Bagging and Boosting

Nate Wells

Math 243: Stat Learning

November 11th, 2020

Outline

In today's class, we will. . .

- Discuss bagging and random forests as methods for reducing variance in decision trees
- Investigate boosting as an **learning* method for improving decision trees

Section 1

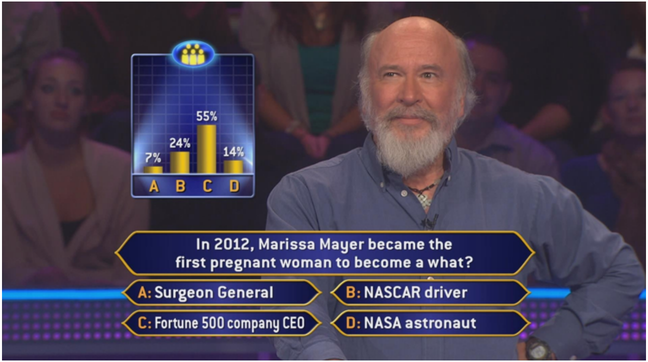
Bagging and Random Forests

Motivation

Can you assemble a collection of weak models and make them strong?

Motivation

Can you assemble a collection of weak models and make them strong?

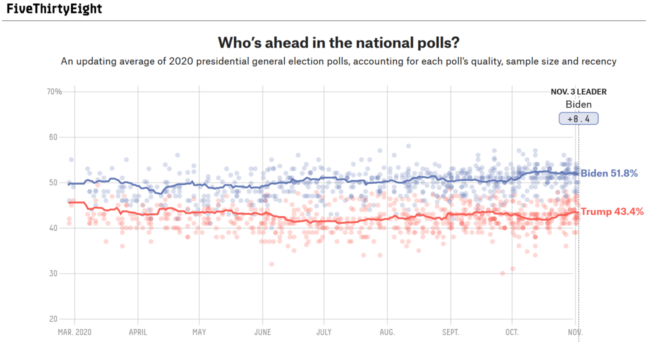


Motivation

Does it always work?

Motivation

Does it always work?



Random Forests

To create a random forest:

1. Select the number of models m to build and a number of predictors k to use at each step t
2. Generate a bootstrap sample for each model
3. Build a tree on the bootstrap sample where at each step, a random selection of k of the p predictors can be used (independent of prior predictors selected)
4. Aggregate the models to create an ensemble model.

Random Forests

To create a random forest:

1. Select the number of models m to build and a number of predictors k to use at each step t
2. Generate a bootstrap sample for each model
3. Build a tree on the bootstrap sample where at each step, a random selection of k of the p predictors can be used (independent of prior predictors selected)
4. Aggregate the models to create an ensemble model.

Advantages of the random forest?

Random Forests

To create a random forest:

1. Select the number of models m to build and a number of predictors k to use at each step t
2. Generate a bootstrap sample for each model
3. Build a tree on the bootstrap sample where at each step, a random selection of k of the p predictors can be used (independent of prior predictors selected)
4. Aggregate the models to create an ensemble model.

Advantages of the random forest?

- Individual models are less correlated, so ensemble has lower variance
- Each tree is quicker to build (why?)

Random Forests

To create a random forest:

1. Select the number of models m to build and a number of predictors k to use at each step t
2. Generate a bootstrap sample for each model
3. Build a tree on the bootstrap sample where at each step, a random selection of k of the p predictors can be used (independent of prior predictors selected)
4. Aggregate the models to create an ensemble model.

Advantages of the random forest?

- Individual models are less correlated, so ensemble has lower variance
- Each tree is quicker to build (why?)

Disadvantages?

Random Forests

To create a random forest:

1. Select the number of models m to build and a number of predictors k to use at each step t
2. Generate a bootstrap sample for each model
3. Build a tree on the bootstrap sample where at each step, a random selection of k of the p predictors can be used (independent of prior predictors selected)
4. Aggregate the models to create an ensemble model.

Advantages of the random forest?

- Individual models are less correlated, so ensemble has lower variance
- Each tree is quicker to build (why?)

Disadvantages?

- Difficult to interpret
- Theoretically properties less well-studied

Hand-drawn Example

Random Forests in R

To create both bagged trees and random forests, we use the randomForest function in the randomForest package in R:

```
library(randomForest)
rfmodel <- randomForest(Pollution_Removal_oz ~ ., data = my_trees_na)
rfmodel
```

```
##
## Call:
##  randomForest(formula = Pollution_Removal_oz ~ ., data = my_trees_na)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 128.5166
##                     % Var explained: 46.63
```

Modifications

We can control how many trees are generated with `ntrees =` and the number of predictors at each split with `mtry=`

Modifications

We can control how many trees are generated with `ntrees =` and the number of predictors at each split with `mtry=`

- By default, `randomForest` uses $p/3$ predictors for regression and $\sqrt{p}$ predictors for classification

Modifications

We can control how many trees are generated with `ntrees =` and the number of predictors at each split with `mtry=`

- By default, `randomForest` uses $p/3$ predictors for regression and $\sqrt{p}$ predictors for classification

```
set.seed(1)
rfmodel2 <- randomForest(Pollution_Removal_oz ~ ., data = my_trees_na,
                         ntrees = 10, mtry = 3)
rfmodel2
```

```
##
## Call:
##  randomForest(formula = Pollution_Removal_oz ~ ., data = my_trees_na,      ntrees
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 147.265
##                     % Var explained: 38.85
```

Modifications

We can control how many trees are generated with `ntrees =` and the number of predictors at each split with `mtry=`

- By default, `randomForest` uses $p/3$ predictors for regression and $\sqrt{p}$ predictors for classification

```
set.seed(1)
rfmodel2 <- randomForest(Pollution_Removal_oz ~ ., data = my_trees_na,
                         ntrees = 10, mtry = 3)
rfmodel2
```

```
##
## Call:
##  randomForest(formula = Pollution_Removal_oz ~ ., data = my_trees_na,      ntrees
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 147.265
##                     % Var explained: 38.85
```

How can we create a bagged model using the `randomForest` function?

Modifications

We can control how many trees are generated with `ntrees =` and the number of predictors at each split with `mtry=`

- By default, `randomForest` uses $p/3$ predictors for regression and $\sqrt{p}$ predictors for classification

```
set.seed(1)
rfmodel2 <- randomForest(Pollution_Removal_oz ~ ., data = my_trees_na,
                         ntrees = 10, mtry = 3)
rfmodel2
```

```
##
## Call:
##  randomForest(formula = Pollution_Removal_oz ~ ., data = my_trees_na,      ntrees
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 147.265
##                     % Var explained: 38.85
```

How can we create a bagged model using the `randomForest` function?

- Set `mtry= p`, where `p` is the total number predictors available

Making predictions

So you have your `randomForest` model. How do you make predictions?

```
my_preds<- predict(rfmodel, test_trees)

data.frame(my_preds,actual = test_trees$Pollution_Removal_oz) %>% head()
```

```
##    my_preds actual
## 1 14.141807   16.6
## 2 26.829172   14.7
## 3  5.344025    0.2
## 4 16.795818   15.0
## 5 25.090853   41.4
## 6 16.105992   10.5
```

Variable Importance

Bagging and Random Forests increase prediction accuracy by reducing variance of the model.

Variable Importance

Bagging and Random Forests increase prediction accuracy by reducing variance of the model.

- But the cost comes in interpretability. We no longer have a single decision tree to follow to reach our prediction.

Variable Importance

Bagging and Random Forests increase prediction accuracy by reducing variance of the model.

- But the cost comes in interpretability. We no longer have a single decision tree to follow to reach our prediction.
- How can we determine which predictors are most influential?

Variable Importance

Bagging and Random Forests increase prediction accuracy by reducing variance of the model.

- But the cost comes in interpretability. We no longer have a single decision tree to follow to reach our prediction.
- How can we determine which predictors are most influential?

One possibility is to record the total amount of RSS/Purity that is decreased due to splits of the given predictor, averaged across all trees in the random forest.

Importance in R

```r
importance(rfmodel)
```

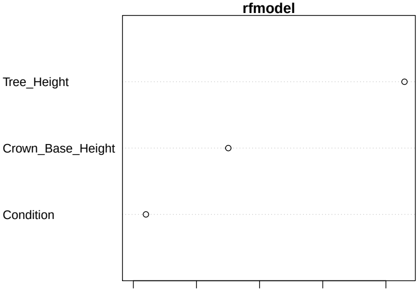
```
##                   IncNodePurity
## Tree_Height           85886.830
## Crown_Base_Height     30087.052
## Condition              3994.087
```

Importance in R

```r
importance(rfmodel)
```

```
##                   IncNodePurity
## Tree_Height           85886.830
## Crown_Base_Height     30087.052
## Condition              3994.087
```

```r
par(mfcol = c(1, 1), mar = c(1, 1, 1, 1))
varImpPlot(rfmodel)
```



Section 2

Boosting

Motivation

Suppose you have a model which, given a binary classification dataset, always returned a classifier with training error strictly lower than 50%.

Motivation

Suppose you have a model which, given a binary classification dataset, always returned a classifier with training error strictly lower than 50%.

- Can one use it to build a strong classifier that has error close to 0?

Motivation

Suppose you have a model which, given a binary classification dataset, always returned a classifier with training error strictly lower than 50%.

- Can one use it to build a strong classifier that has error close to 0?

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.
    - Observations that are incorrectly classifed in the kth iteration recieve more weight in the $(k+1)$th iteration.

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.
  - Observations that are incorrectly classifed in the kth iteration recieve more weight in the $(k+1)$th iteration.
- The overall sequence of classifiers are combined into an ensemble which as high chance of classifying more accurately than any individaul model in the list.

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

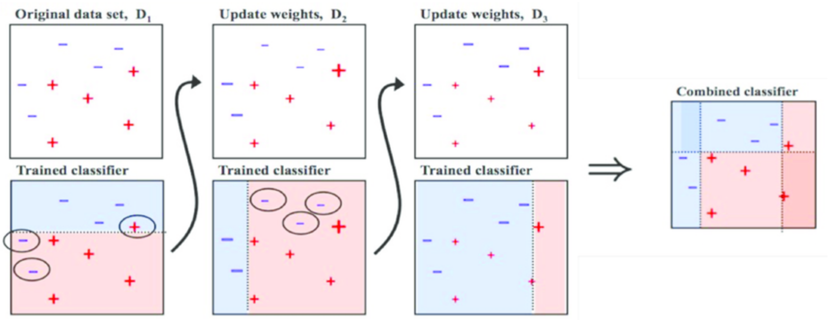
- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.
  - Observations that are incorrectly classifed in the kth iteration recieve more weight in the $(k+1)$th iteration.
- The overall sequence of classifiers are combined into an ensemble which as high chance of classifying more accurately than any individaul model in the list.
- The algorithm relies on using a sequence of **weak** learners (low variance, high bias)

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.
  - Observations that are incorrectly classifed in the kth iteration recieve more weight in the $(k+1)$th iteration.
- The overall sequence of classifiers are combined into an ensemble which as high chance of classifying more accurately than any individaul model in the list.
- The algorithm relies on using a sequence of **weak** learners (low variance, high bias)
  - In the tree setting, we can create weak learners by restricting the depth of the tree.

AdaBoost Graphic



Boosting for regression

Boosting also works in the regression setting. The **gradient boosting machine** is a boosting algorithm that works as follows:

1. Select tree depth D and number of iterations K.
2. Compute the average response $\hat{y}$ and use this as the initial predicted value for each observation
3. Compute the residual for each observation.
4. Fit a regression tree of depth D, using the **residuals** as the response.
5. Predict each observation using the regression tree from the previous step.
6. Update the predicted value of each observation by adding the previous iteration's predicted value to the predicted value generated in the previous step.
7. Repeat at total of K times.

Brief Example

Compute the mean:

```
mu <- mean(my_trees_na$Pollution_Removal_oz)
mu
```

```
## [1] 18.09656
```

Brief Example

Compute the mean:

```
mu <- mean(my_trees_na$Pollution_Removal_oz)
mu
```

```
## [1] 18.09656
```

Compute residuals:

```
boost_tree<- my_trees_na %>%
  mutate(residuals1 = Pollution_Removal_oz - mu)
```

Brief Example

Compute the mean:

```
mu <- mean(my_trees_na$Pollution_Removal_oz)
mu
```

```
## [1] 18.09656
```

Compute residuals:

```
boost_tree<- my_trees_na %>%
  mutate(residuals1 = Pollution_Removal_oz - mu)
```

Fit a new tree

```
boost_tree_model<- tree(residuals1 ~ Crown_Base_Height   ,
                        data = boost_tree)
pruned_boost_tree_model<-prune.tree(boost_tree_model, best = 2)
```

Brief Example

Compute the mean:

```
mu <- mean(my_trees_na$Pollution_Removal_oz)
mu
```

```
## [1] 18.09656
```

Compute residuals:

```
boost_tree<- my_trees_na %>%
  mutate(residuals1 = Pollution_Removal_oz - mu)
```

Fit a new tree

```
boost_tree_model<- tree(residuals1 ~ Crown_Base_Height   ,
                        data = boost_tree)
pruned_boost_tree_model<-prune.tree(boost_tree_model, best = 2)
```

Predict

```
predictions2<- predict(pruned_boost_tree_model, data = boost_tree)
```

Brief Example

Compute the mean:

```
mu <- mean(my_trees_na$Pollution_Removal_oz)
mu
```

```
## [1] 18.09656
```

Compute residuals:

```
boost_tree<- my_trees_na %>%
  mutate(residuals1 = Pollution_Removal_oz - mu)
```

Fit a new tree

```
boost_tree_model<- tree(residuals1 ~ Crown_Base_Height   ,
                        data = boost_tree)
pruned_boost_tree_model<-prune.tree(boost_tree_model, best = 2)
```

Predict

```
predictions2<- predict(pruned_boost_tree_model, data = boost_tree)
```

And so on. . .

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Unlike random forests, boosting is susceptible to over-fitting (since it uses a greedy algorithm to maximize gradient at each step).

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Unlike random forests, boosting is susceptible to over-fitting (since it uses a greedy algorithm to maximize gradient at each step).

- To remedy, we introduce a shrinkage penalty (like in Ridge Regression/LASSO)

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Unlike random forests, boosting is susceptible to over-fitting (since it uses a greedy algorithm to maximize gradient at each step).

- To remedy, we introduce a shrinkage penalty (like in Ridge Regression/LASSO)
  - Instead of adding the full value for a sample to the previous iteration's predicted value, only a fraction of the current predicted value is added.

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Unlike random forests, boosting is susceptible to over-fitting (since it uses a greedy algorithm to maximize gradient at each step).

- To remedy, we introduce a shrinkage penalty (like in Ridge Regression/LASSO)
  - Instead of adding the full value for a sample to the previous iteration's predicted value, only a fraction of the current predicted value is added.
  - This fraction is called the *learning rate* λ, with $0 < \lambda < 1$. (Typical values range from 0.001 to 0.01)

Boosting in R

We use the `gbm` function in the `gmb` package to create Boosted Trees

Boosting in R

We use the `gbm` function in the `gmb` package to create Boosted Trees

- For regression problems, we use the argument `distribution = "gaussian"` and for classification problems, we use `distribution = "bernoulli"`

Boosting in R

We use the `gbm` function in the `gmb` package to create Boosted Trees

- For regression problems, we use the argument `distribution = "gaussian"` and for classification problems, we use `distribution = "bernoulli"`
- The argument `n.trees` controls the number of iterations
- The argument `interaction.depth` controls the depth of each tree
- The argument `shrinkage` controlls the learning rate λ

Boosting in R

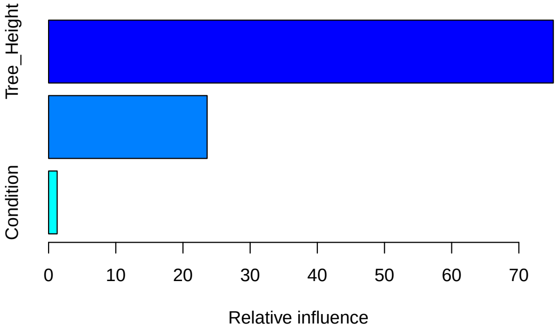
We use the gbm function in the gmb package to create Boosted Trees

- For regression problems, we use the argument `distribution = "gaussian"` and for classification problems, we use `distribution = "bernoulli"`
- The argument `n.trees` controls the number of iterations
- The argument `interaction.depth` controls the depth of each tree
- The argument `shrinkage` controlls the learning rate λ

```
library(gbm)
set.seed(10101)
boosted_tree<-gbm(Pollution_Removal_oz ~., my_trees_na,
                  distribution = "gaussian",
                  n.trees=1000,
                  interaction.depth = 2,
                  shrinkage = 0.02)
```

Summary Information

```
summary(boosted_tree)
```




```
##                                   var   rel.inf
## Tree_Height               Tree_Height 75.130159
## Crown_Base_Height Crown_Base_Height 23.594814
## Condition                   Condition  1.275027
```

Boosted Tree vs. Random Forest

```
my_preds_rf<- predict(rfmodel, test_trees)
my_preds_bt<- predict(boosted_tree, test_trees)
```

Boosted Tree vs. Random Forest

```r
my_preds_rf<- predict(rfmodel, test_trees)
my_preds_bt<- predict(boosted_tree, test_trees)
```

```r
MSE_rf <- mean( (my_preds_rf - test_trees$Pollution_Removal_oz)^2 )
MSE_bt <-  mean( (my_preds_bt - test_trees$Pollution_Removal_oz)^2 )

data.frame( model = c("Random Forest", "Boosted Tree"), MSE = c(MSE_rf, MSE_bt))
```

```
##           model       MSE
## 1 Random Forest 103.82926
## 2  Boosted Tree  99.15018
```