Lecturer: Prof. Homrighausen Scribe: Soo-Young Kim

# 1 Adaboost

## 1.1 AdaBoost: The controversy

**Claim**: Boosting is another version of bagging.

The early versions of Boosting involved (weighted) resampling. Therefore, it was initially speculated that a connection with bagging explained its performance. However, boosting continues to work well when

- The algorithm is trained on weighted data rather than on sampling with weights This removes the randomization component that is essential to bagging
- Weak learners are used that have high bias and low variance This is the opposite of what is prescribed for bagging

**Claim**: Boosting fits an adaptive additive model which explains its effectiveness.

The previous results appeared in Friedman et al. [1] and claimed to have 'solved' the mystery of boosting. A crucial property of boosting is that is essentially never over fits. However, the additive model view really should translate into intuition of 'over fitting is a major concern,' as it is with additive models.

As adaBoost fits an additive model in the base classifier, it cannot have higher order interactions than the base classifier.

For instance, a stump would provide a purely additive fit. (It only splits on one variable. In general, the complexity of a tree can be interpreted as the number of included interactions).

It stands to reason, then, if the Bayes' rule is additive in a similar fashion, stumps should perform well in Boosting

A recent paper investigating this property did substantial simulations using underlying purely additive models Mease and Wyner [2].

Here is an example figure 1 from their paper: Ultimately, interpretations are just modes of human comprehension

The value of the insight is whether it provides fruitful thought about the idea

From this perspective, AdaBoost fits an additive model. However, many of the other connections are still of debatable value

Now we will discuss two current, popular algorithms and their **R** implementations. For detail, see

- GBM (https://cran.r-project.org/web/packages/gbm/gbm.pdf)
- XGBoost (https://cran.r-project.org/web/packages/xgboost/xgboost.pdf)

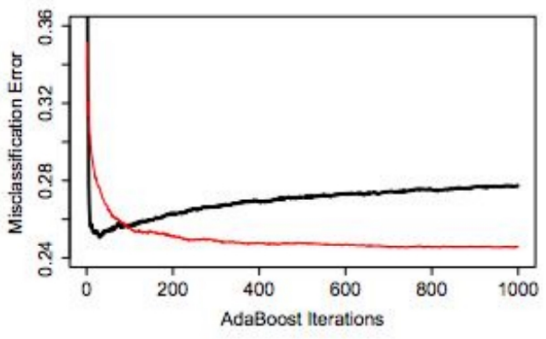


Figure 1: Black, bold line: Stumps. Red, thin line: 8-node trees

# 2 Gradient Boosting Machines (GBM)

Recall: AdaBoost effectively uses forward stagewise minimization of the exponential loss function

GBM takes this idea and

- generalizes to other loss functions
- adds subsampling
- includes methods for choosing $B$
- reports variable importance measures

## 2.1 GBM: loss functions

- gaussian: squared error
- laplace: absolute value
- bernoulli: logistic
- adaboost: exponential
- multinomial: more than one class (unordered)
- poisson: Count data
- coxph: For right censored, survival data

## 2.2 GBM: subsampling

Early implementations of AdaBoost randomly sampled the weights ($w$)

This wasn't essential and has been altered to use deterministic weights

Friedman (2002) introduced stochastic gradient boosting that uses a new subsample at each boosting iteration to find and project the gradient

This has two possible benefits

- Reduces computations/storage (But increases read/write time)
- Can improve performance

You can expect performance gains when both of the following occur:

- There is a small sample size
- The base learner is complex

This suggests the usual 'variance reduction through lowering covariance" interpretation

The effect is complicated, though as subsampling

- increases the variance of each term in the sum
- decreases the covariance of each term in the sum

## 2.3 GBM: choosing $B$

There are three built in methods:

- Independent test set: using the nTrain parameter to say 'use only this amount of data for training'
  Be sure to uniformly permute your data set first.
- Out-of-bag (OOB) estimation: If bag.fraction is $> 0$, then gbm use OOB at each iteration to find a good $B$ (Note: OOB tends to select a too-small $B$)
- $K$-fold cross validation (CV): It will fit cv.folds+1 models
  The '+1' is the fit on all the data that is reported

## 2.4 GBM: variable importance measure

For tree-based methods, there are two variable importance measures:

- relative.influence
- permutation.test.gbm
  This is currently labeled experiemental

These have similar definition relative to bagging, however they use all of the data instead of the OOB

## 2.5 GBM: sample code

```
gbm(Ytrain~.,data=Xtrain,
    distribution="bernoulli", #bernoulli when the response has only
    2 unique values
```

```
n.trees=500,            #The total number of trees to fit.
shrinkage=0.01,         #learning rate(\lambda)
interaction.depth=3,    #The maximum depth of variable interactions.
bag.fraction = 0.5,     #subsampling
n.minobsinnode = 10,    #the minimum number of observations in the trees terminal nodes
cv.folds = 3,           #Number of cross-validation folds to perform: K
keep.data=TRUE,
verbose=TRUE,           #If TRUE, gbm will print out progress and performance indicators.
n.cores=2)              #The number of CPU cores to use.
```

## 2.6 Distributed computing hierarchy

$$\text{Server} \to \text{Node} \to \text{CPU/Processor} \to \text{Core} \to \text{Hyperthreding}$$

Example: A server might have

- 64 nodes
- 2 processors per node
- 16 cores per processor
- hyper threading

The goal is to somehow allocate a job so that these resources are used efficiently. Jobs are composed of threads, which are specific computations

## 2.7 Hyperthreading

$$\text{Core} \longrightarrow \text{Hyperthreding} \begin{cases} \nearrow \text{ Virtual Core} \\ \searrow \text{ Virtual Core} \end{cases}$$

Developed by Intel, Hypertheading allows for each core to pretend to be two cores

This works by trading off computation and read-time for each core

## 2.8 Boosting: Learning slow

It is best to set the learning rate at a small number.

This is usually calibrated by the computational demands of the problem.

A good strategy is to pick a number, say .001

Run with n.trees relatively small and see how long it takes

Keep adding trees with gbm.more. If this is taking too long, increase the learning rate

# 3 XGBoost(Extreme Gradient Boosting)

## 3.1 XGboost: Advances

It has advances related to gbm

- Sparse matrices: Can use sparse matrices as inputs In fact, it has its own matrix-like data structure that is recommended
- OpenMP: Incorporates OpenMP on Windows/Linux OpenMP is a message passing parallelization paradigm for shared memory parallel programming
- Loss functions: You can specifiy your own loss/evaluation functions You need to use xgb.train(or XGB data structure) for this

## 3.2 Tree complexity

The base classifier's complexity must be fixed

As previously stated, it is usually chosen to be $M \in \{4, \ldots, 8\}$

This choice controls the number of interactions available to the tree, and hence to boosting

## 3.3 XGboost: sample code

From the tutorial of xgboost package:

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
bst <- xgboost(data = train$data,
               label = train$label, #the response variable
               max.depth = 2,       #maximum depth of a tree
               eta = 1,             #step size shrinkage used in update to prevents overfitting
               nthread = 2,         #number of parallel threads used to run xgboost
               nround = 2,          #the max number of iterations
               objective = "binary:logistic")
pred <- predict(bst, test$data)
```

# References

[1] Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Ann. Statist.*, 28(2):337–407. With discussion and a rejoinder by the authors.

[2] Mease, D. and Wyner, A. (2008). Evidence contrary to the statistical view of boosting. *J. Mach. Learn. Res.*, 9:131–156.