Tree Based Methods: Bagging, Boosting, and Regression Trees

Rebecca C. Steorts, Duke University

STA 325, Chapter 8 ISL

Agenda

- Bagging
- Random Forests
- Boosting

Bagging

Recall that we introduced the boostrap (Chapter 5) which is useful in many situations in which it is hard or even impossible to directly compute the standard deviation of a quantity of interest.

Here, we will see that we can use the boostrap to improve statistical learning through decision trees.

High variance in decision trees

Decision trees such as regression or classification trees suffer from high variance.

This means that if we split the training data into two parts at random and fit a decision tree to both halves, the results can be quite different.

In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct data sets; linear regression tends to have low variance, if the ratio of n to p is moderately large.

Bagging

Bootstrap aggregation or bagging is a general-purpose procedure for reducing the variance of a statistical learning method.

Bagging

Recall that given a set of independent observations

$$Z_1, \ldots, Z_n$$

each with variance σ^2 the variance of $\bar{Z}$ is σ^2/n.

In other words, averaging a set of observations reduces variance.

One natural way to reduce the variance and hence increase the prediction accuracy of a statistical learning method is to take many training sets from the population and build a separate prediction model using each training set. Then we average the resulting predictions.

Bagging

1. Calculate
$$\hat{f}^1(x), \hat{f}^2(x), \ldots, \hat{f}^B(x)$$
using B seperate training sets.
2. Average them in order to obtain a single low-variance statistical learning model:

$$\hat{f}_{\text{ave}}(x)\frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x).$$

Bagging

1. Calculate
$$\hat{f}^1(x), \hat{f}^2(x), \ldots, \hat{f}^B(x)$$
using B seperate training sets.
2. Average them in order to obtain a single low-variance statistical learning model:

$$\hat{f}_{\text{ave}}(x) \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x).$$

This is not practical! Why?

Bagging

1. Calculate
$$\hat{f}^1(x), \hat{f}^2(x), \ldots, \hat{f}^B(x)$$
using B seperate training sets.
2. Average them in order to obtain a single low-variance statistical learning model:

$$\hat{f}_{\text{ave}}(x)\frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x).$$

This is not practical! Why?

We don't usually have access to B training data sets.

Bagging

But we can bootstrap!

We can take repeated samples from the (single) training data set.

We will generate B different bootstrapped training data sets.

We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$.

Finally, we average all the predictions, to obtain

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

This is called bagging.

Bagging for Regression Trees

To apply bagging to regression trees do the following:

1. construct B regression trees using B bootstrapped training sets
2. average the resulting predictions.

Bagging for Regression Trees

Regression trees are grown deep, and are not pruned.

Hence each individual tree has high variance, but low bias. Thus, a veraging these B trees reduces the variance.

Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.

Bagging for Classification Trees

There are many ways to apply bagging for classification trees.

We explain the simplest way.

Bagging for Classification Trees

For a given test observation, we can record the class predicted by each of the B trees, and take a majority vote.

A majority vote is simply the overall prediction is the most commonly occurring class among the B predictions.

Out-of-Bag Error Estimation

It turns out that there is a very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation or the validation set approach.

Out-of-Bag Error Estimation

Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations.

One can show that on average, each bagged tree makes use of around two-thirds of the observations. (See Exercise 2, Chapter 5).

The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.

We can predict the response for the ith observation using each of the trees in which that observation was OOB.

This will yield around $B/3$ predictions for the ith observation.

Out-of-Bag Error Estimation

In order to obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal).

This leads to a single OOB prediction for the ith observation.

An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or classification error (for a classification problem) can be computed.

The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation.

OOB and CV

It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.

The OOB approach for estimating the test error is particularly convenient when performing bagging on large data sets for which cross-validation would be computationally onerous.

Interpretability of Bagged Trees

Recall that bagging typically results in improved accuracy over prediction using a single tree.

Unfortunately, however, it can be difficult to interpret the resulting model.

But one main attraction of decision trees is their intrepretability!

However, when we bag a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure.

Thus, bagging improves prediction accuracy at the expense of interpretability.

Variable Importance Measures

Although the collection of bagged trees is much more difficult to interpret than a single tree, one can obtain an overall summary of the importance of each predictor using

- the RSS (for bagging regression trees) or
- the Gini index (for bagging classification trees).

Variable Importance Measures

In the case of bagging regression trees, we can record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees.

A large value indicates an important predictor.

Similarly, in the context of bagging classification trees, we can add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.

Random Forests

Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees.

Random Forests

As in bagging, we build a number of decision trees on bootstrapped training samples.

But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.

The split is allowed to use only one of those m predictors.

A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$.

Random Forests

In building a random forest, at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors.

What's the rationale behind this?

Random Forests

Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.

Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split.

All of the bagged trees will look quite similar to each other and the predictions from the bagged trees will be highly correlated.

Averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.

In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.

Random forests

Random forests overcome this problem by forcing each split to consider only a subset of the predictors.

Therefore, on average $(p - m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance.

We can think of this process as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable.

Random Forest versus Bagging

What's the difference between random forests and bagging.

The choice of the predictor subset size m.

If a random forest is built where $m = p$, then bagging is a special case of random forests.

Boosting

Boosting is another approach for improving the predictions resulting from a decision tree.

Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification trees.

Bagging

Recall that bagging works via the following procedure:

1. Create multiple copies of the original training data set using the bootstrap.
2. Fit a separate decision tree to each bootstraped copy.
3. Then combine all of the decision trees in order to create a single predictive model.

Each tree is built on a bootstrap data set, which is independent of the other trees.

Boosting

Boosting works in a similar way, except that the trees are grown sequentially:

1. Each tree is grown using information from previously grown trees.
2. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.

Let's see how this works in the context of regression trees.

Boosting for Regresion Trees

What is the idea behind this procedure?

Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.

In general, statistical learning approaches that learn slowly tend to perform well. Note that in boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown.

Boosting for Regresion Trees

Given the current model, we fit a decision tree to the residuals from the model.

That is, we fit a tree using the current residuals r rather than the outcome Y, as the response.

We then add this new decision tree into the fitted function in order to update the residuals.

Boosting for Regresion Trees

Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.

By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well.

The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Boosting Algorithm for Regression Trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training data set.
2. For $b = 1, 2, \ldots, B$, repeat:

1. Fit a tree $\hat{f}^b$ with d splits ($(d + 1)$ terminal nodes) to the training data (X, r)
2. Update $\hat{f}(x)$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{1}$$

3. Update the residuals

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i) \tag{2}$$

3. Output the boosted model

$$\hat{f}(x) = \sum b = 1^B \lambda \hat{f}^b(x) \tag{3}$$

Tuning parameters

There are three tuning parameters that we must carefully consider.

1. The number of trees B.
2. The shrinkage parameter λ.
3. The number d splits in each tree.

Tuning parameters

1. The number of trees B.

Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all.

We use cross-validation to select B.

Tuning parameters

2. The shrinkage parameter λ. (This is a small positive number).

This controls the rate at which boosting learns.

Typical values are 0.01 or 0.001, and the right choice can depend on the application.

Very small λ can require using a very large value of B in order to achieve good performance.

Tuning parameters

3. The number d splits in each tree.

This controls the complexity of the boosted ensemble.

Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split.

In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable.

More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Boosting for Classification Trees

The details of boosting for classification trees are omitted from ISLR, but they are in ESL if you wish to go over them on your own.

Bagging and Random Forests on Boston data set

Here we apply bagging and random forests to the Boston data, using the randomForest package in R.

The exact results obtained in this section may depend on the version of R and the version of the randomForest package installed on your computer.

Recall that bagging is just a special case of random forests with $m = p$.

Task 1

Setup a training and test set for the Boston data set.

Solution to Task 1

```r
library(MASS)
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
set.seed (1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
boston.test=Boston[-train ,"medv"]
```

Task 2

Apply bagging using the randomForest package in R. What does the argument mtry do?

Solution to Task 2

```
bag.boston=randomForest(medv~.,data=Boston,subset=train, mt
```

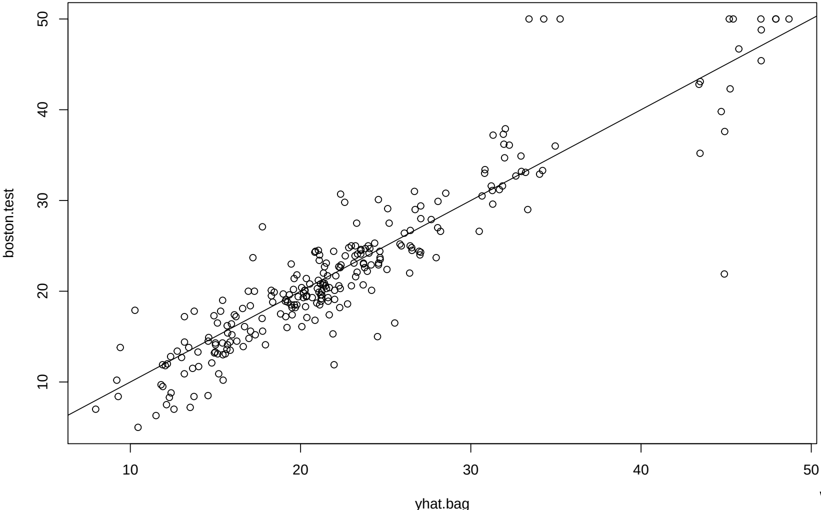
The argument mtry = 13 means that we should use all 13 predictors for each split of the tree, hence, do bagging.

Task 3

How well does the bagged model perform on the test set?

Solution to Task 3

```
yhat.bag = predict(bag.boston,newdata=Boston[-train,])
plot(yhat.bag, boston.test)
abline(0,1)
```



What

Solution to Task 3

```r
mean((yhat.bag-boston.test)^2)
```

```
## [1] 13.33831
```

The test set MSE associated with the bagged regression tree is 13.16, almost half that obtained using an optimally-pruned single tree (investigate this on your own).

Task 4

Change the number of trees grown by randomForest() using the ntree argument. For example, what happens to the MSE when we grow the tree from 13 to 25?

Moving to random forests

Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the mtry argument.

By default, randomForest() uses $p/3$ variables when building a random forest of regression trees, and $\sqrt{p}$ variables when building a random forest of classification trees.

Task 5

Build a random forest on the same data set using mtry $= 6$. Comment on the difference from the test MSE from using the random forest compared to bagging.

Solution to Task 5

```r
set.seed(1)
rf.boston=randomForest(medv~.,data=Boston,subset=train, mtr
yhat.rf = predict(rf.boston ,newdata=Boston[-train ,])
mean((yhat.rf-boston.test)^2)
```

```
## [1] 11.48022
```

We see that the test MSE for a random forest is 11.48; this indicates that random forests yielded an improvement over bagging in this case (versus 13.34).

Task 6

Using the importance() function, comment on the importance of each variable.

Solution to Task 6

```r
importance(rf.boston)
```

```
##            %IncMSE IncNodePurity
## crim     12.547772    1094.65382
## zn        1.375489      64.40060
## indus     9.304258    1086.09103
## chas      2.518766      76.36804
## nox      12.835614    1008.73703
## rm       31.646147    6705.02638
## age       9.970243     575.13702
## dis      12.774430    1351.01978
## rad       3.911852      93.78200
## tax       7.624043     453.19472
## ptratio  12.008194     919.06760
## black     7.376024     358.96935
## lstat    27.666896    6927.98475
```

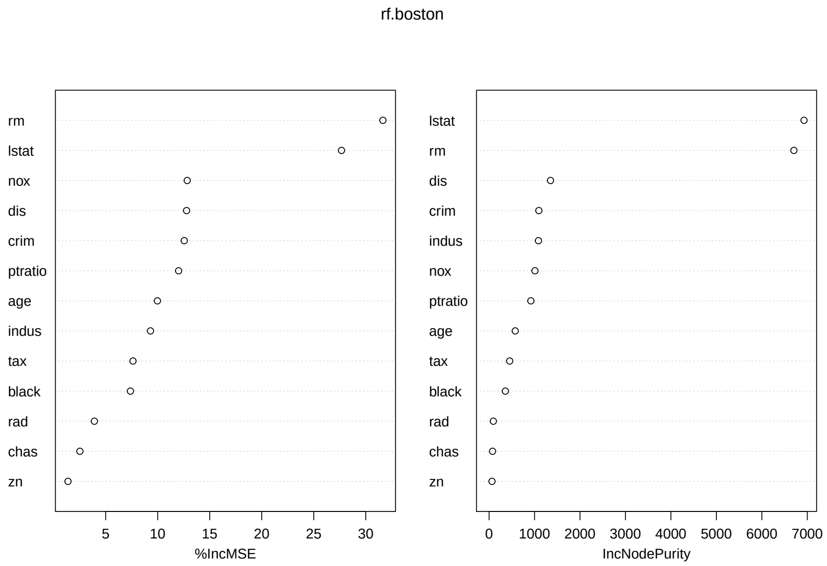
Solution to Task 6

Two measures of variable importance are reported. The former is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model.

The latter is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees.

Solution to Task 6

```r
varImpPlot (rf.boston)
```



The results indicate that across all of the trees considered in the random forest, the wealth level of the community (lstat) and the house size (rm) are by far the two most important variables.

Boosting on the Boston data set

We use the gbmpackage, and within it the gbm() function, to fit boosted regression trees to the Boston data set.

We run gbm() with the option distribution=“gaussian” since this is a regression problem.

If it were a binary classification problem, we would use distribution=“bernoulli”.

The argument n.trees=5000 indicates that we want 5000 trees, and the option interaction.depth=4 limits the depth of each tree.

The summary() function produces a relative influence plot and also outputs the relative influence statistics.

Task 1

Perform boosting on the training data set, treating this as a regression problem.

Solution to Task 1

```r
library(gbm)
```

```
## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

Solution to Task 1

```
set.seed (1)
boost.boston=gbm(medv~.,data=Boston[train,],distribution="g
```

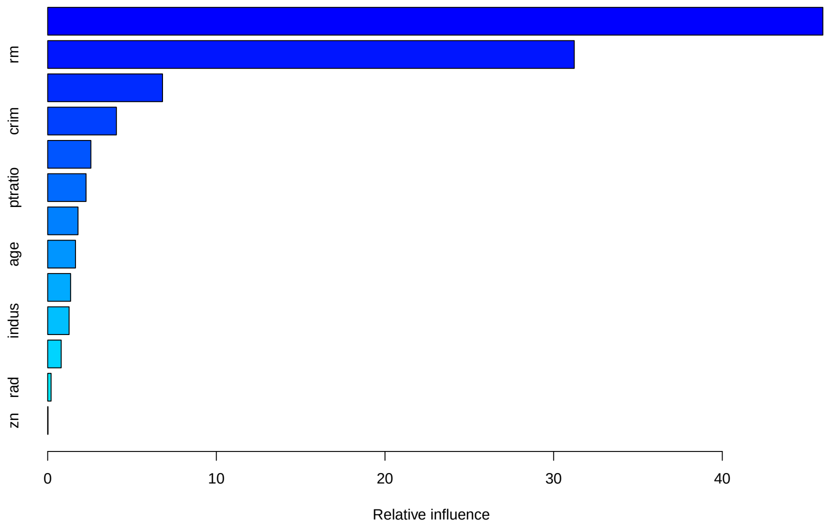
"interaction.depth" refers to the maximum depth of variable interactions. 1 implies an additive model, 2 implies a model with up to 2-way interactions, etc.

Task 2

What does summary of the boosted regression tree tell us?

Solution to Task 2

```r
summary(boost.boston)
```



```
##                       var    rel.inf
```

Solution to Task 2

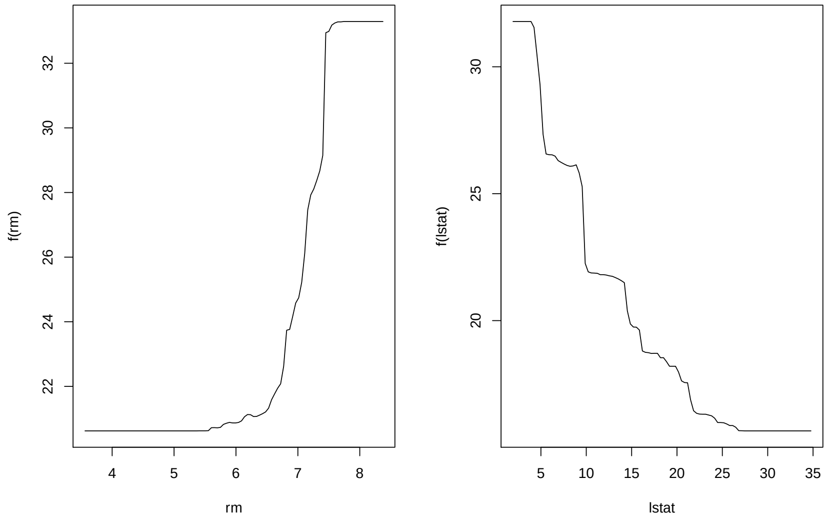
We see that lstat and rm are by far the most important variables.

Task 3

Produce partial dependence plots for these two variables (lstat and rm).

Solution to Task 3

```
par(mfrow=c(1,2))
plot(boost.boston ,i="rm")
plot(boost.boston ,i="lstat")
```



These plots illustrate the marginal effect of the selected variables on the response after integrating out the other variables. In this case, as we might expect, median house prices are increasing with rm and decreasing with lstat.

Task 4

Now use the boosted model to predict medv on the test set. Report the MSE.

Solution to Task 4

```
yhat.boost=predict(boost.boston,newdata=Boston[-train,], n.trees
mean((yhat.boost -boston.test)^2)
```

```
## [1] 11.84434
```

The test MSE obtained is 11.8; similar to the test MSE for random forests and superior to that for bagging.

Task 5

What happens if we vary the shrinkage parameter from its default of 0.001 to 0.02? Report the test MSE.

Solution to Task 5

```
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussi
yhat.boost=predict(boost.boston,newdata=Boston[-train,], n.trees
mean((yhat.boost -boston.test)^2)
```

```
## [1] 11.51109
```

In this case, using $\lambda = 0.2$ leads to a slightly lower test MSE than $\lambda = 0.001$.