

DS 740

Data Mining

Bagging and Random Forests



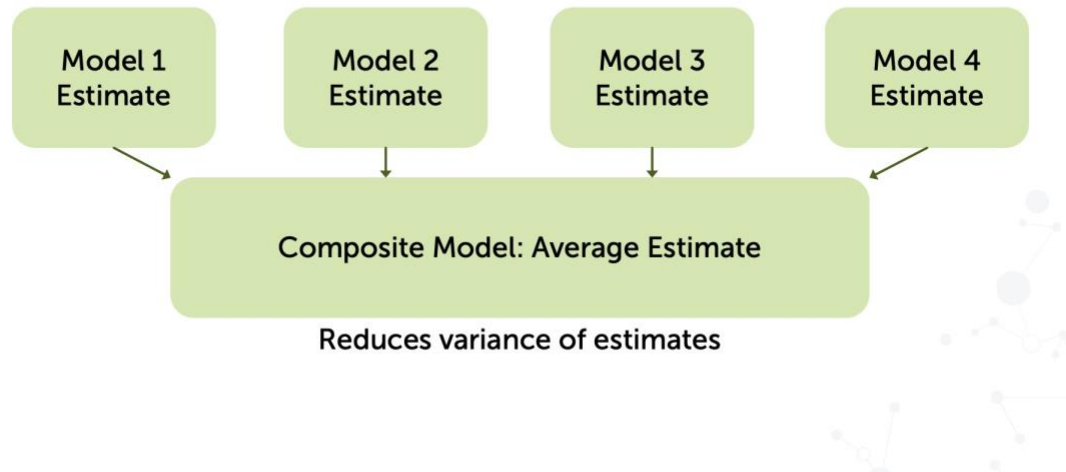
Learning Objectives

- By the end of this lesson, you will be able to:
- Explain how bagging works.
- Explain how random forests work.
- Choose a number of trees for bagging or random forests.
- Interpret a bagged or random forest model.



Bagging Overview

Bootstrap **ag**gregation



Bagging stands for bootstrap aggregation. The idea is that by taking the average of estimates from many models, you can get a composite model with lower variance than any of the original models. Bagging can be applied to many different data mining algorithms, but it's especially effective when applied in the context of decision trees.

Bagging Process

1. Sample from data.
2. Build a tree.
3. Estimate response values for out-of-bag data points.
4. For each data point, average the estimates.

Do this
multiple
times

```
library(randomForest)

sonar_bag = randomForest(factor(V61) ~ ., data = sonar,
                          mtry = 60, importance = TRUE)
# could use subset = in_train
# or data = sonar[in_train, ]
sonar_bag
```

To perform bagging, we take a bootstrap sample from the data and build a tree to fit that bootstrap sample, then we use the tree to estimate the response values for the out of bag data points, in other words, the data points that were not included in that bootstrap sample. We repeat this process multiple times. The final estimate for the response value for any given data point will be the average of all of the estimates found when that data point was not in the bootstrap sample.

Notes:

```
library(randomForest)
library(randomForest)
sonar_bag = randomForest(factor(X61) ~ ., data = sonar,
                          mtry = 60, importance = TRUE)
# Could use subset = in_train
# or data = sonar[in_train, ]
sonar_bag
```

Bagging in R

```
sonar_bag = randomForest(factor(V61) ~ ., data = sonar,  
                          mtry = 60, importance = TRUE)  
# Could use subset = in_train  
# or data = sonar[in_train, ]  
sonar_bag
```

number of predictor variables

classification problem: confusion matrix

regression problem: percentage of variation explained by model

To perform bagging in R, we can use the random forest function, which is in the random forest package. In the arguments for this function, `mtry` represents the number of predictor variables we want to try or consider at each split of a tree. For classic bagging, this is always equal to the total number of predictor variables in the data set.

Importance equals true tells R to compute and importance measure for each of the predictor variables. This can be useful for interpreting the model. If we wanted, we could also include the argument `subset equals n train` to restrict our analysis to just a training data set, or as usual, we could use `data equals sonar square bracket n train` to specify the rows of the training set when we specify the data set we want to use.

Finally, typing the name of the object we created, in this case, `sonar bag`, will show us the confusion matrix for our classification problem. Or in the case of a regression problem, it would show us the percentage of variation explained by the model.

Notes:

```
library(randomForest)  
library(randomForest)  
sonar_bag = randomForest(factor(X61) ~ ., data = sonar,  
                          mtry = 60, importance = TRUE)  
# Could use subset = in_train
```

```
# or data = sonar[in_train, ]  
sonar_bag
```

Averaging Estimates for Classification

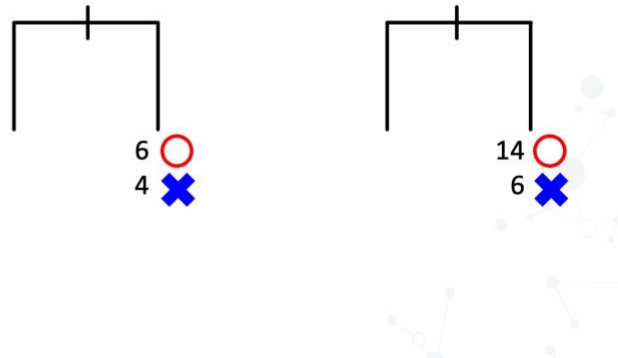
Two options:

- Majority vote
- Average the probability of class membership

Example

Majority vote: ○

Estimated probability of ○ : 0.65



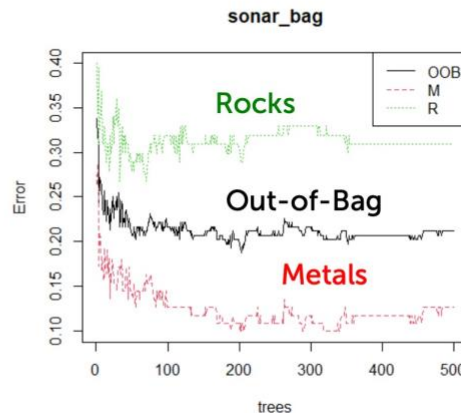
For regression problems, we can simply take the mean response value. But for classification problems, we have two possible choices. We could simply take the majority vote from all of the terminal nodes that would contain that data point or we can take the average of the probabilities of the class memberships.

For example, suppose we have two trees. And a particular data point would end up in the right-hand node of each of these trees. If we took a majority vote, we would end up with 2 votes out of 2 in favor of the red circle class. Instead if we took the estimated probability of being in the red circle class, we would be averaging 6 out of 10 and 14 out of 20 to get an average of 0.65. Averaging the probability tends to reduce the variance of the estimates and gives a slightly more nuanced understanding.

How Many Trees to Use?

- Not a parameter that risks over-fitting.
- Enough so that error rate levels out.

```
plot(sonar_bag)
legend("topright",
      colnames(sonar_bag$err.rate),
      col = 1:3, lty = 1:3)
```



So how many times should we repeat the process of bootstrap sampling and building a tree? This isn't a parameter that risks overfitting to the data. So you don't need to use cross validation to choose a good value. You only need to choose enough trees so that the error rate levels out. The error rates for the sonar example using the default value for the number of trees, 500, are shown here.

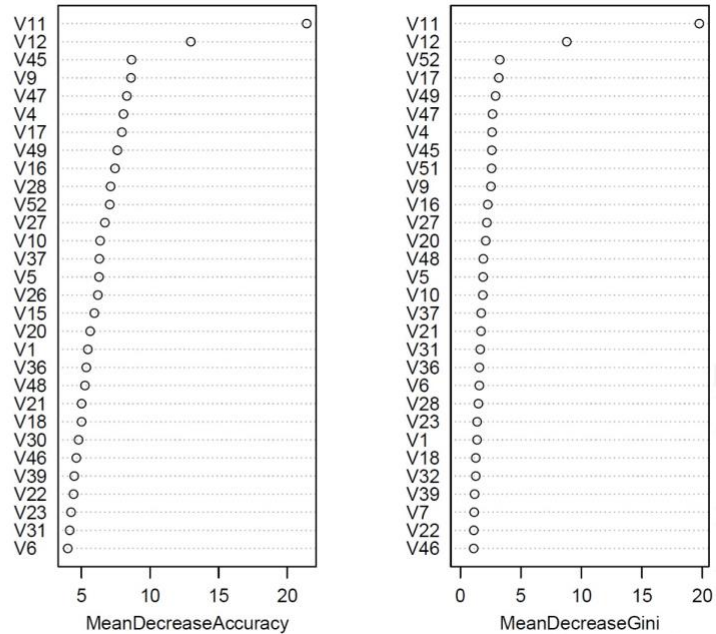
All three of these curves show the out of bag error rates, meaning the error rate computed on data points that were left out of the bootstrap sample that was used to build each particular tree. This gives a more honest assessment of the error, similar to if we were doing cross validation. Here you can see that the error rate on predicting rocks is higher than the error rate for predicting metals. But the overall average out of bag error rate levels out after about 100 trees. So our default value of 500 trees is just fine.

Notes:

```
plot(sonar_bag)
legend("topright",
      colnames(sonar_bag$err.rate),
      col = 1:3, lty = 1:3)
```


Variable Importance

```
importance(sonar_bag)
varImpPlot(sonar_bag)
```



With bagging, we don't get a single decision tree that we can make a diagram of to interpret. So instead, we look at the importance of each variable as measured by its contribution to accuracy and its contribution to node purity as measured by the Gini index.

Notes:

```
importance(sonar_bag)
varImpPlot(sonar_bag)
```

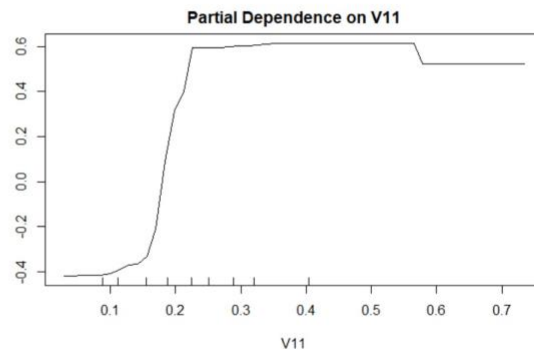
If the variable importance plot is hard to read in Rstudio, try printing it to a pdf:

```
pdf("Sonar_bag.pdf")
varImpPlot(sonar_bag)
dev.off()
```

Partial Dependence Plot Gives the Marginal Effect of One Variable

- Set each data point's value of V11 equal to .03
- Make predictions
- Take the mean prediction
 - $\text{mean} \left(\log \frac{P(\text{Metal})}{P(\text{Rock})} \right)$ for classification problems
 - Averages across possible values of the other variables
- Repeat for V11 = .04, .05, ...

```
partialPlot(sonar_bag, pred.data = sonar,  
            x.var = V11, which.class = "M")
```



Another useful tool for interpreting bagged or random forest models is a partial dependence plot. This shows the effects of one predictor variable on the predicted response after averaging across all the other predictor variables. The way this works is we choose one predictor variable, let's say v11, which was the most important predictor in the sonar bag model, and we take our existing data set and use it to create a hypothetical or simulated data set in which we set the value of v11 for each of the data points, equal to its minimum. In this data set, that's about 0.03.

Then we make predictions for these new hypothetical data points that have v11 equal to 0.03 and their original values of all of the other predictors. And we take the mean prediction across all of the data points in the data set. For classification problems, this is taking the mean of the log odds, or in other words, the mean of the log ratio of probabilities of being metal to being rock.

So what this does is it averages across the possible values of all the other variables to get a single prediction for what happens when v11 equals 0.03. So this gets us 1 point in our graph, the point where v11 is equal to 0.03. We would then repeat this process for v11 equal to 0.04, 0.05, and so on, up to the maximum value of v11.

So what we can see here is that v11 is positively associated with the predicted response. So in this case, it's positively associated with the

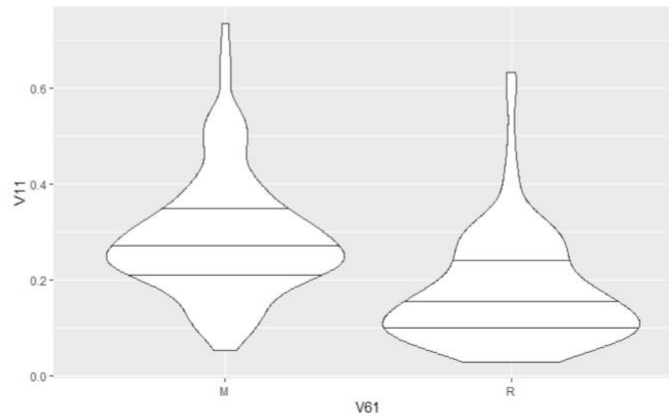
predicted probability of being metal, and we see that this transition happens fairly early in the range of possible values of v11. So data points with v11 less than about 0.1 are highly likely to be rock. Then there's a steep transition, and data points with a value of v11 greater than about 0.2 are highly likely to be metal.

Notes:

```
partialPlot(sonar_bag, pred.data = sonar,  
            x.var = V11, which.class = "M")
```

Metals Typically Have a Higher Value of V11

```
gf_violin(V11 ~ V61, data = sonar,  
          draw_quantiles = c(.25, .5, .75))
```



In our partial dependence plot, we saw that higher values of v11 were associated with higher predicted probabilities of being metal. Another useful tool for interpretation is to relate this back to the observed value of the response variable. In this case, our response variable is categorical, and our predictor variable, v11, is quantitative. So some good graphs to look at their relationship include a box plot, a histogram with facets, or a violin plot. Here, we can see in the violin plot that metals do indeed typically have higher values of v11 than rocks.

Notes:

```
gf_violin(V11 ~ V61, data = sonar,  
          draw_quantiles = c(.25, .5, .75))
```

Joint Effect of V11 and V12

Hold all other variables constant

```
example_data <- sonar %>%  
  mutate(v1 = median(v1),  
         v2 = median(v2),  
         v3 = median(v3),  
         ...  
         v60 = median(v60))
```

```
example_data <- sonar %>%  
  mutate(across(c(-v11, -v12, -v61), median))
```

Let's look at the joint effect of v11 and v12 on the predicted response by holding all the other variables constant at their medians. This is sort of the opposite of the partial dependence plot because we're keeping v11 and v12 equal to their values in the data and holding all the other variables constant. We could do this using the mutate function from the dplyr package and simply set each of the variables 1 through 60, except for 11 and 12, equal to their medians.

But this would take a long time to type. A faster way to do this is to use mutate, but inside that, to use across function to specify that we want to apply the median function to all of the variables across the data set except for v11, v12, and v61. v61 is the response variable, which we're not really going to use here. And applying the median to it would create an error because it was a categorical variable.

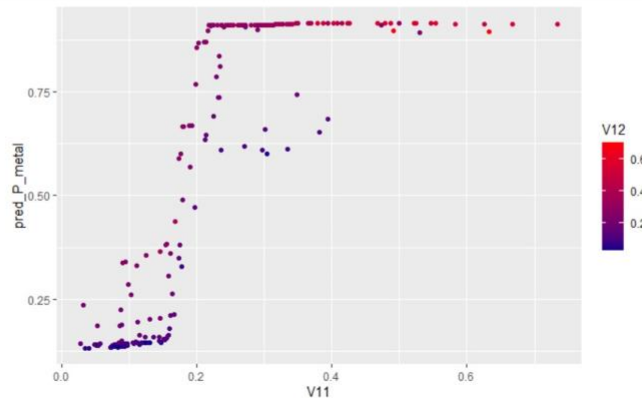
Notes:

```
example_data <- sonar %>%  
  mutate(V1 = median(V1),  
         V2 = median(V2),  
         V3 = median(V3),  
         ...  
         V60 = median(V60))
```

```
example_data <- sonar %>%  
  mutate(across(c(-V11, -V12, -V61), median))
```

Viewing Predictions

```
example_data <- example_data %>%  
  mutate(pred_P_metal = predict(sonar_bag, example_data, type = "prob")[,1])  
  
example_data %>%  
  gf_point(pred_P_metal ~ V11, color =~ V12) %>%  
  gf_refine(scale_color_gradient(low = "darkblue", high = "red"))
```



By graphing the predicted probability of being metal for all the data points in our hypothetical example data set, we see a similar result to the partial dependence plot that v11 is positively associated with the predicted probability of being metal and that the probability rises steeply in a short range around v11 equal to 0.1 to 0.2. We also see that v12 is positively associated with the predicted probability of being metal because there are more bright red points on the top of the graph than on the bottom.


Additionally, v11 and v12 appear to be positively associated with each other because there are more red points on the right hand side of the graph where v11 is large and more dark blue points showing v12 being small on the left hand side of the graph.

Notes:

```
example_data <- example_data %>%  
  mutate(pred_P_metal = predict(sonar_bag, example_data, type =  
    "prob")[,1])  
  
example_data %>%  
  gf_point(pred_P_metal ~ V11, color =~ V12) %>%  
  gf_refine(scale_color_gradient(low = "darkblue", high = "red"))
```

Random Forests

Problem with Bagging:

Average of highly correlated quantities  high variance.

Solution: Random forests.

- Modification of bagging.
- De-correlate trees.
- At each split, only consider a subset of predictors.

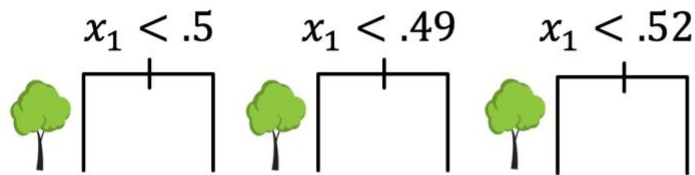


The goal of bagging is to reduce the variance of estimates by taking averages. But the average of a bunch of highly correlated quantities will tend to have higher variance than the average of a bunch of quantities with low correlations.

Random forests are a modification of bagging that reduces the correlation between the trees we generate and, therefore, between the estimates we get out of them. The way random forests work is that at each split that we generate in a tree, we only consider a randomly chosen subset of the possible predictor variables.

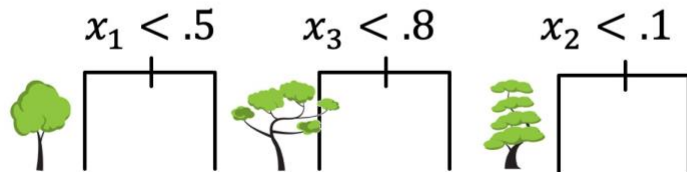
Bagging vs. Random Forests

Bagging:



Similar trees,
highly correlated
predictions

Random forests:



Differentiated
trees, lower
prediction
correlation

For example, suppose that one of our predictor variables, say x_1 , is highly informative about the response, while the other predictor variables only give us a moderate amount of information about the response. In that situation, it's likely that all of the trees generated through bagging will have their first split based on the variable x_1 , and there will only be small differences in the threshold value for that split based on which data points were randomly chosen to be in the bag for that tree. That means that all of the bagged trees will be very similar, so their predictions will be highly correlated.

In a random forest, the variable x_1 simply wouldn't be an option at some of the splits. So some of the trees would have to have their first split based on some other variable. This leads to greater differences between the trees and lower correlation between their predictions.

When to Use Random Forests

Most useful when

- there are strong correlations between predictors

or

- one or a handful of predictors are much more informative about the response variable than the others



How Many Predictor Variables to Use

Test with cross-validation.

Good starting points:

- \sqrt{p} for classification
- $p/3$ for regression

p = number of predictor variables

The number of predictor variables we sample at each of the nodes is a model parameter that can be selected using cross validation. Some good starting points are the square root of p for classification problems or $p/3$ for regression problems, where p is the number of predictor variables.

Random Forests in R

Same as bagging, but with `mtry` equal to the desired number of predictor variables to consider at each node.

```
sonar_rf = randomForest(factor(V61) ~ ., data = sonar,  
                        mtry = 8, importance = TRUE)
```

Notes:

```
sonar_rf = randomForest(factor(V61) ~ ., data = sonar,  
                        mtry = 8, importance = TRUE)
```

Random Forests and bagging in caret

```
set.seed(88)
data_used = sonar

ctrl = trainControl(method = "cv", number = 5)
sonar_caret = train(V61 ~ .,
  data = data_used,
  method = "rf",
  tuneGrid = expand.grid(mtry = c(6, 7, 8, 9, 10, 60)),
  trControl = ctrl)

sonar_caret
```

The R Markdown document associated with this lesson also includes an example of how to implement a random forest or bagged model in Caret. Depending on what data set you choose, this might be useful to you on the project.

Notes:

```
set.seed(88)
data_used = sonar

ctrl = trainControl(method = "cv", number = 5)
sonar_caret = train(V61 ~ .,
  data = data_used,
  method = "rf",
  tuneGrid = expand.grid(mtry = c(6, 7, 8, 9, 10, 60)),
  trControl = ctrl)

sonar_caret
```

Summary

- Bagging reduces variance by averaging estimates from multiple models.
- Each model is fit to a bootstrap sample of the original data.
- Random forests follow the same strategy as bagging, but each split is chosen from a random subset of predictors.
- Variable importance and partial dependence plots are useful for interpreting the model from bagging or random forests.

