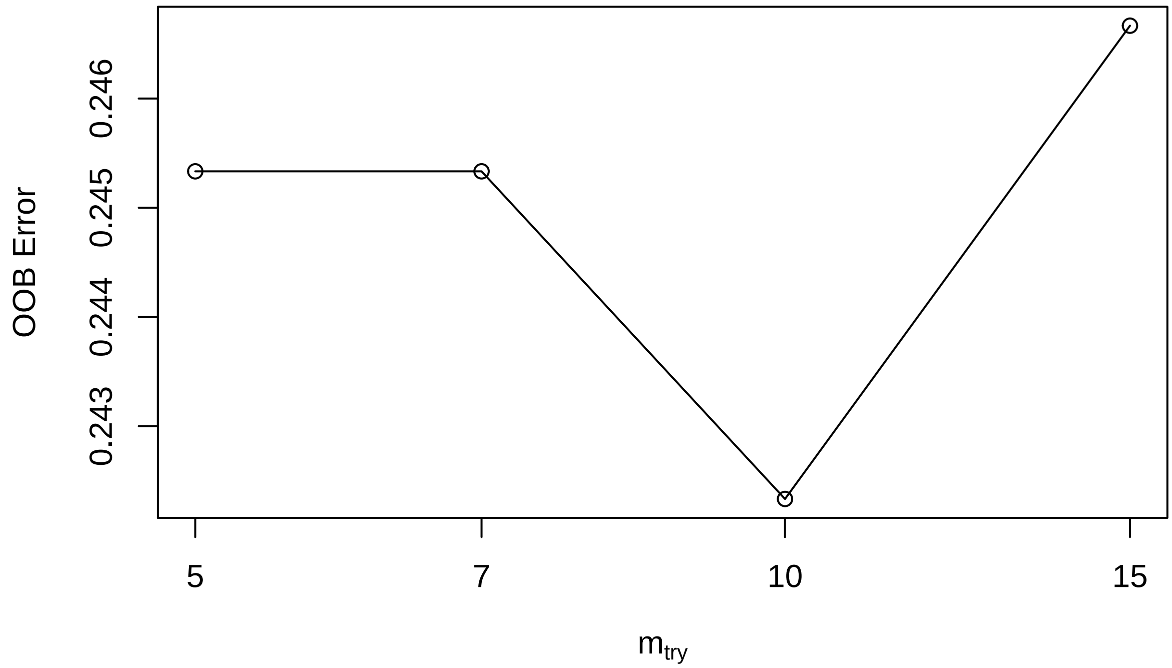
```
## mtry = 7  OOB error = 29.77%
## Searching left ...
## mtry = 5     OOB error = 29.7%
## 0.002239642 1e-05
## mtry = 4     OOB error = 30.13%
## -0.01459035 1e-05
## Searching right ...
## mtry = 10    OOB error = 29.4%
## 0.01010101 1e-05
## mtry = 15    OOB error = 29.8%
## -0.01360544 1e-05
```




```r
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))
```

```
## mtry = 7  OOB error = 24.53%
## Searching left ...
## mtry = 5     OOB error = 24.53%
## 0 1e-05
## Searching right ...
## mtry = 10    OOB error = 24.23%
## 0.01222826 1e-05
## mtry = 15    OOB error = 24.67%
## -0.01788171 1e-05
```




```
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
##   0.436   0.003   0.440
##
## $rf
##    user  system elapsed
##  60.353   0.177  60.841
##
## $svm
##    user  system elapsed
##   1.972   0.027   2.205
##
## $logistic
##    user  system elapsed
##   0.324   0.005   0.352
##
```

```
## $xgboost
##    user  system elapsed
##  29.538   0.169  32.836
```

```r
print(time_hog)
```

```
## $bp
##    user  system elapsed
##   0.281   0.002   0.304
##
## $rf
##    user  system elapsed
##  51.520   0.293  56.545
##
## $svm
##    user  system elapsed
##   1.657   0.015   1.776
##
## $logistic
##    user  system elapsed
##   0.282   0.005   0.324
##
## $xgboost
##    user  system elapsed
##  22.855   0.106  25.171
```

Final Model

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