Decision Trees
Ch8 ISLP; CH2 IMLP

Zhongjian Lin
University of Georgia

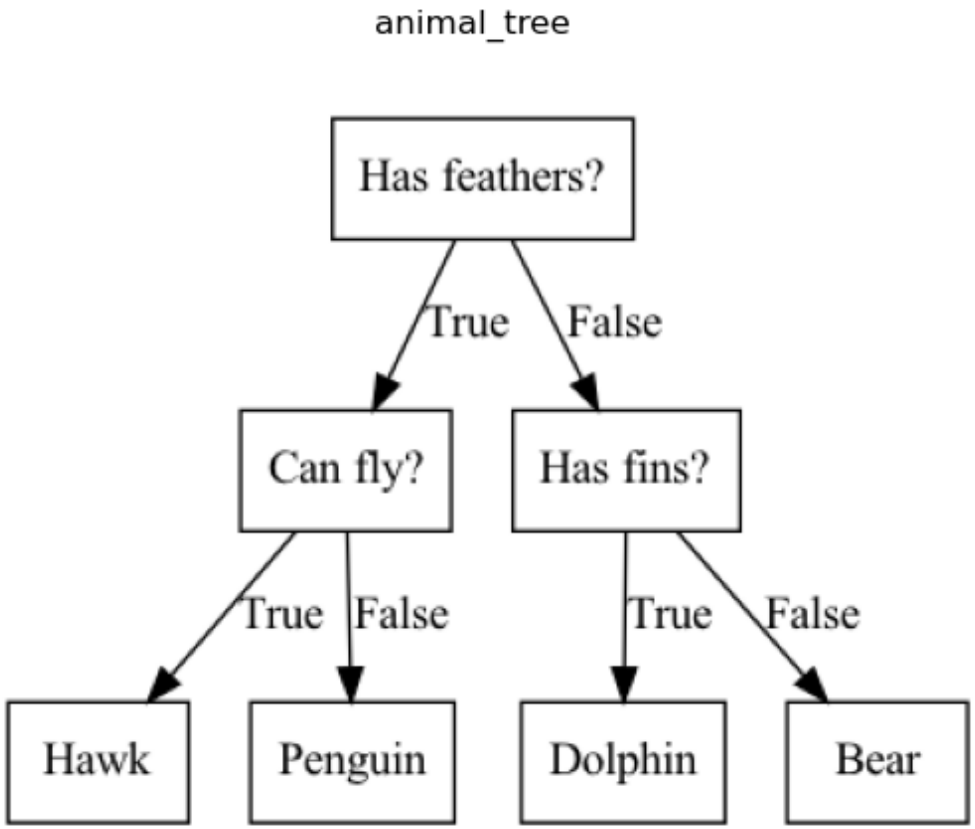
September 17, 2024

Decision Tree

Tree-based methods for regression and classification involve stratifying or segmenting the predictor space into a number of simple regions. The goal is to create a model that predicts the value of a target variable based on several input variables. In order to make a prediction for a given observation, we typically use the mean or the mode response value for the training observations in the region to which it belongs. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision tree methods.

Decision Trees

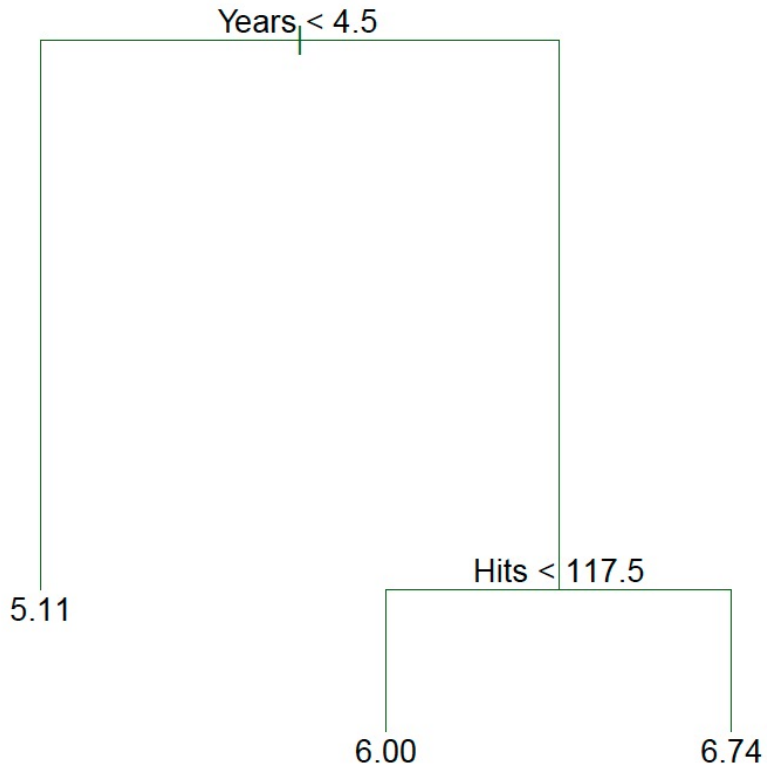
Decision trees are a widely used models for classification and regression tasks. Essentially, they learn a hierarchy of "if-else" questions, leading to a decision.



Instead of building these models by hand, we can learn them from data using supervised learning.

Regression Trees

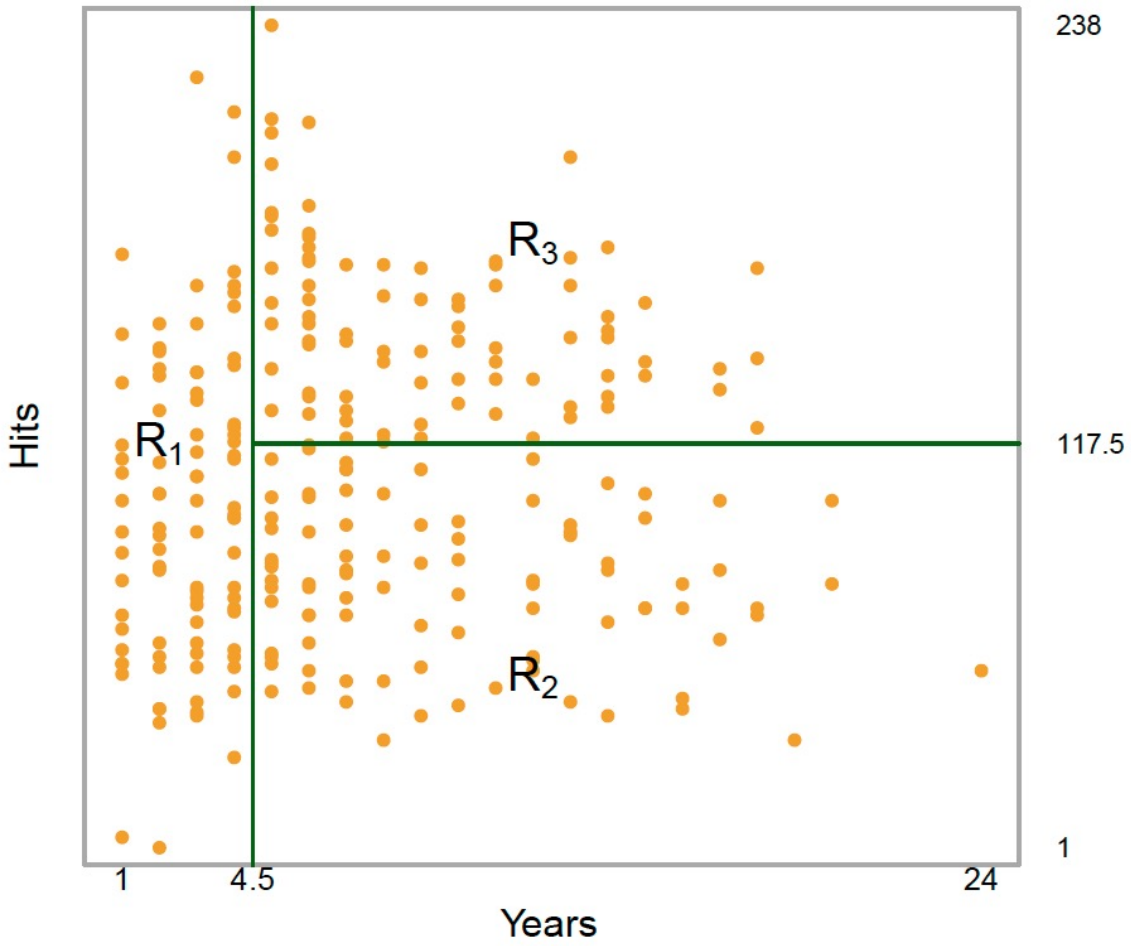
We predict a baseball player's Salary based on Years (the number of years that he has played in the major leagues) and Hits (the number of hits that he made in the previous year).



It consists of a series of splitting rules, starting at the top of the tree. The top split assigns observations having *Years* < 4.5 to the left branch. Then the senior group is further subdivided by Hits.

Predicting Baseball Players' Salaries Using Regression Trees

We arrive at three regions developed from the regression tree. In keeping with the tree analogy, the regions R_1, R_2, and R_3 are known as terminal nodes or leaves of the tree.



Prediction vis Stratification of the Feature Space

There are two steps to build a regression tree

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

Stratification

How do we construct the regions $R_1, \cdots, R_J$? In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model. The goal is to find boxes $R_1, \cdots, R_J$ that minimize the RSS, given by

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

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

Recursive Binary Splitting

The recursive binary splitting approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.

1. We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
2. In greater detail, for any j and s, we define the pair of half-planes

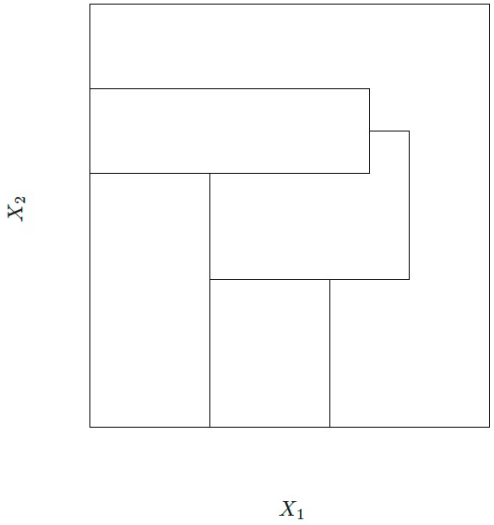
$$R_1(j,s) = \{X|X_j < s\}, \quad R_2(j,s) = \{X|X_j \geq s\}.$$

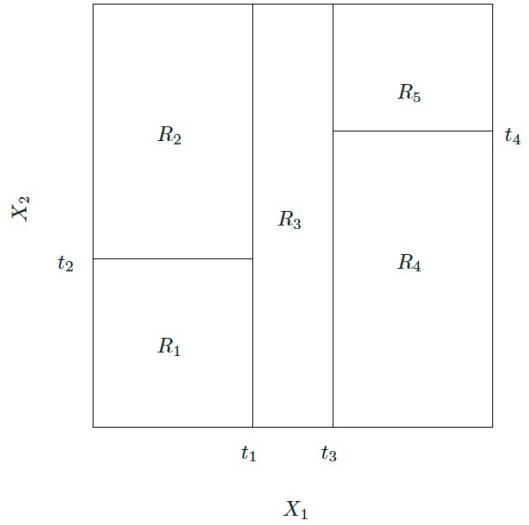
   We seek the value of j and s that minimize the equation

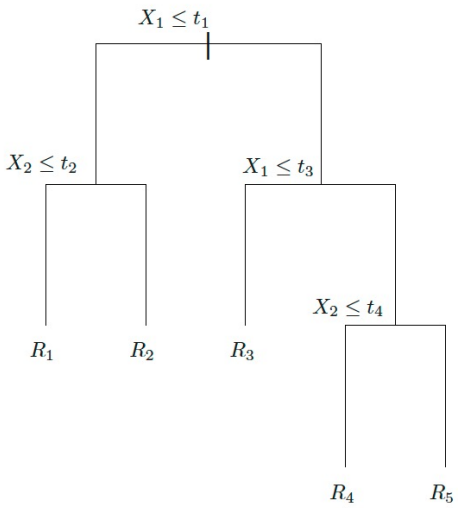
$$\sum_{x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

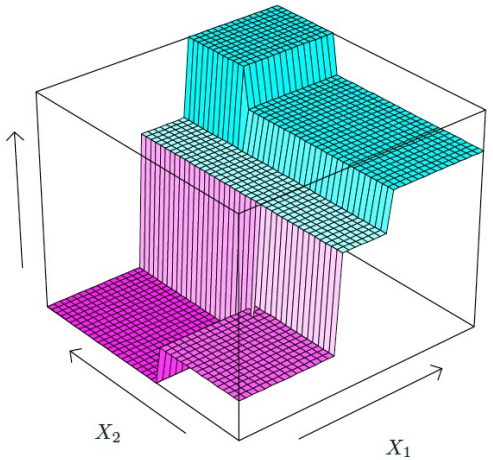
3. Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
4. Once the regions $R_1, \cdots, R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

A five regions example



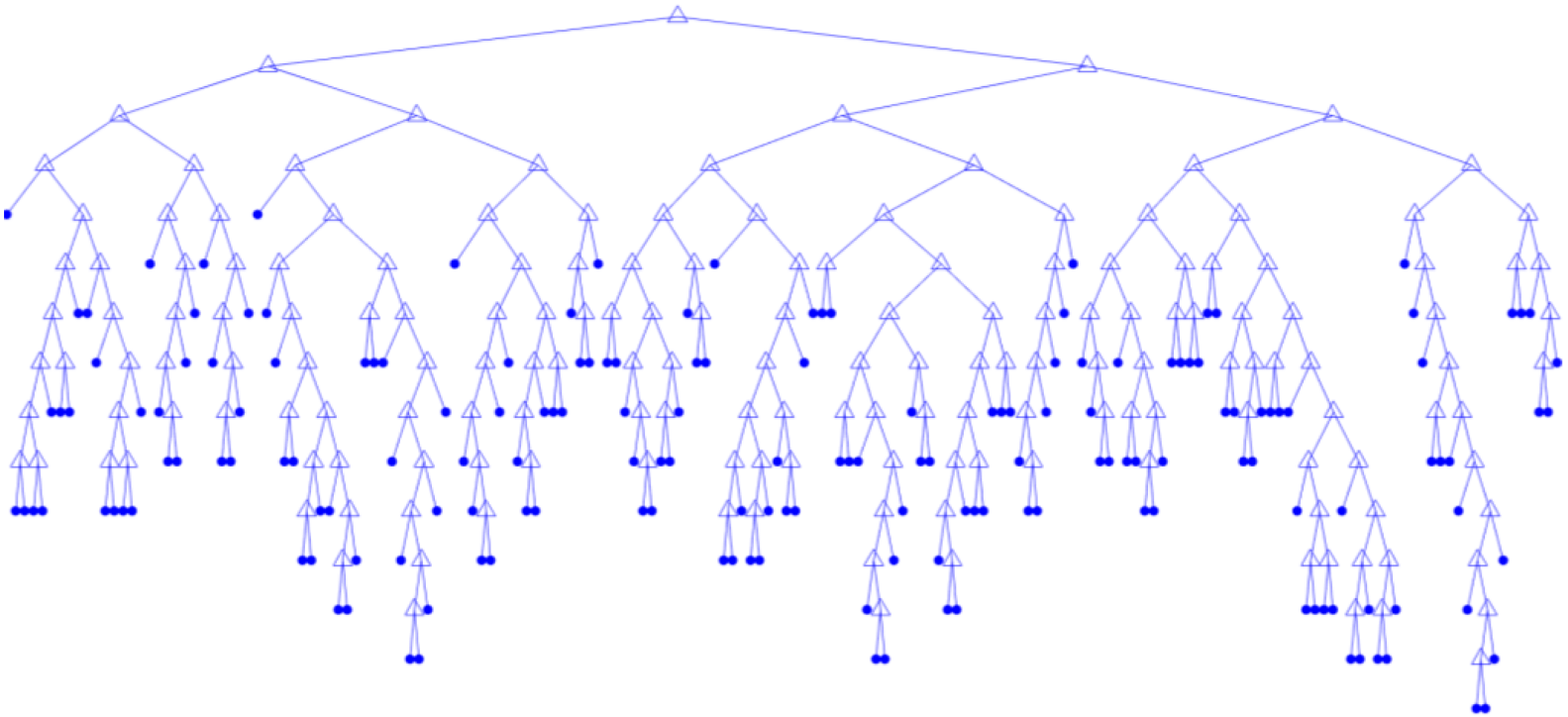






Overfitting

The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. This is because the resulting tree might be too complex.



Tree Pruning

A smaller tree with fewer splits (that is, fewer regions $R_1, \cdots, R_J$) 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 build 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 since a seemingly worthless split early on in the tree might be followed by a very good split.
- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree. Intuitively, our goal is to select a subtree that subtree leads to the lowest test error rate. Instead, we need a way to select a small set of subtrees for consideration.

Cost complexity pruning

We consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

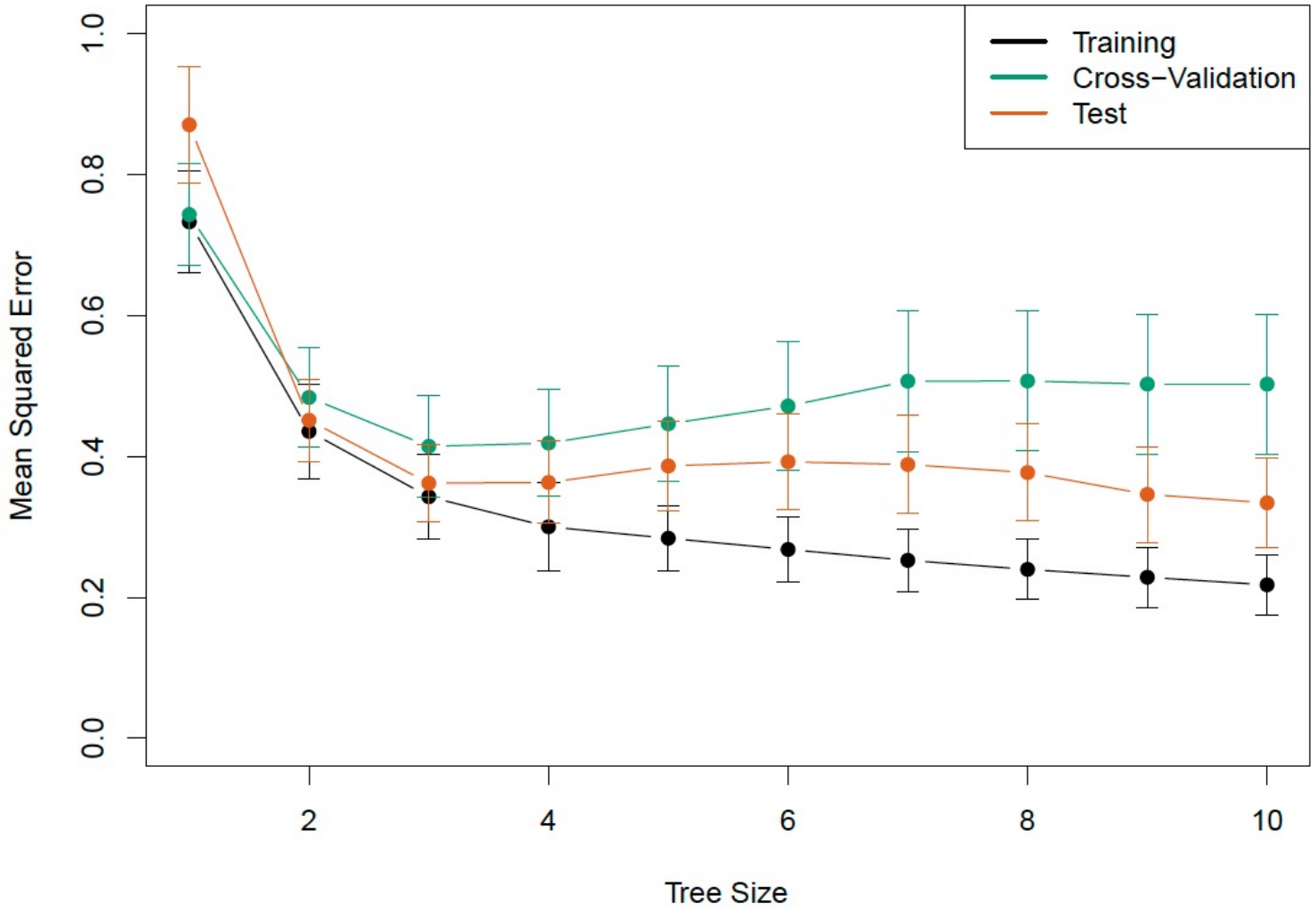
$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \tag{1}$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the predicted response associated with R_m. The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. When $\alpha = 0$, then the subtree T will simply equal T_0, because then Equation (1) just measures the training error. However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity in Equation (1) will tend to be minimized for a smaller subtree.

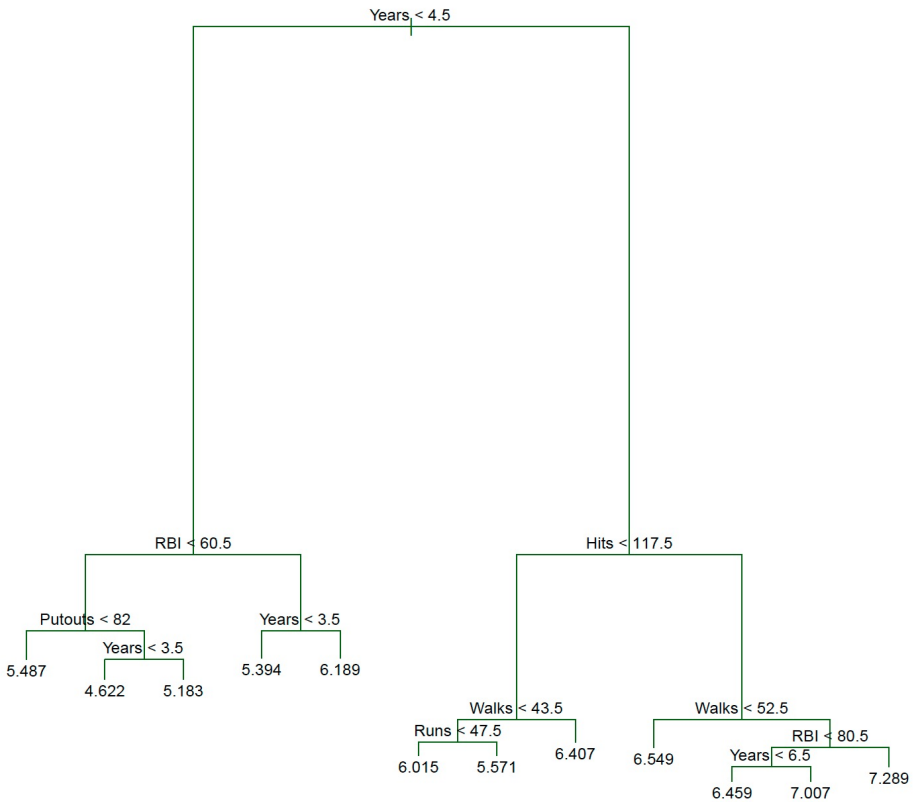
Algorithm

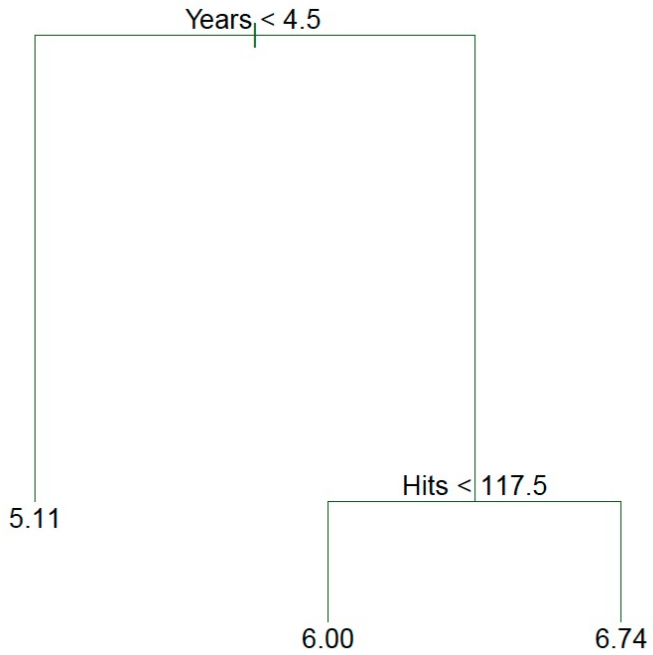
1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \cdots, K$:
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α. Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Hitters


Training
Cross−Validation
Test
Mean Squared Error
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Tree Size

Pruning Regression Tree





Classification Tree

- A classification tree is very similar to a regression tree, except that it is classification used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.
- In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region.
- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.

Criterion of Growing a Tree

We use classification error rate as the criterion for the recursive binary splitting. The classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

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

Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

- Gini Index: a measure of node purity-a small value indicates that a node contains predominantly observations from a single class.

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

- Entropy: take on a small value if the mth node is pure.

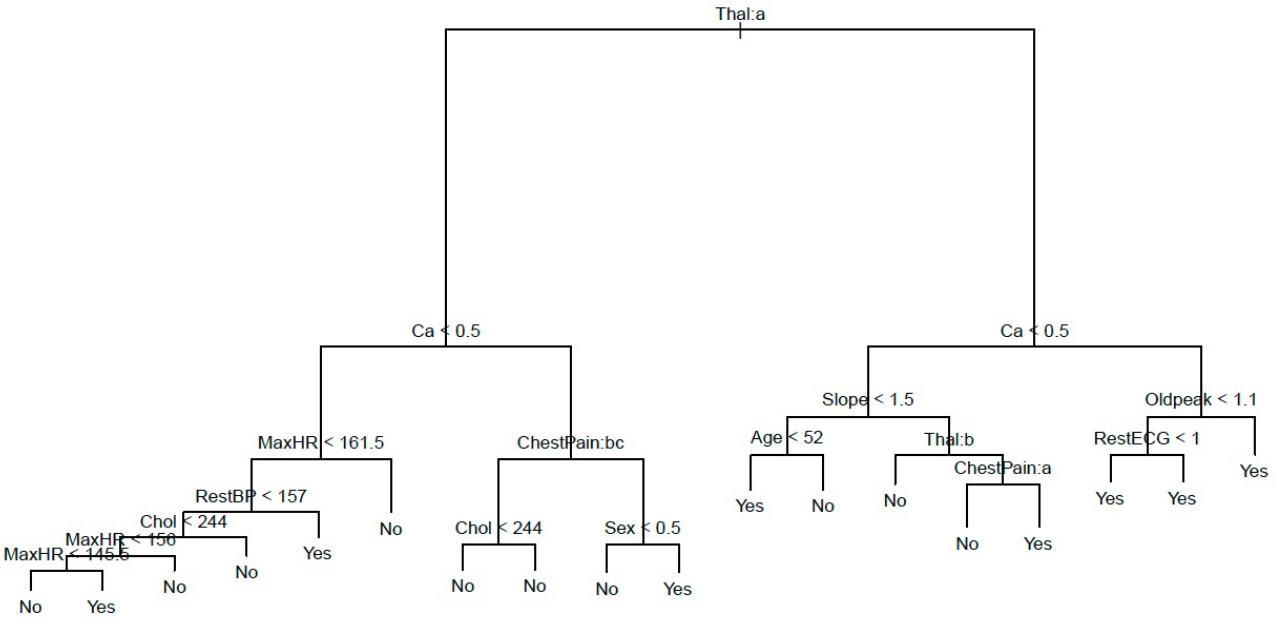
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$$

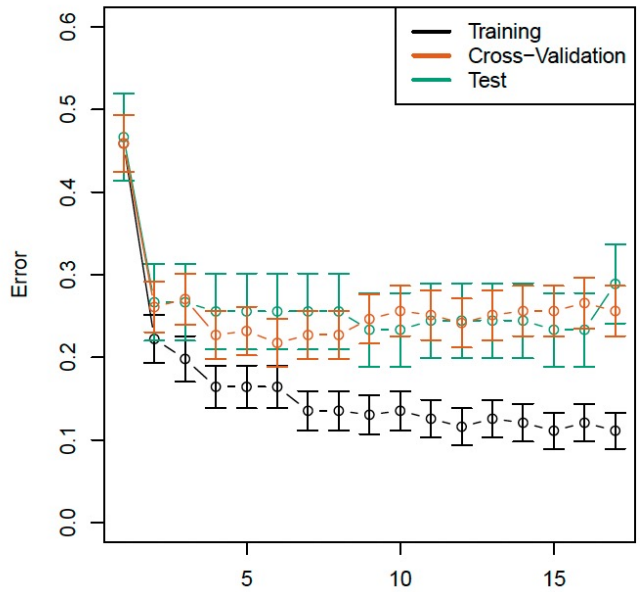
In fact, it turns out that the Gini index and the entropy are quite similar numerically.

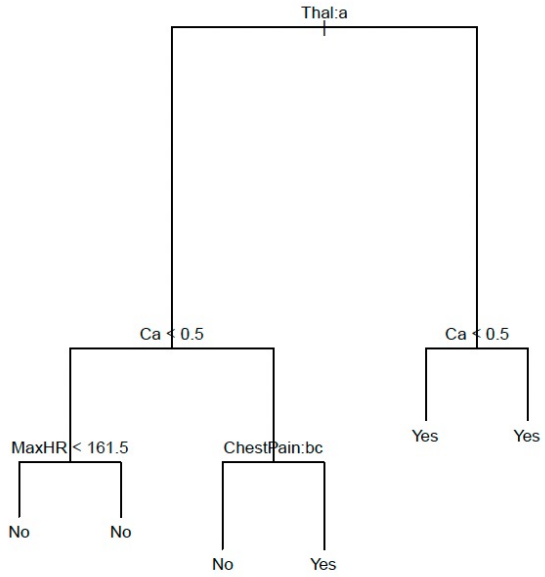
Classification Tree

When building a classification tree, either the Gini index or the entropy are typically used to evaluate the quality of a particular split, since these two approaches are more sensitive to node purity than is the classification error rate. Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

Heart Disease



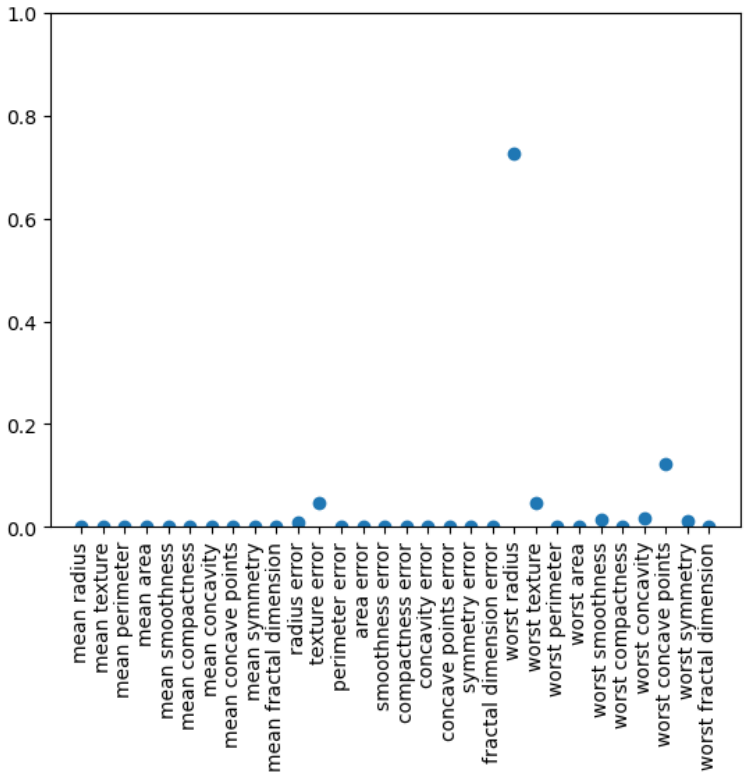




Feature Importance in trees

We can derive some statistics to summarize the workings of the tree. The most commonly used summary is feature importance, which rates how important each feature is for the decision a tree makes.

```
import matplotlib.pyplot as plt
plt.plot(tree.feature_importances_, 'o')
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.ylim(0, 1)
```



Feature Importance

- Here, we see that the feature used at the top split ("worst radius") is by far the most important feature. This confirms our observation in analyzing the tree, that the first level already separates the two classes fairly well.
- However, if a feature has a low feature_importance, it doesn't mean that this feature is uninformative. It only means that this feature was not picked by the tree, likely because another feature encodes the same information.
- In contrast to the coefficients in linear models, feature importances are always positive, and don't encode which class a feature is indicative of. The feature importances tell us that worst radius is important, but it does not tell us whether a high radius is indicative of a sample being "benign" or "malignant". In fact, there might not be such a simple relationship between features and class, as you can see in the example below:

Trees Versus Linear Models

Regression and classification trees have a very different flavor from the more classical approaches for regression and classification

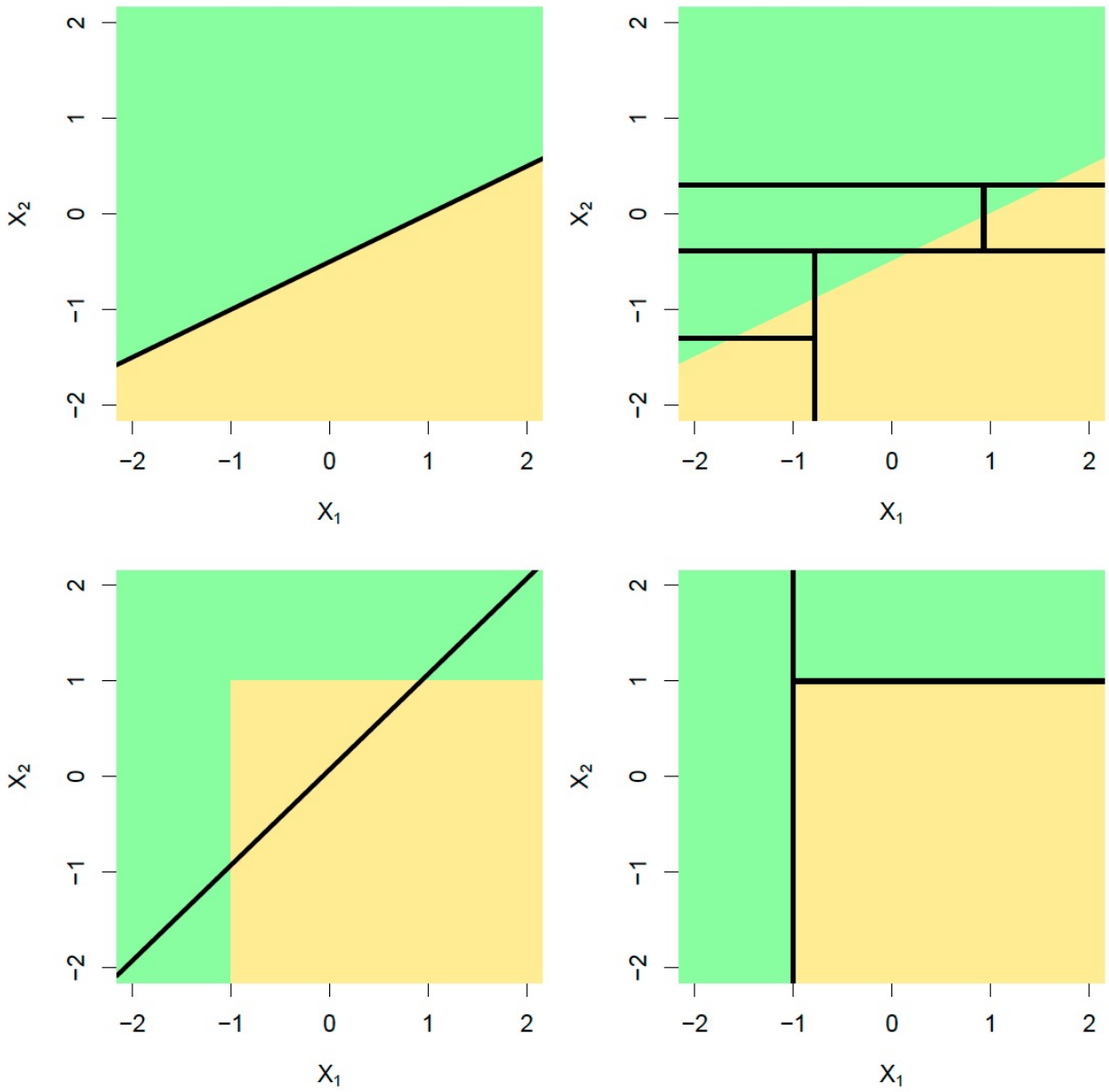
$$\text{Linear Regression:} f(X) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j. \tag{2}$$

$$\text{Regression Tree: } f(X) = \sum_{m=1}^{M} c_m 1\{X \in R_m\}, \tag{3}$$

where $R_1, \cdots, R_M$ represent a partition of feature space. Which model is better? It depends on the problem at hand.

- If the relationship between the features and the response is well approximated by a linear model as in Equation (2), then an approach such as linear regression will likely work well, and will outperform a method such as a regression tree that does not exploit this linear structure.
- If instead there is a highly nonlinear and complex relationship between the features and the response as indicated by model in Equation (3), then decision trees may outperform classical approaches.

Tree Versus Linear Models



Advantage

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression.
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.

Disadvantage

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches
- Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.

However, by aggregating many decision trees, using methods like *bagging*, *random forests*, and *boosting*, the predictive performance of trees can be substantially improved.