Puppy or Fried Chicken?

Summary:

In this project, we created a classifier for grey images of puppies versus images of fried chickens. We tried different features (SIFT, LBP) and different classifiers (GBM, BP Neural Networks, SVM, Random Forest, Logistic Regression and Majority Vote). When pursuing low error rate, we also keep an eye on processing time.

Install Packages

```
packages.used=c("gbm", "caret","DMwR" ,"nnet","randomForest","EBImage","e1071")

# check packages that need to be installed.
packages.needed=setdiff(packages.used,
                        intersect(installed.packages()[,1],
                                  packages.used))
# install additional packages
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}
```

Read Data

```
sift.features=read.csv("../data/sift_features.csv", header = T)
lbp=read.csv("../data/lbp.csv", header = F)
label=read.csv("../data/labels.csv")

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

```
## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.1

## Loading required package: ggplot2

##
## Attaching package: 'caret'

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

## Loading required package: grid

## 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
```

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

Train and Validate set

```
all=data.frame(cbind(label,t(sift.features)))
set.seed(1)
test.index=sample(1:2000,400,replace=F)
colnames(all)[1]="y"
test.sift=all[test.index,]
test.x.sift=test.sift[,-1]
train.sift=all[-test.index,]
```

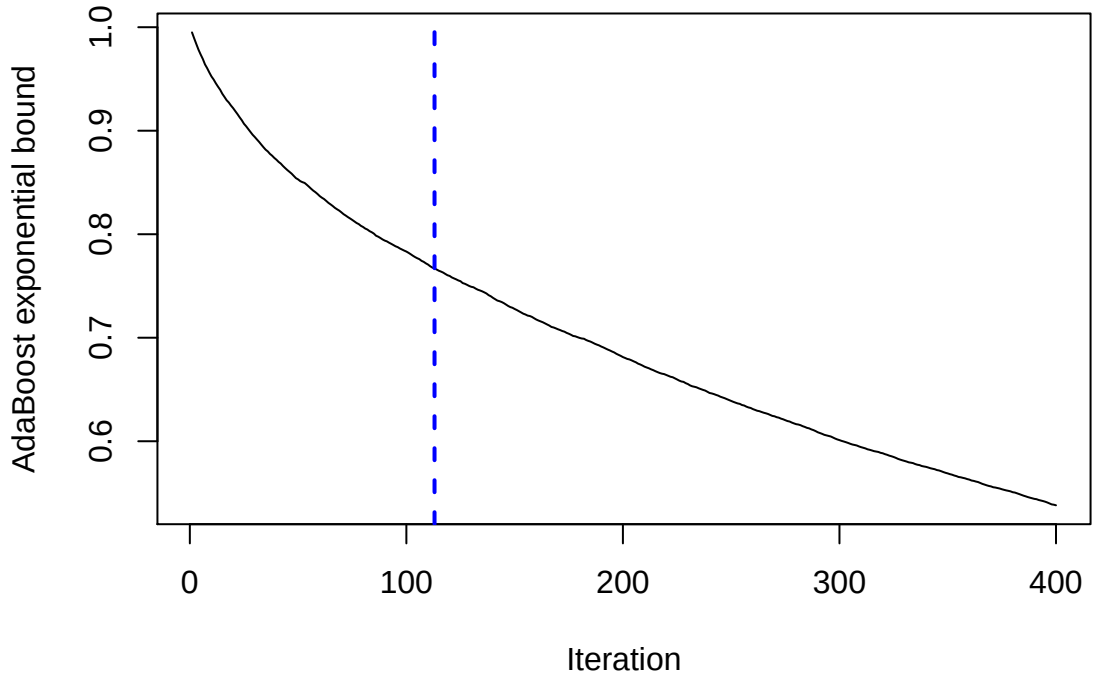
Baseline: GBM + SIFT

Tune parameters: n.trees=113, shrinkage=0.1

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 4141: X4141 has no variation.
```

```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```



```
##
## baseline.pre   0   1
##            0 153  55
##            1  49 143
```

Other models + SIFT

We tried to apply other models on 5000-dimensional SIFT features. When the accuracy rate increased to ~80%, the processing time increased dramatically. So we used PCA to reduce the dimensional of SIFT features. However, when the dimension decreased to 500, the results of models didn't seem satisfying. Thus, we started to explore other features.

Local Binary Patterns (LBP)

- **Divide the examined window into cells (e.g. 16x16 pixels for each cell).**
- **For each pixel in a cell, compare the pixel to each of its 8 neighbors**
- **Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number.**
- **Compute the histogram over the cell. This histogram can be seen as a 256-dimensional feature vector.**
- **Optionally normalize the histogram to 59-dimensional feature vector.**
- **Concatenate histograms of all cells. This gives a feature vector for the entire window.**

Then we extracted LBP features in MATLAB. The processing time of 2000 images is 210s. The column dimension of the result feature matrix is 59.

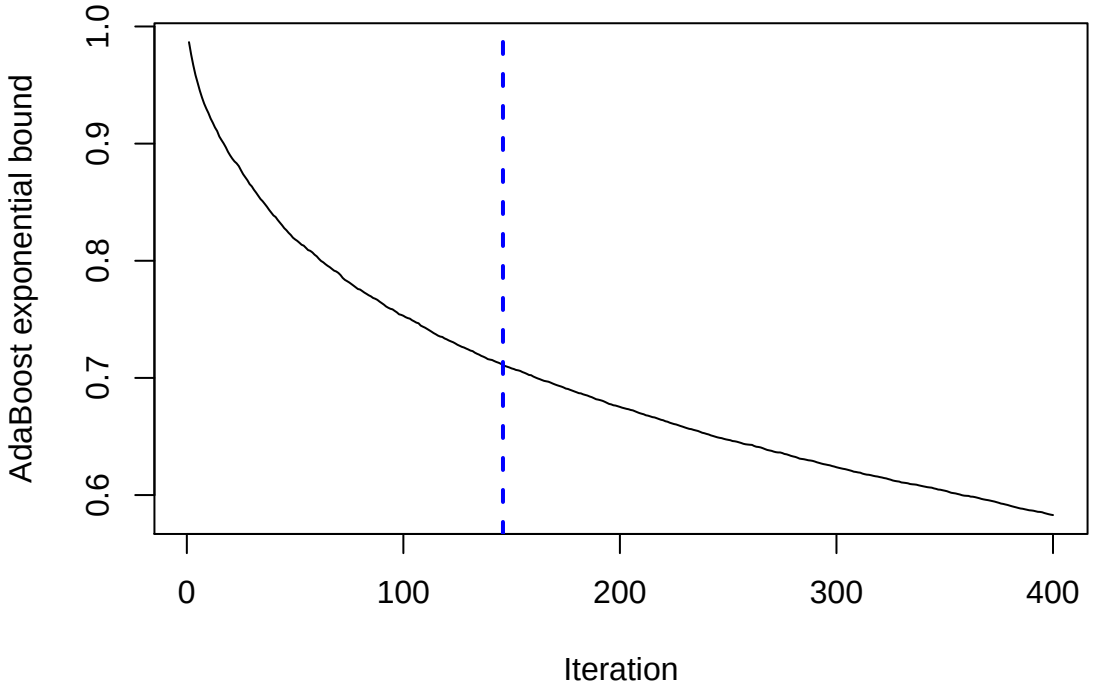
```
new.data=data.frame(cbind(label,lbp))
colnames(new.data)[1]="y"
test=new.data[test.index,]
test.x=test[,-1]
train=new.data[-test.index,]
```

GBM + LBP

Tune parameters: n.trees=146, shrinkage=0.1

```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```




```
##
## baseline.pre   0   1
##            0 161  43
##            1  41 155
```

Advanced Models + LBP

BP Neural Networks + LBP

Tune Parameters: size = 1, decay = 0.01

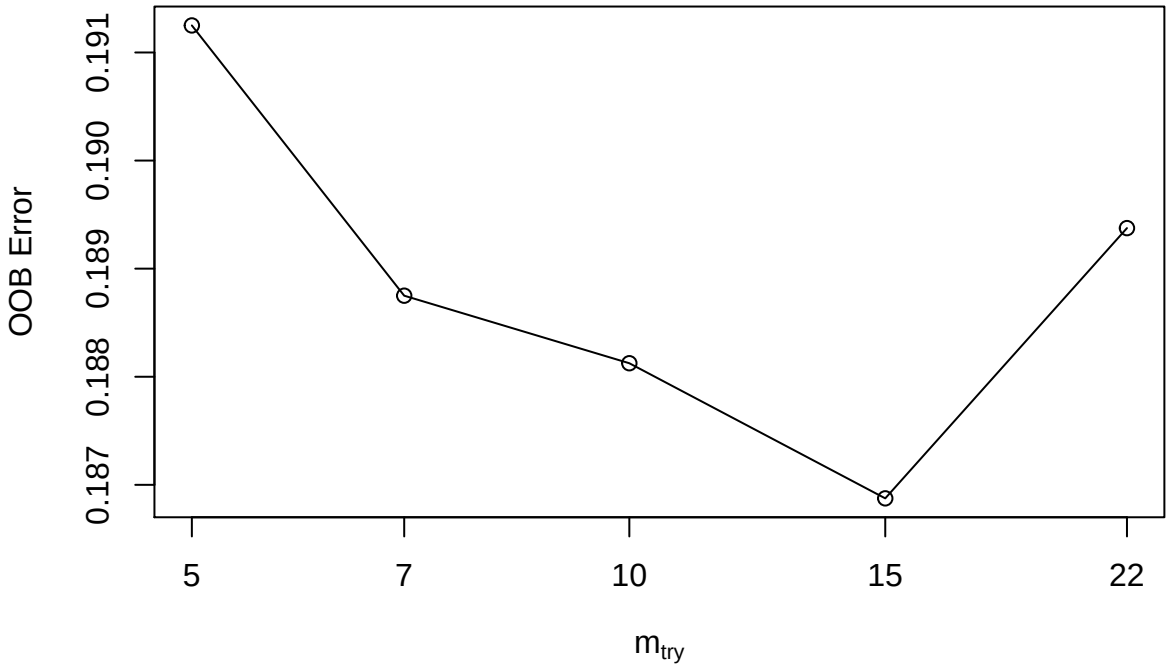
```
bp.model=train.bp(train)
bp.pre=test.bp(bp.model,test.x)
table(bp.pre,test$y)
```

```
##
## bp.pre   0   1
##      0 175  23
##      1  27 175
```

Random Forest + LBP

Tune Parameter: m.try=15

```
## mtry = 7  OOB error = 18.88%
## Searching left ...
## mtry = 5     OOB error = 19.12%
## -0.01324503 1e-05
## Searching right ...
## mtry = 10    OOB error = 18.81%
## 0.003311258 1e-05
## mtry = 15    OOB error = 18.69%
## 0.006644518 1e-05
## mtry = 22    OOB error = 18.94%
## -0.01337793 1e-05
```




```
##    user  system elapsed
##  37.981   0.431  38.940
```

```
##
## rf.pre   0   1
##      0 169  36
##      1  33 162
```

SVM + LBP

Tune Parameters: cost=10, gamma=0.01

```
system.time(svm.model <- train.svm(train))
```

```
##    user  system elapsed
##   0.578   0.014   0.602
```

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

```
##
## svm.pre   0   1
##       0 186  25
##       1  16 173
```

Logistic Regression+ LBP

```
system.time(log.model <- train.log(train))
```

```
##    user  system elapsed
##   0.079   0.005   0.085
```

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

```
##
## log.pre   0   1
##       0 177  25
##       1  25 173
```

Majority Vote (NN, SVM, Log) + LBP

```
pre=(as.numeric(as.character(bp.pre))+as.numeric(as.character(log.pre))+as.numeric(as.character(svm.pre
pre=ifelse(pre>=2,1,0)
table(pre,test$y)
```

```
##
## pre   0   1
##   0 177  23
##   1  25 175
```

Cross Validation Error Rate

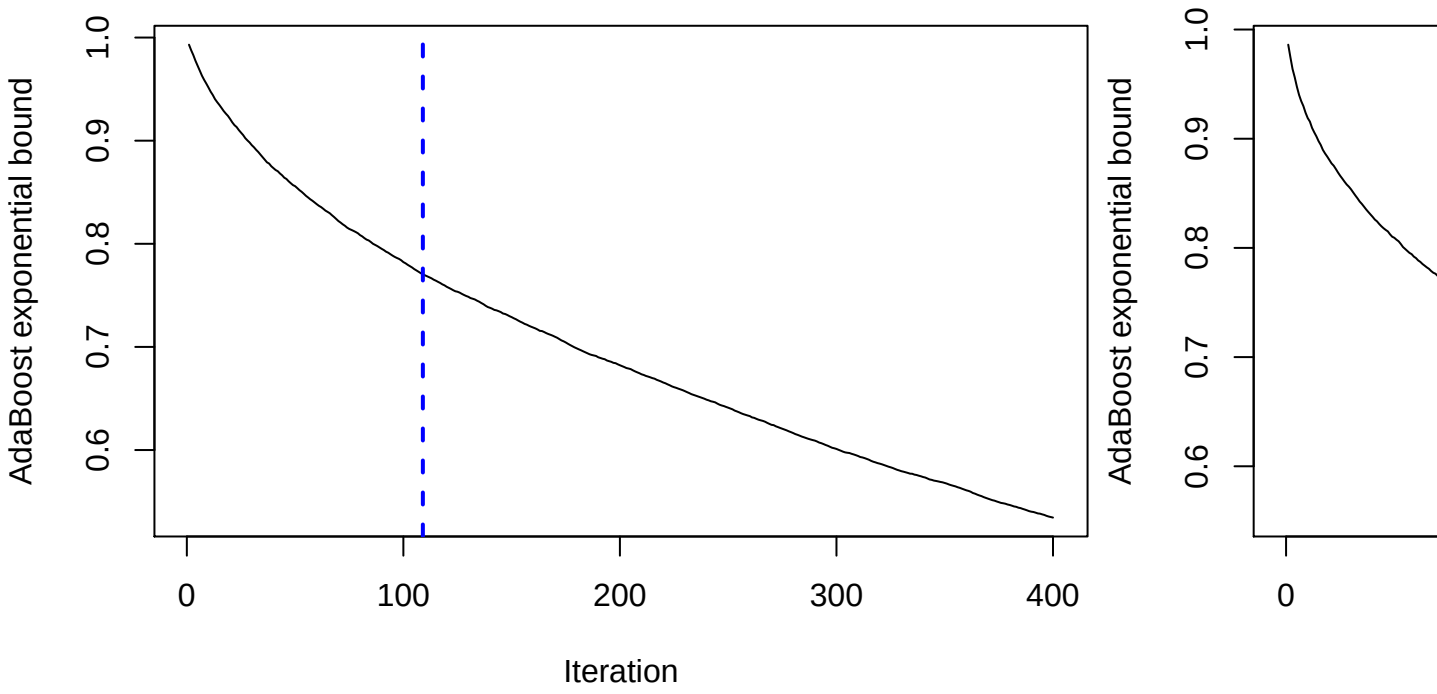
```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 4141: X4141 has no variation.
```

```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```

```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```

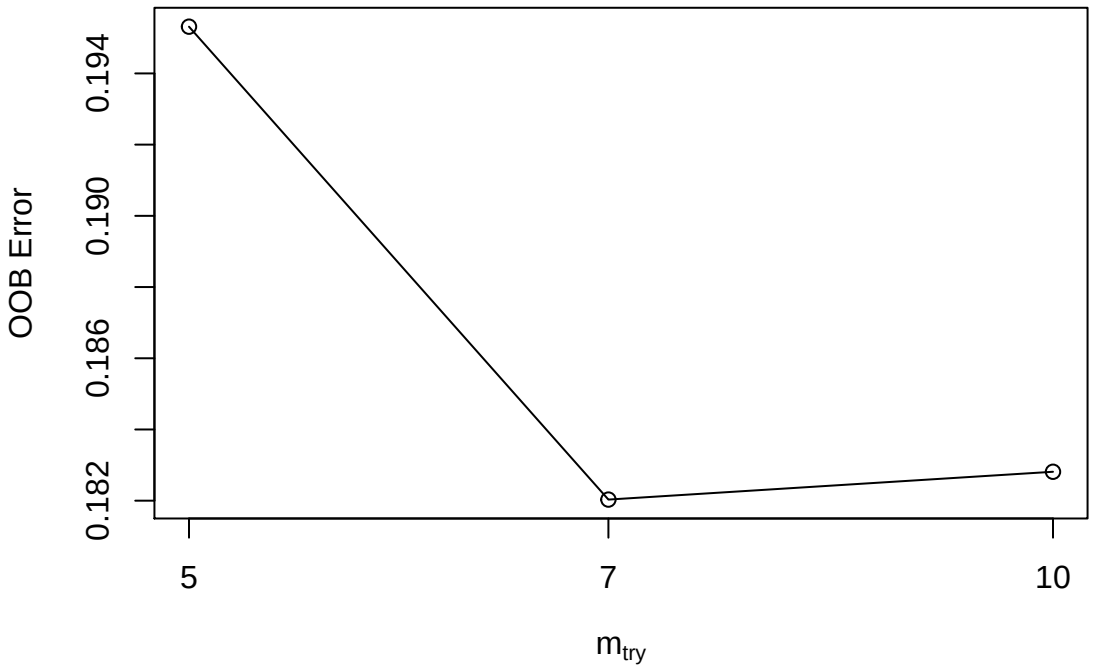



```
## mtry = 7  OOB error = 18.2%
## Searching left ...
## mtry = 5     OOB error = 19.53%
## -0.07296137 1e-05
## Searching right ...
## mtry = 10    OOB error = 18.28%
## -0.004291845 1e-05
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 4141: X4141 has no variation.
```

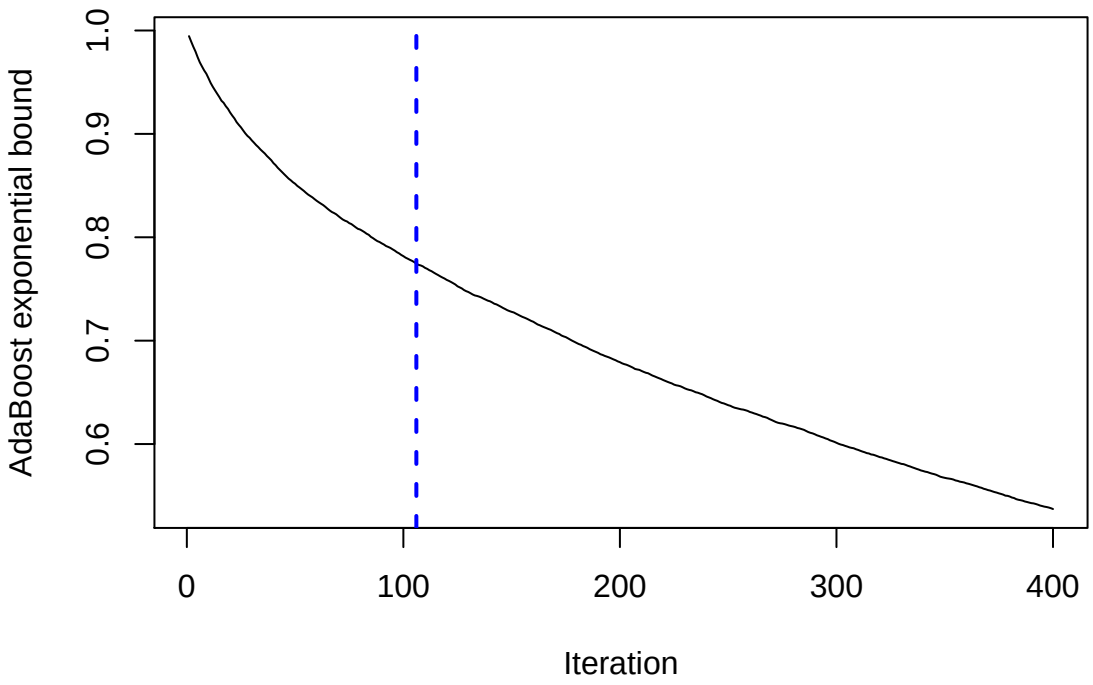
```
## Using OOB method...
```

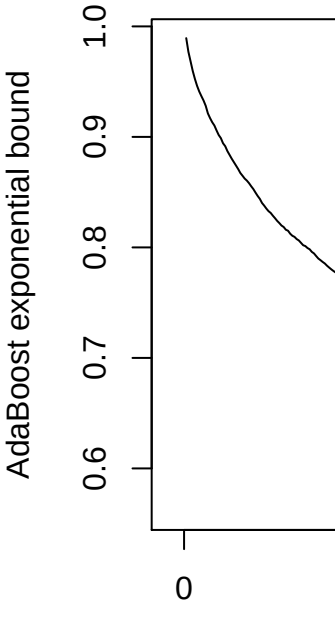
```
## Warning in gbm.perf(gbm1): OOB generally underestimates the optimal number
## of iterations although predictive performance is reasonably competitive.
## Using cv.folds>0 when calling gbm usually results in improved predictive
## performance.
```




```
## Using OOB method...

## Warning in gbm.perf(gbm1): 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.
```





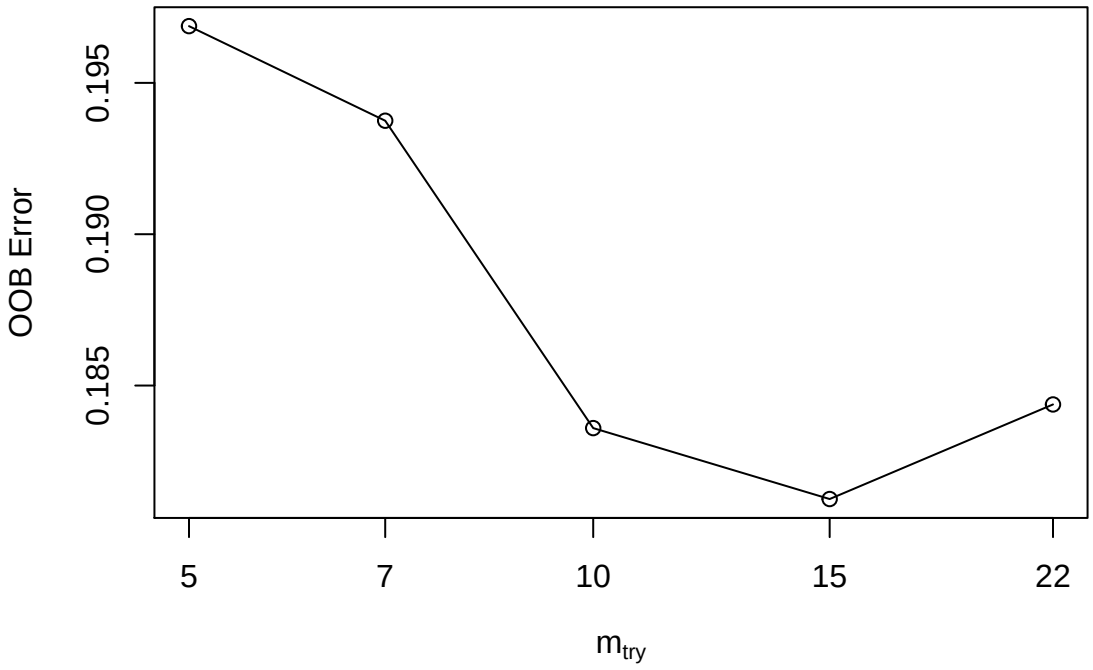

```
## mtry = 7  OOB error = 19.38%
## Searching left ...
## mtry = 5     OOB error = 19.69%
## -0.01612903 1e-05
## Searching right ...
```

```
## mtry = 10    OOB error = 18.36%
## 0.05241935 1e-05
## mtry = 15    OOB error = 18.12%
## 0.01276596 1e-05
## mtry = 22    OOB error = 18.44%
## -0.01724138 1e-05
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 4141: X4141 has no variation.
```

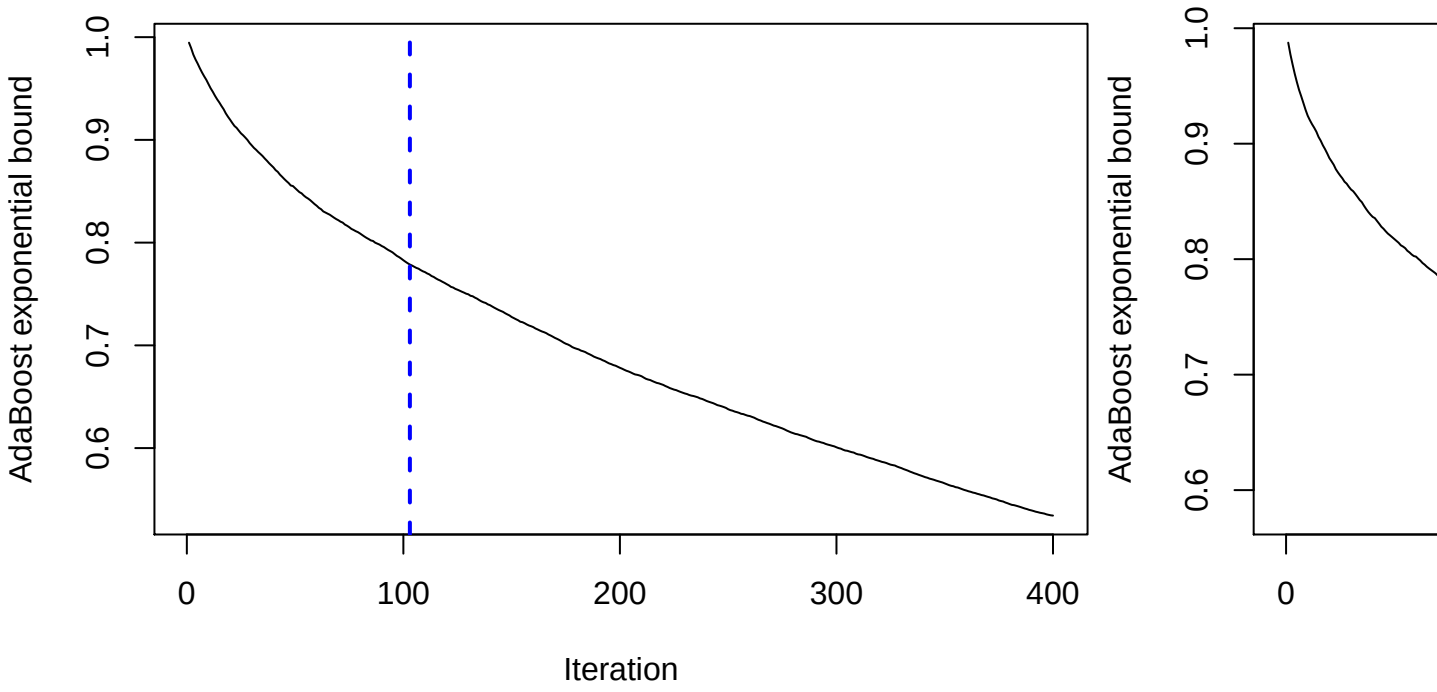
```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```




```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```

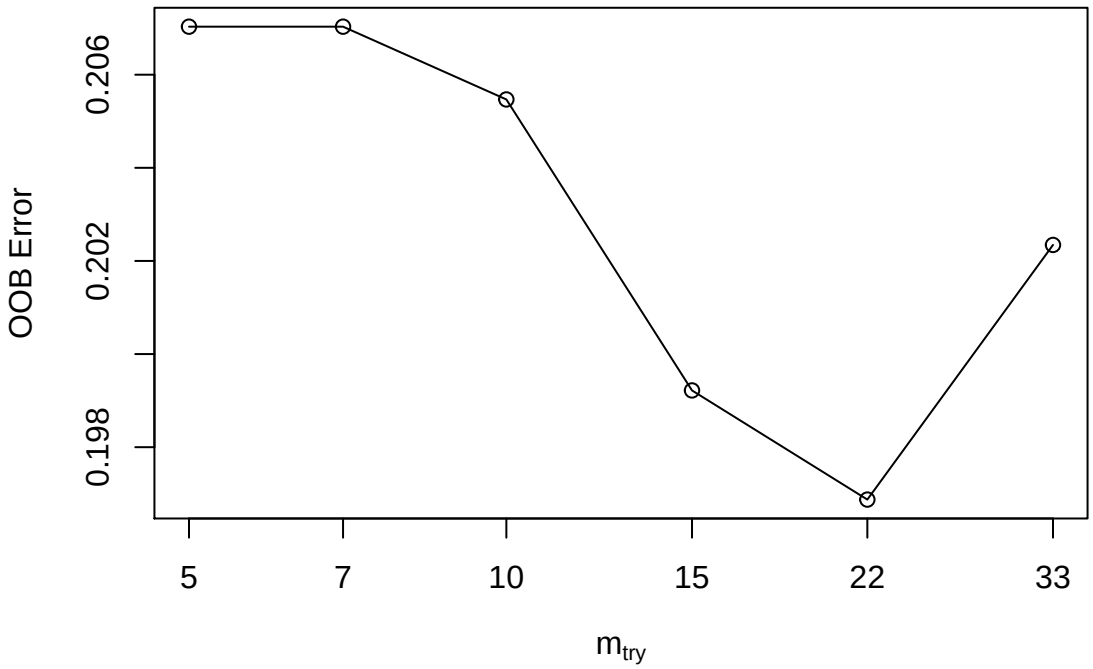



```
## mtry = 7  OOB error = 20.7%
## Searching left ...
## mtry = 5     OOB error = 20.7%
## 0 1e-05
## Searching right ...
## mtry = 10    OOB error = 20.55%
## 0.00754717 1e-05
## mtry = 15    OOB error = 19.92%
## 0.03041825 1e-05
## mtry = 22    OOB error = 19.69%
## 0.01176471 1e-05
## mtry = 33    OOB error = 20.23%
## -0.02777778 1e-05
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 4141: X4141 has no variation.
```

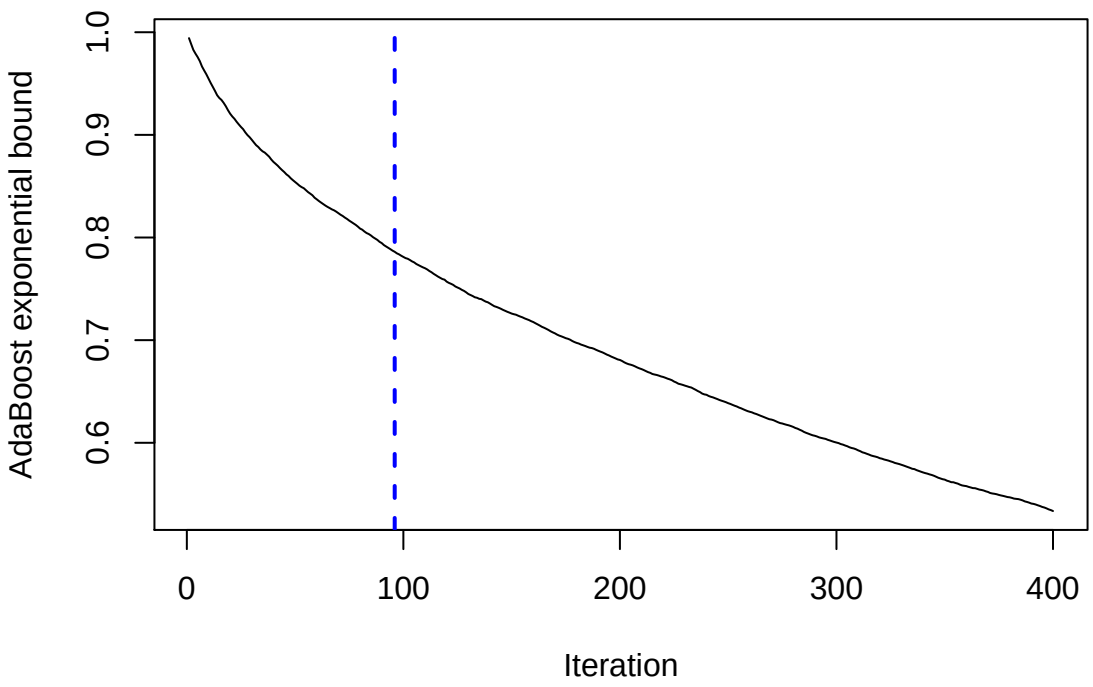
```
## Using OOB method...
```

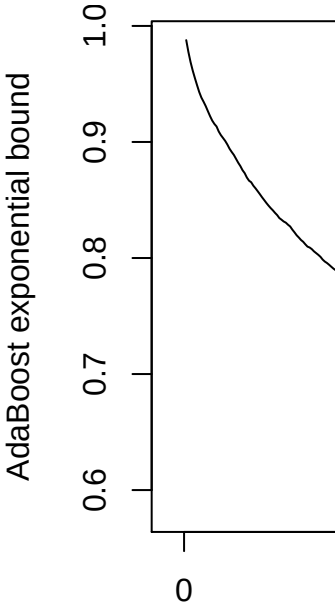
```
## Warning in gbm.perf(gbm1): OOB generally underestimates the optimal number
## of iterations although predictive performance is reasonably competitive.
## Using cv.folds>0 when calling gbm usually results in improved predictive
## performance.
```




```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```





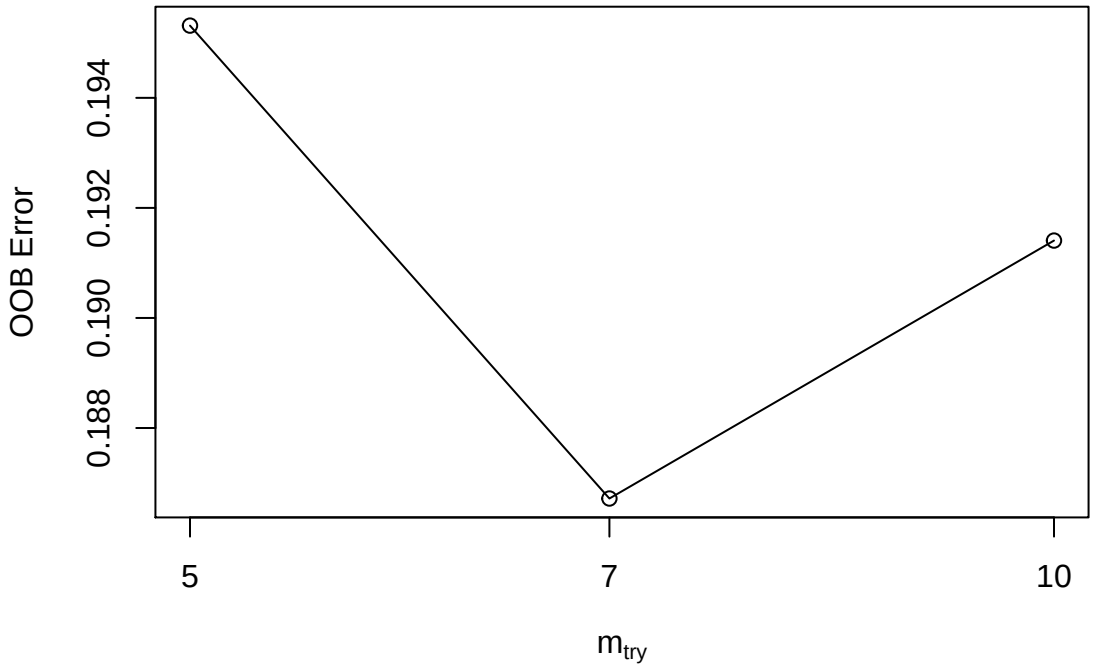

```
## mtry = 7  OOB error = 18.67%
## Searching left ...
## mtry = 5     OOB error = 19.53%
## -0.0460251 1e-05
## Searching right ...
```

```
## mtry = 10    OOB error = 19.14%
## -0.0251046 1e-05
```

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 4141: X4141 has no variation.
```

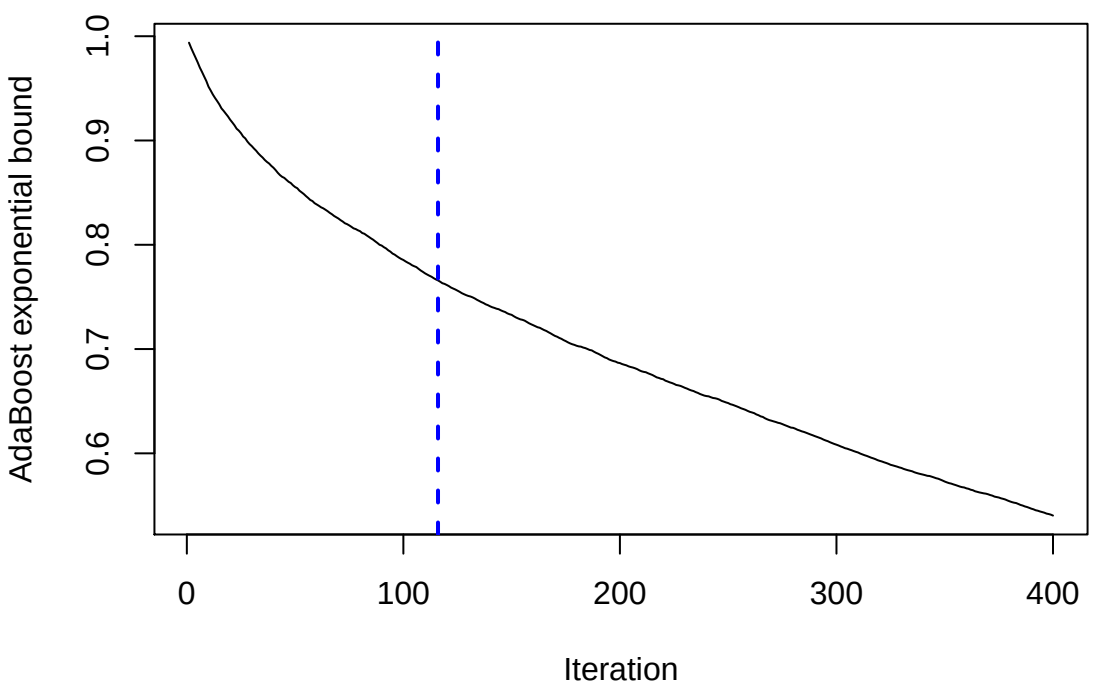
```
## Using OOB method...
```

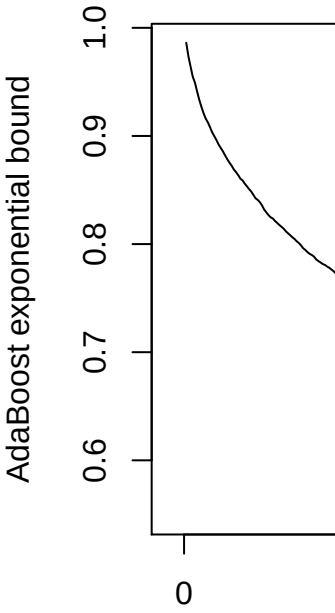
```
## Warning in gbm.perf(gbm1): OOB generally underestimates the optimal number
## of iterations although predictive performance is reasonably competitive.
## Using cv.folds>0 when calling gbm usually results in improved predictive
## performance.
```



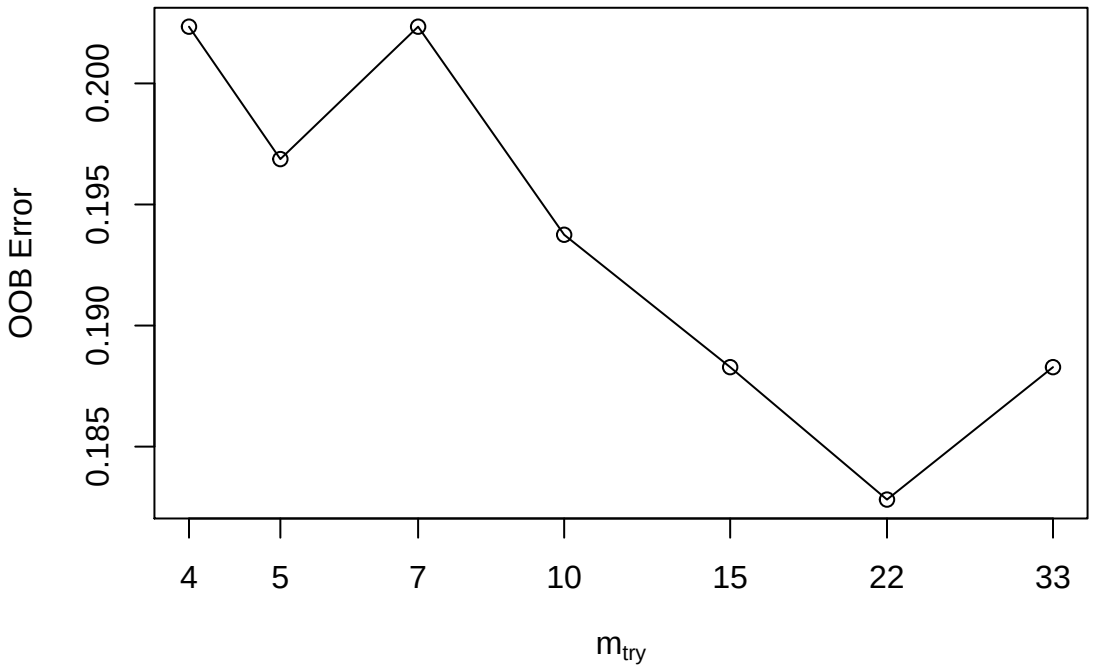

```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```





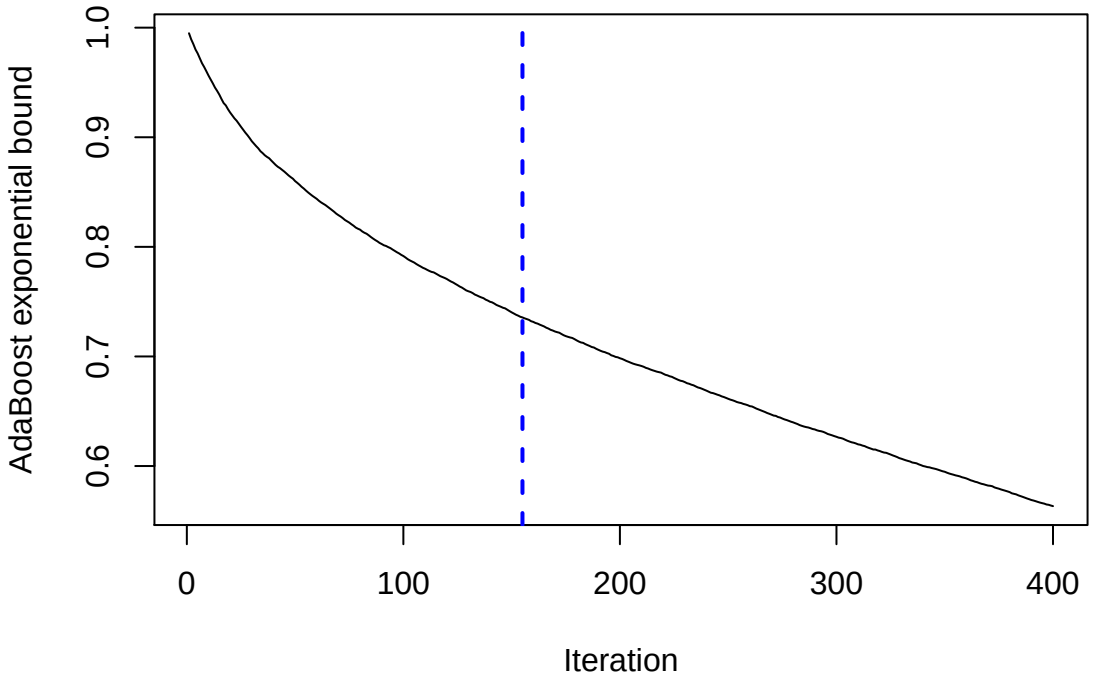

```
## mtry = 7  OOB error = 20.23%
## Searching left ...
## mtry = 5     OOB error = 19.69%
## 0.02702703 1e-05
## mtry = 4     OOB error = 20.23%
## -0.02777778 1e-05
## Searching right ...
## mtry = 10    OOB error = 19.38%
## 0.01587302 1e-05
## mtry = 15    OOB error = 18.83%
## 0.02822581 1e-05
## mtry = 22    OOB error = 18.28%
## 0.02904564 1e-05
## mtry = 33    OOB error = 18.83%
## -0.02991453 1e-05
```



Final Train & Time

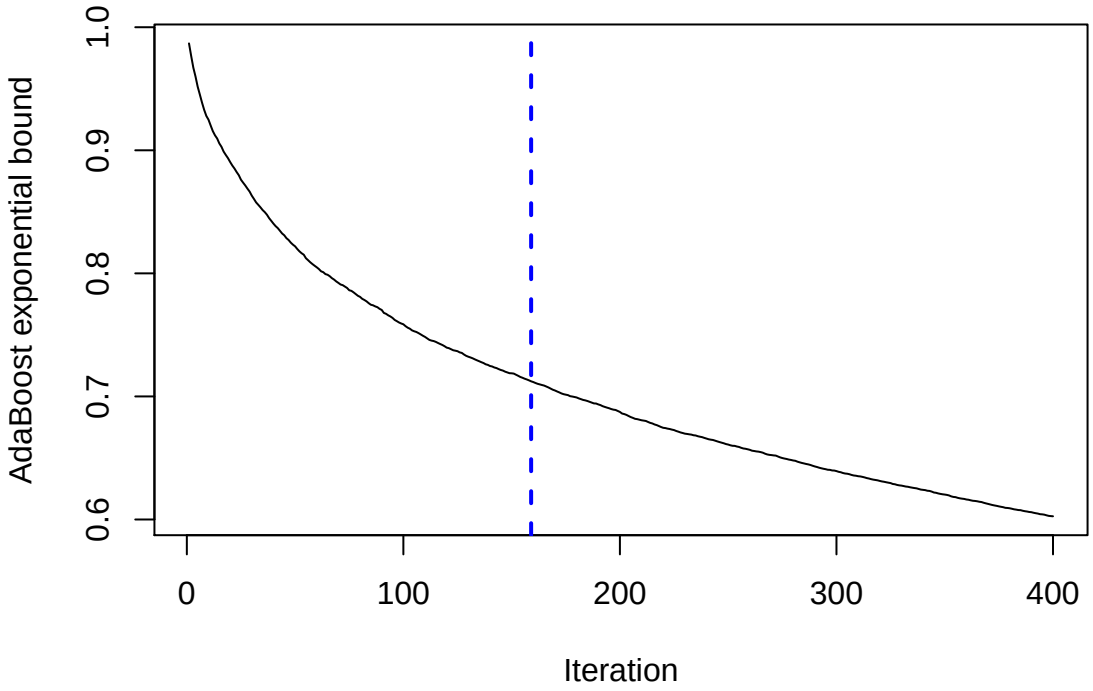
```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```

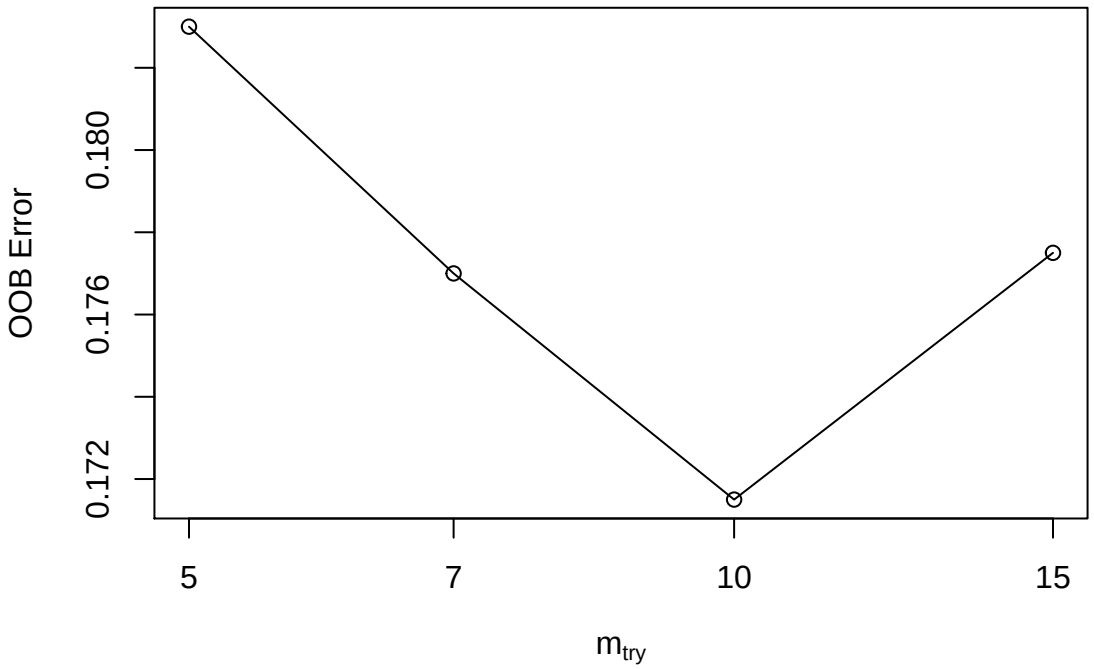



```
## Using OOB method...
```

```
## Warning in gbm.perf(gbm1): 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.
```




```
## mtry = 7  OOB error = 17.7%
## Searching left ...
## mtry = 5     OOB error = 18.3%
## -0.03389831 1e-05
## Searching right ...
## mtry = 10    OOB error = 17.15%
## 0.03107345 1e-05
## mtry = 15    OOB error = 17.75%
## -0.03498542 1e-05
```




```
library(EBImage)
summary=readImage("../figs/summary.png")
display(summary)
```

Summary of Models with LBP features

	Parameters	CV Error Rate	Training Time
GBM	n.trees=146 shrinkage=0.1	23%	1.08s
Neural Networks	size = 1 decay = 0.01	13.5%	0.23s
Radom Forest	m.try=15	19.75%	35.43s
SVM	cost=10 gamma=0.01	12.06%	0.86s
Logistic Regression		13.56%	0.09s
Majority Vote **(NN, SVM, Log)**		13.12%	

Final Model

We choose Majority Vote as our final model. Since training time of each model is very short, time won't be a problem for majority vote. Although we sacrifice little accuracy, We can get a more robust model.