Tree-based methods

December 29, 2024

Table of contents

Tree-based methods: A high level overview

- *Tree-based* methods can be used for regression and classification problems.
- They involve splitting up the predictor space into a number of simple regions. This is called *stratifying* or *segmenting*.
- The set of rules used for segmenting can be represented as a tree, which is why these techniques are called *decision-tree* methods.

Tree-based methods: A high level overview

- *Tree-based* methods can be used for regression and classification problems.
- They involve splitting up the predictor space into a number of simple regions. This is called *stratifying* or *segmenting*.
- The set of rules used for segmenting can be represented as a tree, which is why these techniques are called *decision-tree* methods.

Tree-based methods: A high level overview

- *Tree-based* methods can be used for regression and classification problems.
- They involve splitting up the predictor space into a number of simple regions. This is called *stratifying* or *segmenting*.
- The set of rules used for segmenting can be represented as a tree, which is why these techniques are called *decision-tree* methods.

Advantages and disadvantages of *tree-based methods*

+ *Tree-based methods* are simple to interpret usefully.

– They typically cannot compete with the best *supervised learning* approaches in terms of prediction accuracy.

- *Bagging*, *random forests*, and *boosting* are methods that grow multiple trees which are then combined to yield a single consensus prediction.
- Combining multiple trees often results in dramatic improvements in prediction accuracy, at the expense of being somewhat less interpretable.

Advantages and disadvantages of *tree-based methods*

+ *Tree-based methods* are simple to interpret usefully.
– They typically cannot compete with the best *supervised learning* approaches in terms of prediction accuracy.

- *Bagging*, *random forests*, and *boosting* are methods that grow multiple trees which are then combined to yield a single consensus prediction.
- Combining multiple trees often results in dramatic improvements in prediction accuracy, at the expense of being somewhat less interpretable.

Advantages and disadvantages of *tree-based methods*

+ *Tree-based methods* are simple to interpret usefully.
– They typically cannot compete with the best *supervised learning* approaches in terms of prediction accuracy.

- *Bagging*, *random forests*, and *boosting* are methods that grow multiple trees which are then combined to yield a single consensus prediction.
- Combining multiple trees often results in dramatic improvements in prediction accuracy, at the expense of being somewhat less interpretable.

Advantages and disadvantages of *tree-based methods*

+ *Tree-based methods* are simple to interpret usefully.
− They typically cannot compete with the best *supervised learning* approaches in terms of prediction accuracy.

- *Bagging*, *random forests*, and *boosting* are methods that grow multiple trees which are then combined to yield a single consensus prediction.
- Combining multiple trees often results in dramatic improvements in prediction accuracy, at the expense of being somewhat less interpretable.

The basics of decision trees

The basics of decision trees

Decision trees can be applied to both *regression* and *classification* problems.

We first consider regression problems and then move on to classification.

The basics of decision trees

Decision trees can be applied to both *regression* and *classification* problems.

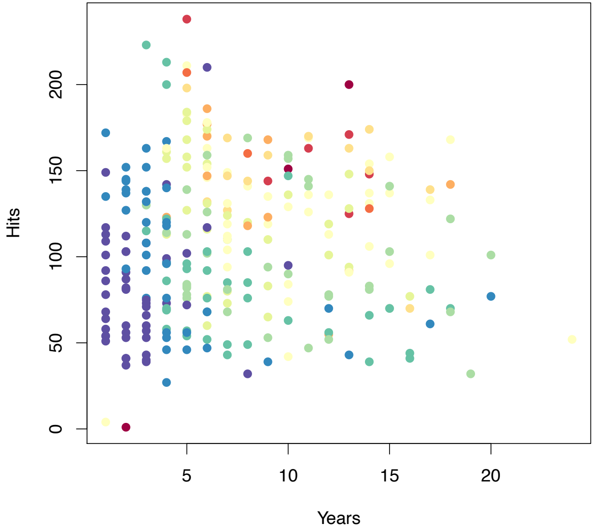
We first consider regression problems and then move on to classification.

The basics of decision trees

Regression trees

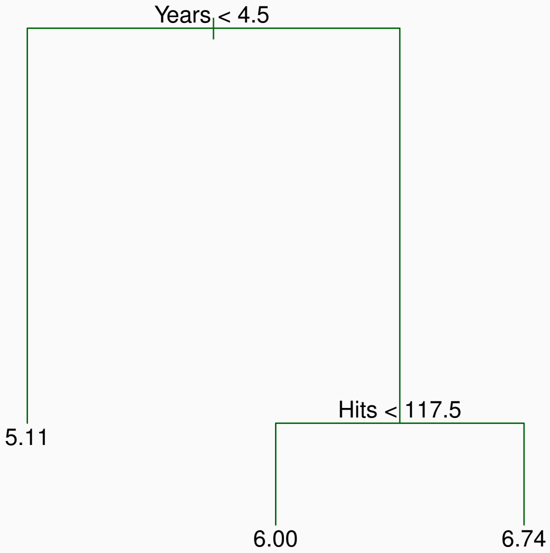
Example: How can we segment the `Hitters` salary data?

Figure 1: Scatter plot of the `Years` and `Hits` variables from the `Hitters` data set. `Salary` is color-coded from low (blue, green) to high (yellow, red).



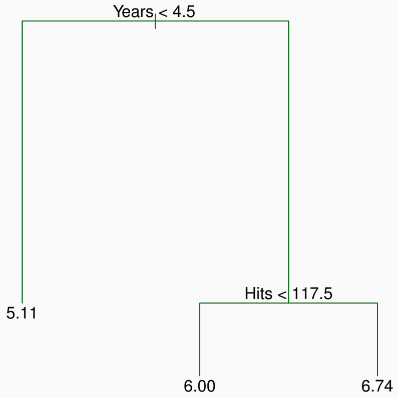
Example: Decision tree for the `Hitters` data

Figure 2: A *regression tree* for predicting the log `Salary` of a baseball player, based on the number of `Years` that he has played in the major leagues and the number of `Hits` that he made in the previous year.



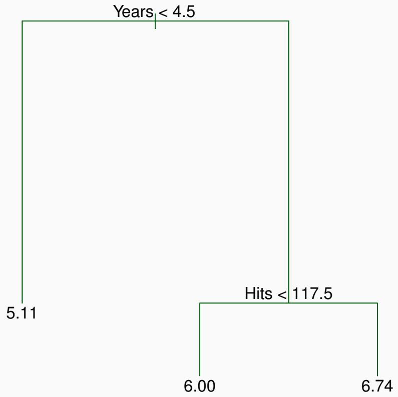
Example: Decision tree for the `Hitters` data

- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to `Years` < 4.5 and the right-hand branch corresponds to `Years` ≥ 4.5.
- The tree has two *internal nodes* and three *terminal nodes* or *leaves*. The number in each leaf is the mean of the response for the observations that fall there.



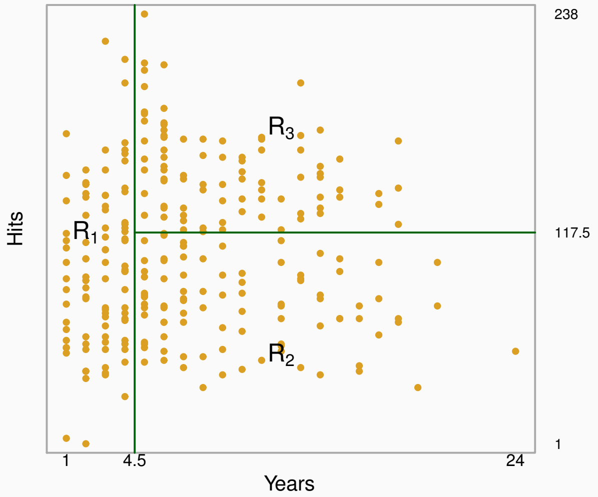
Example: Decision tree for the `Hitters` data

- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to `Years` < 4.5 and the right-hand branch corresponds to `Years` ≥ 4.5.
- The tree has two *internal nodes* and three *terminal nodes* or *leaves*. The number in each leaf is the mean of the response for the observations that fall there.



Example: Decision tree for the `Hitters` data, results

Figure 3: Overall, the tree stratifies/segments the players into three regions of predictors space: $R_1 = \{X | \texttt{Years} < 4.5\}$, $R_2 = \{X | \texttt{Years} \geq 4.5, \texttt{Hits} < 117.5\}$, and $R_3 = X | \texttt{Years} \geq 4.5, \texttt{Hits} \geq 117.5$.



Some vocabulary for tree-based models

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes* or *leaves*.
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split/segmented/stratified are referred to as *internal nodes*.
- In the `Hitters` tree, the two internal nodes are indicated by the text `Years` < 4.5 and `Hits` < 117.5.

Some vocabulary for tree-based models

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes* or *leaves*.
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split/segmented/stratified are referred to as *internal nodes*.
- In the `Hitters` tree, the two internal nodes are indicated by the text `Years` < 4.5 and `Hits` < 117.5.

Some vocabulary for tree-based models

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes* or *leaves*.
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split/segmented/stratified are referred to as *internal nodes*.
- In the `Hitters` tree, the two internal nodes are indicated by the text `Years` < 4.5 and `Hits` < 117.5.

Some vocabulary for tree-based models

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes* or *leaves*.
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split/segmented/stratified are referred to as *internal nodes*.
- In the `Hitters` tree, the two internal nodes are indicated by the text `Years` < 4.5 and `Hits` < 117.5.

Interpretation of the results

- `Years` is the most important factor in determining `Salary`, and players with less experience earn a lower `Salary` than more experienced players.
- Given that a player is less experienced, the number of `Hits` that he made in the previous year seems to play little role in his `Salary`.
- But among the players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher `Salary`.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret, and explain.

Interpretation of the results

- `Years` is the most important factor in determining `Salary`, and players with less experience earn a lower `Salary` than more experienced players.
- Given that a player is less experienced, the number of `Hits` that he made in the previous year seems to play little role in his `Salary`.
- But among the players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher `Salary`.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret, and explain.

Interpretation of the results

- `Years` is the most important factor in determining `Salary`, and players with less experience earn a lower `Salary` than more experienced players.
- Given that a player is less experienced, the number of `Hits` that he made in the previous year seems to play little role in his `Salary`.
- But among the players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher `Salary`.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret, and explain.

Interpretation of the results

- `Years` is the most important factor in determining `Salary`, and players with less experience earn a lower `Salary` than more experienced players.
- Given that a player is less experienced, the number of `Hits` that he made in the previous year seems to play little role in his `Salary`.
- But among the players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher `Salary`.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret, and explain.

Prediction via stratification of the feature space

1. We divide the *predictor space*—that is, the set of possible values for $X_1, X_2, \ldots, X_p$—into J non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, where $j = 1, \ldots, J$, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Prediction via stratification of the feature space

1. We divide the *predictor space*—that is, the set of possible values for $X_1, X_2, \ldots, X_p$—into J non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, where $j = 1, \ldots, J$, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Shapes of the regions

- In theory, the regions could have any shape. We choose to divide the predictor space into (high-dimensional) rectangles or *boxes*, for simplicity and ease of interpretation of the resulting model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Shapes of the regions

- In theory, the regions could have any shape. We choose to divide the predictor space into (high-dimensional) rectangles or *boxes*, for simplicity and ease of interpretation of the resulting model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Greedy, rather than exhaustive tree-building

- It is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as *recursive binary splitting*.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Greedy, rather than exhaustive tree-building

- It is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as *recursive binary splitting*.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Greedy, rather than exhaustive tree-building

- It is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as *recursive binary splitting*.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Greedy, rather than exhaustive tree-building

- It is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as *recursive binary splitting*.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

More details on the tree-building process

- We first select the *predictor* X_j and the *cutpoint* s such that splitting the predictor space into the regions $\{X|X_j < s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data *further* so as to minimize the RSS within each of the resulting regions.
- This time, instead of splitting the entire predictor space, we split one of the two previously identifies regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

More details on the tree-building process

- We first select the *predictor* X_j and the *cutpoint* s such that splitting the predictor space into the regions $\{X|X_j < s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data *further* so as to minimize the RSS within each of the resulting regions.
- This time, instead of splitting the entire predictor space, we split one of the two previously identifies regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

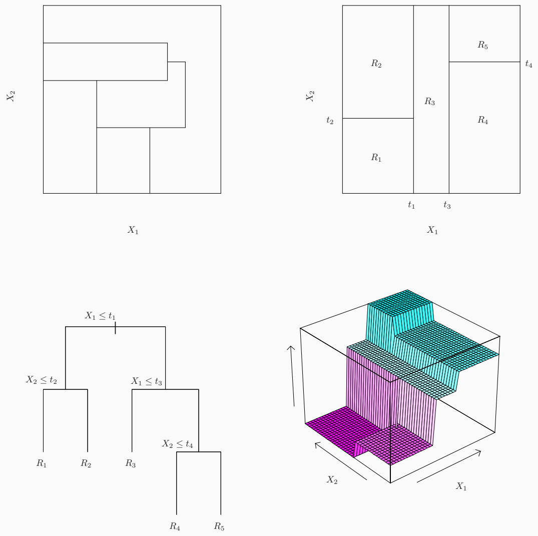
More details on the tree-building process

- We first select the *predictor* X_j and the *cutpoint* s such that splitting the predictor space into the regions $\{X|X_j < s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data *further* so as to minimize the RSS within each of the resulting regions.
- This time, instead of splitting the entire predictor space, we split one of the two previously identifies regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

More details on the tree-building process

- We first select the *predictor* X_j and the *cutpoint* s such that splitting the predictor space into the regions $\{X|X_j < s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data *further* so as to minimize the RSS within each of the resulting regions.
- This time, instead of splitting the entire predictor space, we split one of the two previously identifies regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

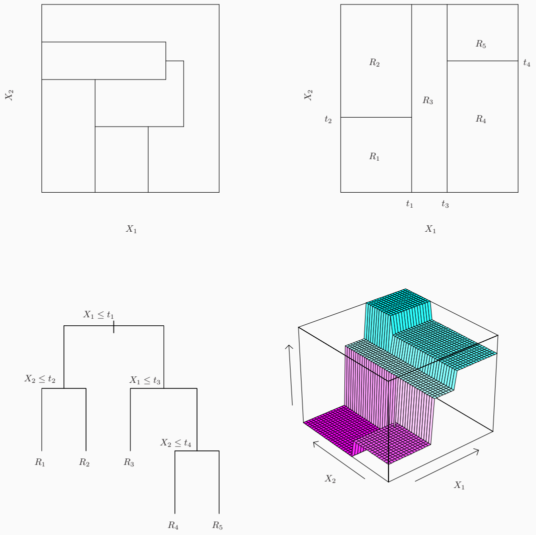
Making predictions using regression trees



We predict the response for a given test observation using the mean of the training observations in the region into which that test observation belongs.

Figure 4: Top left: A partition of two-dimensional feature space that could not result from recursive binary splitting.

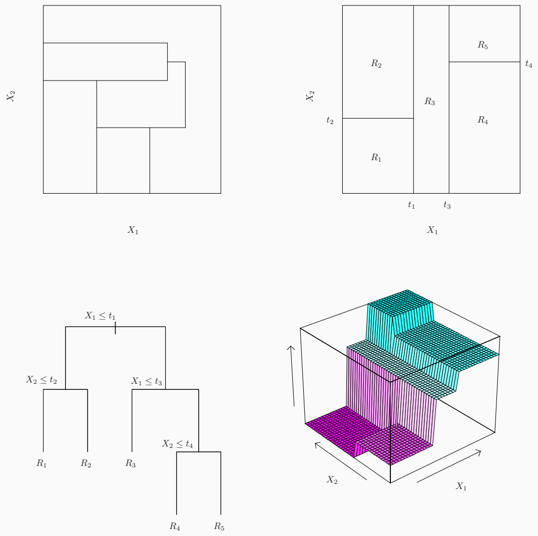
Making predictions using regression trees



We predict the response for a given test observation using the mean of the training observations in the region into which that test observation belongs.

Figure 4: **Top left**: A partition of two-dimensional feature space that could not result from recursive binary splitting. **Top right**: The output of recursive binary splitting on a two-dimensional example.

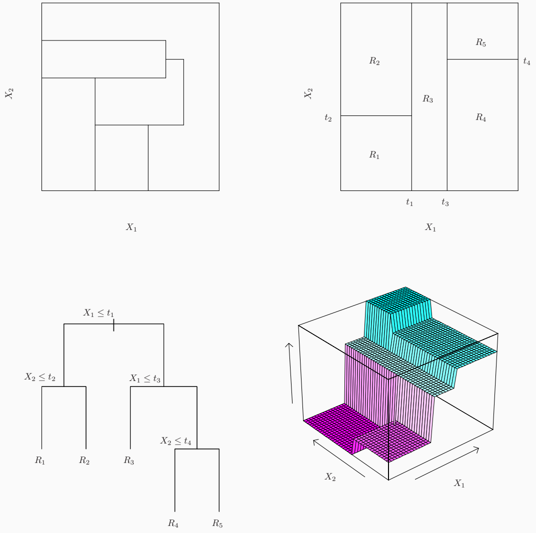
Making predictions using regression trees



We predict the response for a given test observation using the mean of the training observations in the region into which that test observation belongs.

Figure 4: Top left: A partition of two-dimensional feature space that could not result from recursive binary splitting. **Top right**: The output of recursive binary splitting on a two-dimensional example. **Bottom left**: A tree corresponding to the partition in the top right panel.

Making predictions using regression trees



We predict the response for a given test observation using the mean of the training observations in the region into which that test observation belongs.

Figure 4: Top left: A partition of two-dimensional feature space that could not result from recursive binary splitting. **Top right**: The output of recursive binary splitting on a two-dimensional example. **Bottom left**: A tree corresponding to the partition in the top right panel. **Bottom right**: A perspective plot of the prediction surface for that tree.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted*: a seemingly worthless split early on in the tree might be followed by a very good split—that is, a split that leads to a large reduction in RSS later on.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, \dots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted*: a seemingly worthless split early on in the tree might be followed by a very good split—that is, a split that leads to a large reduction in RSS later on.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted*: a seemingly worthless split early on in the tree might be followed by a very good split—that is, a split that leads to a large reduction in RSS later on.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted*: a seemingly worthless split early on in the tree might be followed by a very good split—that is, a split that leads to a large reduction in RSS later on.

Pruning a tree (continued)

- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*.
- *Cost complexity pruning*—also known as *weakest link pruning*—is used to do this.
- We consider a sequence of trees indexed by a nonnegative tuning parameter α. To each value of α corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

  is as small as possible.
- Here $|T|$ indicates the number of terminal nodes or leaves of the tree T, R_m is the box (i.e., the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Pruning a tree (continued)

- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*.
- *Cost complexity pruning*—also known as *weakest link pruning*—is used to do this.
- We consider a sequence of trees indexed by a nonnegative tuning parameter α. To each value of α corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

  is as small as possible.
- Here $|T|$ indicates the number of terminal nodes or leaves of the tree T, R_m is the box (i.e., the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Pruning a tree (continued)

- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*.
- *Cost complexity pruning*—also known as *weakest link pruning*—is used to do this.
- We consider a sequence of trees indexed by a nonnegative tuning parameter α. To each value of α corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

  is as small as possible.
- Here $|T|$ indicates the number of terminal nodes or leaves of the tree T, R_m is the box (i.e., the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Pruning a tree (continued)

- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*.
- *Cost complexity pruning*—also known as *weakest link pruning*—is used to do this.
- We consider a sequence of trees indexed by a nonnegative tuning parameter α. To each value of α corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

  is as small as possible.
- Here $|T|$ indicates the number of terminal nodes or leaves of the tree T, R_m is the box (i.e., the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$

Summary of the *tree-building* algorithm

Algorithm

1. Use *recursive binary splitting* to grow a large tree on the training data, stopping only when each *terminal node* contains fewer than some minimum number of observations.
2. Apply *cost complexity pruning* to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use k-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
    - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
    - 3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

`Hitters` example, continued

- First, we randomly divided the data set in half, yielding 132 observations in the *training set* and 131 observations in the *test set*.
- We then built a large regression tree on the training data and varied α in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

`Hitters` example, continued

- First, we randomly divided the data set in half, yielding 132 observations in the *training set* and 131 observations in the *test set*.
- We then built a large regression tree on the training data and varied α in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

`Hitters` example, continued

- First, we randomly divided the data set in half, yielding 132 observations in the *training set* and 131 observations in the *test set*.
- We then built a large regression tree on the training data and varied α in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

Hitters example, unpruned regression tree

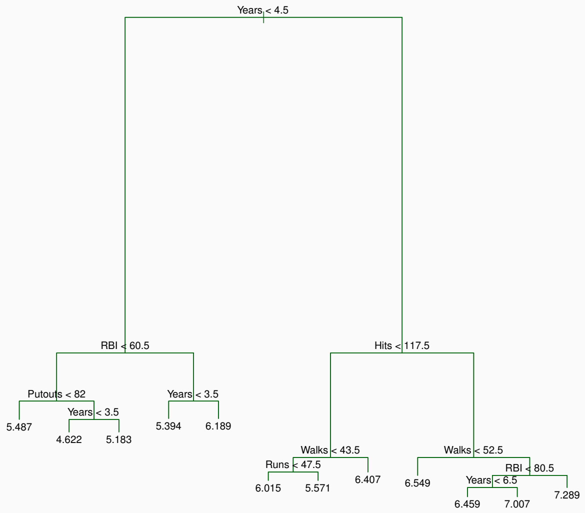
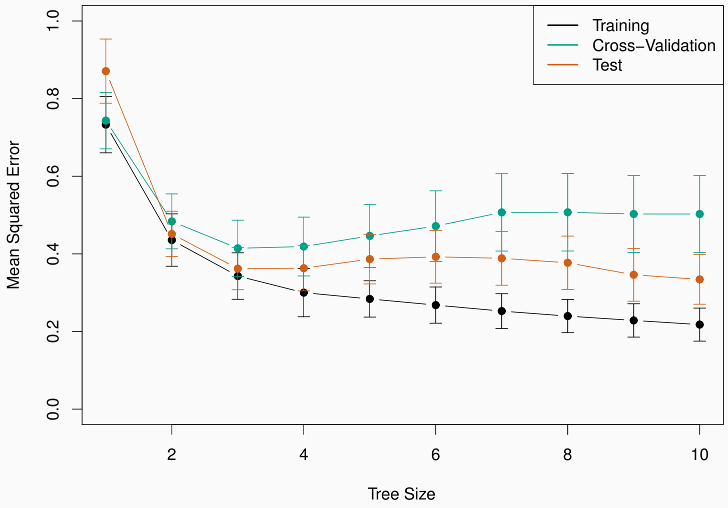


Figure 5: Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown.

`Hitters` example training, cross-validation, test MSE

Figure 6: Regression tree analysis for `Hitters` data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. Minimum cross-validation error occurs at a tree size of three.



Regression trees

Fill in the blanks

1. Predictive power of individual trees is typically ____ than that of other methods.
2. The regions in the predictor spaces produced by a regression tree are called _____ or ____________.
3. The points at which the tree splits are called ____________.

True or false?

1. __ Variables appearing higher up in the tree are more predictively important.
2. __ Terminal nodes may overlap.
3. __ For all observations within a given terminal node, a regression tree will make the same prediction.
4. __ Recursive binary splitting chooses from among all possible partitions of the feature space.
5. __ Cost complexity pruning is used to reduce the risk of overfitting.

Regression trees

Fill in the blanks

1. Predictive power of individual trees is typically lower than that of other methods.
2. The regions in the predictor spaces produced by a regression tree are called leaves or terminal nodes.
3. The points at which the tree splits are called internal nodes.

True or false?

1. T Variables appearing higher up in the tree are more predictively important.
2. F Terminal nodes may overlap.
3. T For all observations within a given terminal node, a regression tree will make the same prediction.
4. F Recursive binary splitting chooses from among all possible partitions of the feature space.
5. T Cost complexity pruning is used to reduce the risk of overfitting.

The basics of decision trees

Classification trees

Classification trees

- Similar to regression trees, but for *qualitative* rather than *quantitative* response.
- Prediction is that each observation belongs to the *most commonly occurring class* of training observations in its region.

Classification trees

- Similar to regression trees, but for *qualitative* rather than *quantitative* response.
- Prediction is that each observation belongs to the *most commonly occurring class* of training observations in its region.

Classification trees, details

- Also uses *recursive binary splitting*, like regression trees.
- RSS cannot be used as a splitting criterion.
- An alternative to RSS is the *classification error rate*; the fraction of training observations in each region that don't belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk})$$

  where $\hat{p}_{mk}$ is the proportion of training observations in the mth region that are from the kth class.
- Classification error is not sufficiently sensitive for tree-growing and in practice two other measures introduced on the next slide are preferred.

Classification trees, details

- Also uses *recursive binary splitting*, like regression trees.
- RSS cannot be used as a splitting criterion.
- An alternative to RSS is the *classification error rate*; the fraction of training observations in each region that don't belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk})$$

  where $\hat{p}_{mk}$ is the proportion of training observations in the mth region that are from the kth class.
- Classification error is not sufficiently sensitive for tree-growing and in practice two other measures introduced on the next slide are preferred.

Classification trees, details

- Also uses *recursive binary splitting*, like regression trees.
- RSS cannot be used as a splitting criterion.
- An alternative to RSS is the *classification error rate*; the fraction of training observations in each region that don't belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk})$$

  where $\hat{p}_{mk}$ is the proportion of training observations in the mth region that are from the kth class.
- Classification error is not sufficiently sensitive for tree-growing and in practice two other measures introduced on the next slide are preferred.

Classification trees, details

- Also uses *recursive binary splitting*, like regression trees.
- RSS cannot be used as a splitting criterion.
- An alternative to RSS is the *classification error rate*; the fraction of training observations in each region that don't belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk})$$

  where $\hat{p}_{mk}$ is the proportion of training observations in the mth region that are from the kth class.
- Classification error is not sufficiently sensitive for tree-growing and in practice two other measures introduced on the next slide are preferred.

Definition of splitting criteria: Gini index

Gini index

The Gini index is a measure of total variance across the K classes given by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

It takes a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.

The Gini index is referred to as a measure of node *purity*—a small value indicates that a node contains predominantly observations from a single class.

Definition of splitting criteria: Gini index

Gini index

The Gini index is a measure of total variance across the K classes given by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

It takes a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.

The Gini index is referred to as a measure of node *purity*—a small value indicates that a node contains predominantly observations from a single class.

Definition of splitting criteria: Deviance, cross-entropy

Deviance, cross-entropy

An alternative to the Gini index is the *deviance* or *cross-entropy* given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

The Gini coefficient and the cross-entropy are numerically very similar.

Definition of splitting criteria: Deviance, cross-entropy

Deviance, cross-entropy

An alternative to the Gini index is the *deviance* or *cross-entropy* given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

The Gini coefficient and the cross-entropy are numerically very similar.

Example: Classification trees applied to `Heart` data

- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of a heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.

Example: Classification trees applied to `Heart` data

- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of a heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.

Example: Classification trees applied to `Heart` data

- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of a heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.

Example: Classification trees applied to `Heart` data

- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of a heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.

Example: Classification trees applied to Heart data

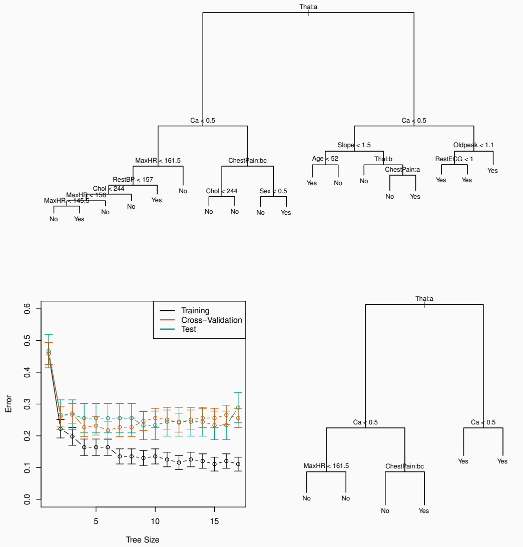


Figure 7: `Heart` data. Top: The unpruned tree. Bottom left: Cross-validation error, training, and test error, for different sizes of the pruned tree. Bottom right: The pruned tree corresponding to the minimal cross-validation error.

Classification trees

Fill in the blanks

1. In classification problems, the RSS is not defined. Instead, one might want to use the ____________________.
2. An optimally sensitive error measure is needed; two such measures include the __________ and the ________.

True or false?

1. __ Classification trees predict that all observations within a terminal node have the most commonly occurring value in that node.
2. __ The Gini index is close to zero when there is a lot of variation within a given class.

Classification trees

Fill in the blanks

1. In classification problems, the RSS is not defined. Instead, one might want to use the classification error rate.
2. An optimally sensitive error measure is needed; two such measures include the Gini index and the deviance.

True or false?

1. T Classification trees predict that all observations within a terminal node have the most commonly occurring value in that node.
2. F The Gini index is close to zero when there is a lot of variation within a given class.

The basics of decision trees

Trees versus linear models

Trees versus linear models

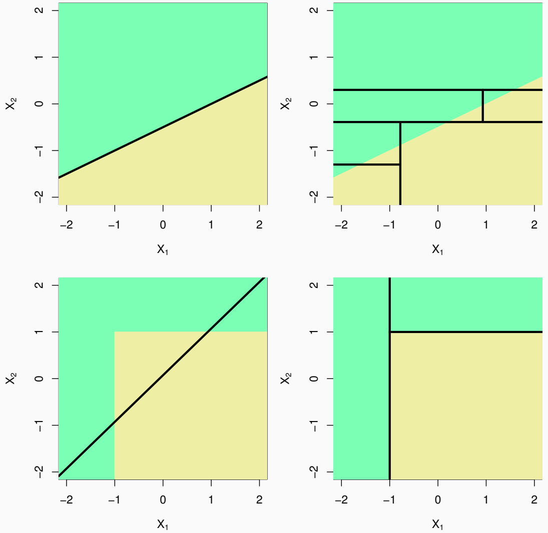


Figure 8: Top row: True linear boundary; Bottom row: true non-linear boundary; Left column: linear model; Right column: tree-based model.

The basics of decision trees

Advantages and disadvantages of trees

Advantages and disadvantages of trees

Advantages

- Easy to explain; even easier than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically and are easily interpreted even by a non-expert (especially if they are small).
- Can handle qualitative predictors without dummies.

Disadvantages

Lower predictive accuracy than other regression and classification approaches we cover.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce such methods next.

Advantages and disadvantages of trees

Advantages

- Easy to explain; even easier than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically and are easily interpreted even by a non-expert (especially if they are small).
- Can handle qualitative predictors without dummies.

Disadvantages

Lower predictive accuracy than other regression and classification approaches we cover.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce such methods next.

Advantages and disadvantages of trees

Advantages

- Easy to explain; even easier than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically and are easily interpreted even by a non-expert (especially if they are small).
- Can handle qualitative predictors without dummies.

Disadvantages

Lower predictive accuracy than other regression and classification approaches we cover.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce such methods next.

Advantages and disadvantages of trees

Advantages

- Easy to explain; even easier than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically and are easily interpreted even by a non-expert (especially if they are small).
- Can handle qualitative predictors without dummies.

Disadvantages

Lower predictive accuracy than other regression and classification approaches we cover.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce such methods next.

Advantages and disadvantages of trees

Advantages

- Easy to explain; even easier than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically and are easily interpreted even by a non-expert (especially if they are small).
- Can handle qualitative predictors without dummies.

Disadvantages

Lower predictive accuracy than other regression and classification approaches we cover.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce such methods next.

Advantages and disadvantages of trees

Advantages

- Easy to explain; even easier than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically and are easily interpreted even by a non-expert (especially if they are small).
- Can handle qualitative predictors without dummies.

Disadvantages

Lower predictive accuracy than other regression and classification approaches we cover.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce such methods next.

Bagging, random forests, boosting

Bagging, random forests, boosting

Bagging

Bootstrap aggregation/bagging

- General-purpose procedure for reducing variance of statistical learning methods.
- Particularly useful and frequently used in decision trees.

Recap: Variance of a set of IID random variables

Recall that for n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of $\bar{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i$ is σ^2/n.

Averaging a set of observations reduces variance. This is nice to know but not practical since we generally do not have access to multiple training sets.

Bootstrap aggregation/bagging

- General-purpose procedure for reducing variance of statistical learning methods.
- Particularly useful and frequently used in decision trees.

Recap: Variance of a set of IID random variables

Recall that for n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of $\bar{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i$ is σ^2/n.

Averaging a set of observations reduces variance. This is nice to know but not practical since we generally do not have access to multiple training sets.

Bootstrap aggregation/bagging

- General-purpose procedure for reducing variance of statistical learning methods.
- Particularly useful and frequently used in decision trees.

Recap: Variance of a set of IID random variables

Recall that for n independent observations $Z_1, \dots, Z_n$, each with variance σ^2, the variance of $\bar{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i$ is σ^2/n.

Averaging a set of observations reduces variance. This is nice to know but not practical since we generally do not have access to multiple training sets.

Bootstrap aggregation/bagging

- General-purpose procedure for reducing variance of statistical learning methods.
- Particularly useful and frequently used in decision trees.

Recap: Variance of a set of IID random variables

Recall that for n independent observations $Z_1, \dots, Z_n$, each with variance σ^2, the variance of $\bar{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i$ is σ^2/n.

Averaging a set of observations reduces variance. This is nice to know but not practical since we generally do not have access to multiple training sets.

Bootstrap aggregation/bagging (continued)

- We can *bootstrap* by taking repeated samples from the (single) training data set.
- We generate B bootstrapped training data sets and then train our method on the bth bootstrapped training set to get $\hat{f}^{*b}(x)$, the prediction at a point x.
- Then average all predictions to get

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

- This process is called *bootstrap aggregation* or *bagging*.

Bootstrap aggregation/bagging (continued)

- We can *bootstrap* by taking repeated samples from the (single) training data set.
- We generate B bootstrapped training data sets and then train our method on the bth bootstrapped training set to get $\hat{f}^{*b}(x)$, the prediction at a point x.
- Then average all predictions to get

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

- This process is called *bootstrap aggregation* or *bagging*.

Bootstrap aggregation/bagging (continued)

- We can *bootstrap* by taking repeated samples from the (single) training data set.
- We generate B bootstrapped training data sets and then train our method on the bth bootstrapped training set to get $\hat{f}^{*b}(x)$, the prediction at a point x.
- Then average all predictions to get

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

- This process is called *bootstrap aggregation* or *bagging*.

Bootstrap aggregation/bagging (continued)

- We can *bootstrap* by taking repeated samples from the (single) training data set.
- We generate B bootstrapped training data sets and then train our method on the bth bootstrapped training set to get $\hat{f}^{*b}(x)$, the prediction at a point x.
- Then average all predictions to get

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

- This process is called *bootstrap aggregation* or *bagging*.

Bagging classification trees

- Above description was for *regression* trees.
- For *classification* trees, record class predicted by each of the B trees and take *majority vote.*
- Overall prediction is most commonly occurring class among B predictions.

Bagging classification trees

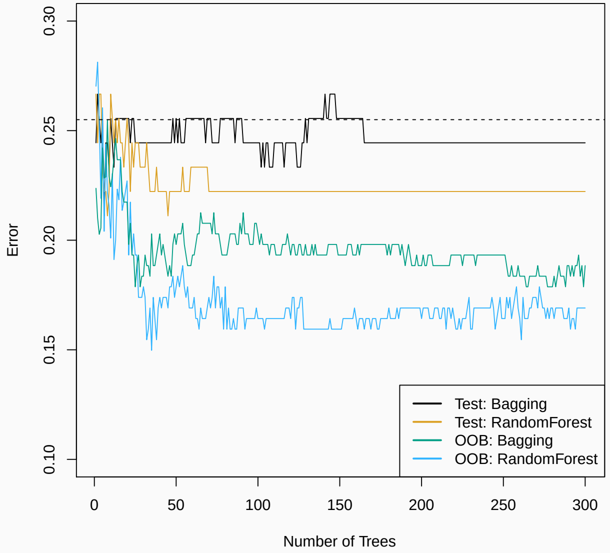
- Above description was for *regression* trees.
- For *classification* trees, record class predicted by each of the B trees and take *majority vote*.
- Overall prediction is most commonly occurring class among B predictions.

Bagging classification trees

- Above description was for *regression* trees.
- For *classification* trees, record class predicted by each of the B trees and take *majority vote*.
- Overall prediction is most commonly occurring class among B predictions.

Example: Bagging the `Heart` data

Figure 9: Bagging and random forest[2] results for `Heart` data. Test error (black and orange) shown as a function of B (number of bootstrap sets). Random forests were applied with $m = \sqrt{p}$. Dashed line indicates test error resulting from single classification tree. Green and blue traces show OOB error, which by chance is considerably lower in this case.



Out-of-bag error estimation

- Straightforward way to estimate test error of bagged model.
- Recall that key to bagging is repeated tree fitting to bootstrapped subsets of observations. On average, each bagged tree uses about 2/3 of the observations.
- Remaining 1/3 not used is called *out-of-bag* (OOB) observations.
- Can predict response for ith observation using each tree in which that observation was OOB. Results in $B/3$ predictions for ith observation, which we can average.
- This is essentially the LOO cross-validation error for bagging, if B is large.

Out-of-bag error estimation

- Straightforward way to estimate test error of bagged model.
- Recall that key to bagging is repeated tree fitting to bootstrapped subsets of observations. On average, each bagged tree uses about 2/3 of the observations.
- Remaining 1/3 not used is called *out-of-bag* (OOB) observations.
- Can predict response for ith observation using each tree in which that observation was OOB. Results in $B/3$ predictions for ith observation, which we can average.
- This is essentially the LOO cross-validation error for bagging, if B is large.

Out-of-bag error estimation

- Straightforward way to estimate test error of bagged model.
- Recall that key to bagging is repeated tree fitting to bootstrapped subsets of observations. On average, each bagged tree uses about 2/3 of the observations.
- Remaining 1/3 not used is called *out-of-bag* (OOB) observations.
- Can predict response for ith observation using each tree in which that observation was OOB. Results in $B/3$ predictions for ith observation, which we can average.
- This is essentially the LOO cross-validation error for bagging, if B is large.

Out-of-bag error estimation

- Straightforward way to estimate test error of bagged model.
- Recall that key to bagging is repeated tree fitting to bootstrapped subsets of observations. On average, each bagged tree uses about 2/3 of the observations.
- Remaining 1/3 not used is called *out-of-bag* (OOB) observations.
- Can predict response for ith observation using each tree in which that observation was OOB. Results in $B/3$ predictions for ith observation, which we can average.
- This is essentially the LOO cross-validation error for bagging, if B is large.

Out-of-bag error estimation

- Straightforward way to estimate test error of bagged model.
- Recall that key to bagging is repeated tree fitting to bootstrapped subsets of observations. On average, each bagged tree uses about 2/3 of the observations.
- Remaining 1/3 not used is called *out-of-bag* (OOB) observations.
- Can predict response for ith observation using each tree in which that observation was OOB. Results in $B/3$ predictions for ith observation, which we can average.
- This is essentially the LOO cross-validation error for bagging, if B is large.

True or false?

1. __ Bagging means fitting a large number of trees to bootstrapped subsets of the training data and averaging their predictions.
2. __ Bagging can only be applied to regression trees, not classification trees.

True or false?

1. T Bagging means fitting a large number of trees to bootstrapped subsets of the training data and averaging their predictions.
2. F Bagging can only be applied to regression trees, not classification trees.

Bagging, random forests, boosting

Random forests

Random forests

- Improvement over bagged trees using a tweak that *decorrelates* trees. Reduces variance when we average trees.
- Like bagging, build several decision trees on bootstrapped training samples.
- Unlike bagging, when building these trees, each time a split is considered, *random selection 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 selection of m predictors is taken at each split and typically we choose $m \approx \sqrt{p}$, that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the `Heart` data).

Random forests

- Improvement over bagged trees using a tweak that *decorrelates* trees. Reduces variance when we average trees.
- Like bagging, build several decision trees on bootstrapped training samples.
- Unlike bagging, when building these trees, each time a split is considered, *random selection 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 selection of m predictors is taken at each split and typically we choose $m \approx \sqrt{p}$, that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the `Heart` data).

Random forests

- Improvement over bagged trees using a tweak that *decorrelates* trees. Reduces variance when we average trees.
- Like bagging, build several decision trees on bootstrapped training samples.
- Unlike bagging, when building these trees, each time a split is considered, *random selection 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 selection of m predictors is taken at each split and typically we choose $m \approx \sqrt{p}$, that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the `Heart` data).

Example: Random forests for gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Example: Random forests for gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Example: Random forests for gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Example: Random forests for gene expression data

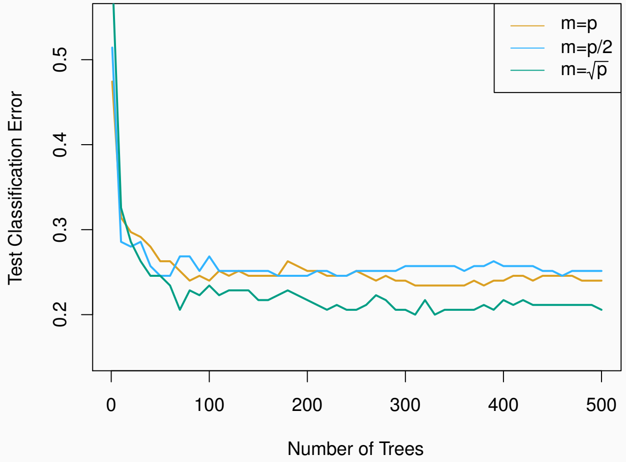
- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Example: Random forests for gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Example: Random forests for gene expression data

Figure 10: Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node. Random forests $(m > p)$ lead to a slight improvement over bagging $(m = p)$. A single classification tree has an error rate of 45.7 %.



Random forests

True or false?

1. __ Random forest models are an extension of bagging.
2. __ Random forest models reduce variance by producing trees that are less strongly correlated than in bagging.
3. __ In a random forest model, each new tree is fit to a bootstrapped sample from the whole set of predictors.

Random forests

True or false?

1. T Random forest models are an extension of bagging.
2. T Random forest models reduce variance by producing trees that are less strongly correlated than in bagging.
3. F In a random forest model, each new tree is fit to a bootstrapped sample from the whole set of predictors.

Bagging, random forests, boosting

Boosting

Boosting

- Like *bagging*, general approach applicable to many statistical learning methods in regression or classification. We only discuss boosting for decision trees.
- Bagging creates multiple copies of the original training data using bootstrap; fitting a separate tree to subsets of each copy and then recombining all of the trees in order to create a single predictive model.
- Each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works similarly, but trees are grown *sequentially*: each tree is grown from information from previously grown trees.

Boosting

- Like *bagging*, general approach applicable to many statistical learning methods in regression or classification. We only discuss boosting for decision trees.
- Bagging creates multiple copies of the original training data using bootstrap; fitting a separate tree to subsets of each copy and then recombining all of the trees in order to create a single predictive model.
- Each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works similarly, but trees are grown *sequentially*: each tree is grown from information from previously grown trees.

Boosting

- Like *bagging*, general approach applicable to many statistical learning methods in regression or classification. We only discuss boosting for decision trees.
- Bagging creates multiple copies of the original training data using bootstrap; fitting a separate tree to subsets of each copy and then recombining all of the trees in order to create a single predictive model.
- Each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works similarly, but trees are grown *sequentially*: each tree is grown from information from previously grown trees.

Boosting

- Like *bagging*, general approach applicable to many statistical learning methods in regression or classification. We only discuss boosting for decision trees.
- Bagging creates multiple copies of the original training data using bootstrap; fitting a separate tree to subsets of each copy and then recombining all of the trees in order to create a single predictive model.
- Each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works similarly, but trees are grown *sequentially*: each tree is grown from information from previously grown trees.

Boosting algorithm for regression trees

Algorithm

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   2.1 Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   2.3 Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

The idea behind *boosting* for regression trees

- 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*.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- 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 wehre 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.

The idea behind *boosting* for regression trees

- 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*.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- 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 wehre 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.

The idea behind *boosting* for regression trees

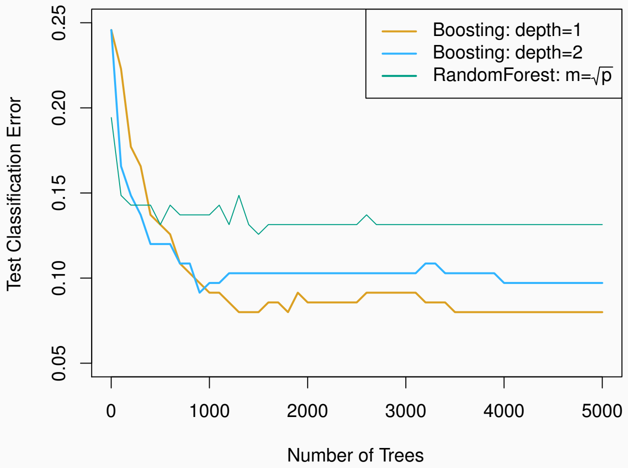
- Unlike fitting a single large decision tree to the data, which amounts to *fitting the data hard* and potentially overfitting, the boosting approach instead *learns slowly*.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- 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 wehre 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.

The idea behind *boosting* for regression trees

- 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*.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- 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 wehre 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.

Example: Boosting for gene expression data

Figure 11: Results from performing boosting and random forests on 15-class gene expression data set to predict *cancer* versus *normal*. The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant. The test error rate for a single tree is 24 %.



Tuning parameters for boosting

- 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.
- The *shrinkage parameter* λ, 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 problem. Very small λ can require using a very large value of B to achieve good performance.
- The *number of splits* d in each tree, which 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 and resulting in an additive model. 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.

Tuning parameters for boosting

- 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.
- The *shrinkage parameter* λ, 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 problem. Very small λ can require using a very large value of B to achieve good performance.
- The *number of splits* d in each tree, which 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 and resulting in an additive model. 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.

Tuning parameters for boosting

- 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.
- The *shrinkage parameter* λ, 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 problem. Very small λ can require using a very large value of B to achieve good performance.
- The *number of splits* d in each tree, which 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 and resulting in an additive model. 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.

Fill in the blanks

1. The parameter controlling the rate of learning is denoted by __.
2. The parameter controlling the number of trees fit is denoted by __.

True or false?

1. __ Boosting grows additional trees sequentially, the next tree learning from the residuals of the current one.
2. __ A smaller B will typically require a larger λ and vice versa.

Boosting

Fill in the blanks

1. The parameter controlling the rate of learning is denoted by λ.
2. The parameter controlling the number of trees fit is denoted by B.

True or false?

1. T Boosting grows additional trees sequentially, the next tree learning from the residuals of the current one.
2. T A smaller B will typically require a larger λ and vice versa.

Conclusion

- Decision trees are simple and interpretable models for regression and classification.
- However they are often note competitive with other methods in terms of prediction accuracy.
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods—random forests and boosting—are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Conclusion

- Decision trees are simple and interpretable models for regression and classification.
- However they are often note competitive with other methods in terms of prediction accuracy.
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods—random forests and boosting—are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Conclusion

- Decision trees are simple and interpretable models for regression and classification.
- However they are often note competitive with other methods in terms of prediction accuracy.
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods—random forests and boosting—are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Conclusion

- Decision trees are simple and interpretable models for regression and classification.
- However they are often note competitive with other methods in terms of prediction accuracy.
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods—random forests and boosting—are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

This material draws extensively on James, G., Witten, D., Hastie, T. & Tibshirani, R. (2021). *An introduction to statistical learning* and the lecture slides available from these authors.