Predictive Analytics (ISE529)

Tree-Based Methods (I)

Dr. Tao Ma
ma.tao@usc.edu

Tue/Thu, May 22 - July 1, 2025, Summer

USC Viterbi
School of Engineering
Daniel J. Epstein Department of Industrial and Systems Engineering

USC University of Southern California

Outline

- Build a decision tree
- Regression trees
- Classification trees
- Bagging, Random Forests (RF), Boosting

TREE-BASED METHODS

Tree-Based Methods

- Tree-based methods can be applied to both regression and classification problems.
- These involve *stratifying* or *segmenting* the predictor space into a number of simple regions.
- 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.

Pros and Cons

- Tree-based methods are simple and useful for interpretation. However, they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence, we also discuss *bagging*, *random forests*, and *boosting*. These methods produce multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

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

BUILD A SIMPLE DECISION TREE

Build a Simple Decision Tree

- This example builds classification models in the form of a tree structure.
- It breaks down a dataset into smaller subsets. The final result is a tree with *decision nodes* and *leaf nodes*.
- A decision node has two or more branches. Leaf node represents a classification or decision.
- The *topmost decision node* in a tree which corresponds to the best predictor called *root node*.
- Decision trees can handle both categorical and numerical data.

Build a Simple Decision Tree

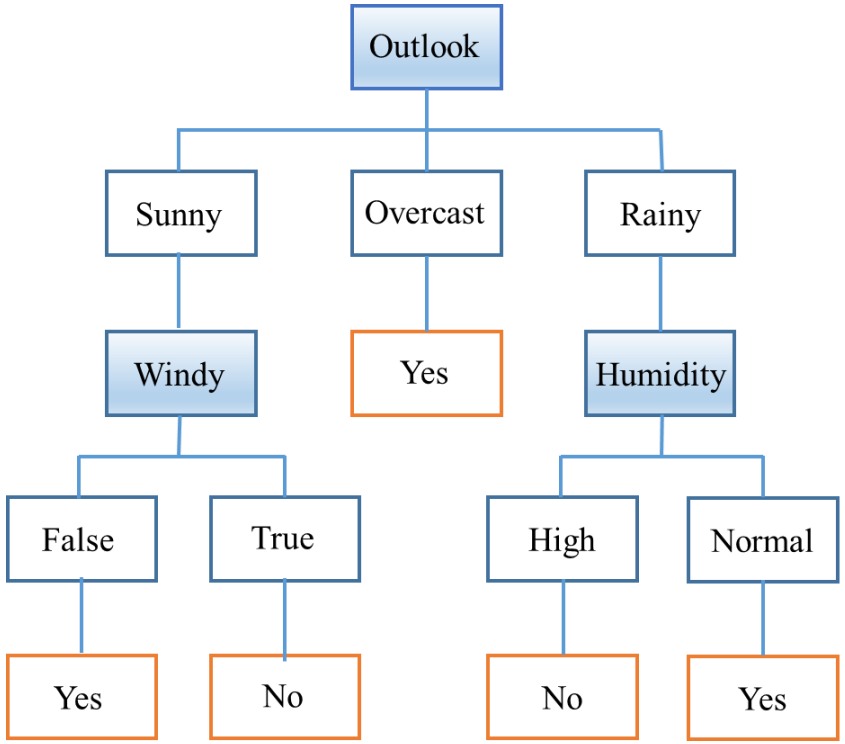
USC Viterbi School of Engineering

Data Set:

Predictors Response/Target

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Decision Tree Outcome



Build a Simple Decision Tree

- The algorithm for building this decision tree proposed by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking.
- The algorithm uses *Entropy and Information Gain* to construct a decision tree.
- Decision tree includes all predictors with the dependence assumptions between predictors.

Entropy and Information Gain

A decision tree is built top-down from a *root node* and involves partitioning the data into subsets that contain instances with similar values (homogenous).

The algorithm uses **entropy** to calculate the *homogeneity* of a sample.

If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.

Entropy and Information Gain

Calculate *Entropy* using the *frequency table* of **one attribute**:

$$Entropy(S) = \sum_{i=1}^{n} -p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(Play Golf) = Entropy(5,9)
= Entropy(0.36, 0.64)
= $-0.36\log_2(0.36) - 0.64\log_2(0.64)$
= 0.94

More Examples for Entropy

More examples of calculating **Entropy** using the frequency table:

C1	0
C2	6

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Entropy = $-0 \log_2(0) - 1 \log_2(1) = -0 - 0 = 0$

C1	1
C2	5

P(C1) = 1/6 P(C2) = 5/6

Entropy = $-(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.65$

C1	2
C2	4

P(C1) = 2/6 P(C2) = 4/6

Entropy = $-(2/6) \log_2(2/6) - (4/6) \log_2(4/6) = 0.92$

Entropy and Information Gain

Calculate **weighted entropy** using the frequency table of **two attributes**:

$$Entropy(T, X) = \sum_{c \in X} p(c) E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

Entropy(T=PlayGolf, X=Outlook)
= P(Sunny)*E(3,2) + P(Overcast)*E(4,0) + P(Rainy)*E(2,3)
= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971
= 0.693

Entropy and Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding **attribute** (decision nodes) that returns the highest information gain (i.e., the most homogeneous branches).

Step 1: Calculate entropy of the response/target (e.g., Play Golf).

$$\begin{aligned}\text{Entropy(Play Golf)} &= \text{Entropy}(5,9)\\ &= \text{Entropy}(0.36, 0.64)\\ &= -0.36\log_2(0.36) - 0.64\log_2(0.64)\\ &= 0.94\end{aligned}$$

Entropy and Information Gain

Step 2:

The dataset is then split on the different attributes. Calculate the entropy for each attribute. The resulting **weighted entropy** is subtracted from the entropy before the split. The result is the Information Gain or decrease in entropy.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

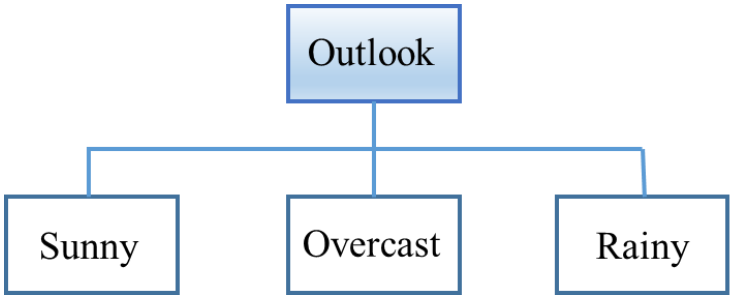
		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

Entropy and Information Gain

Step 3:

Choose attribute with the largest **information gain as the decision node**, divide the dataset by its branches and repeat the same process for every branch.

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14
Information Gain = 0.247				

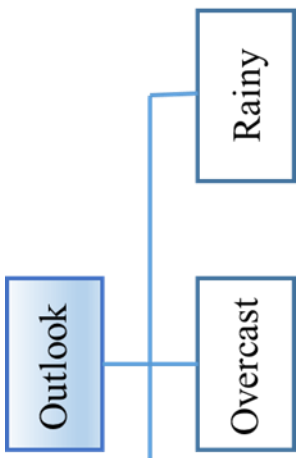


Entropy and Information Gain

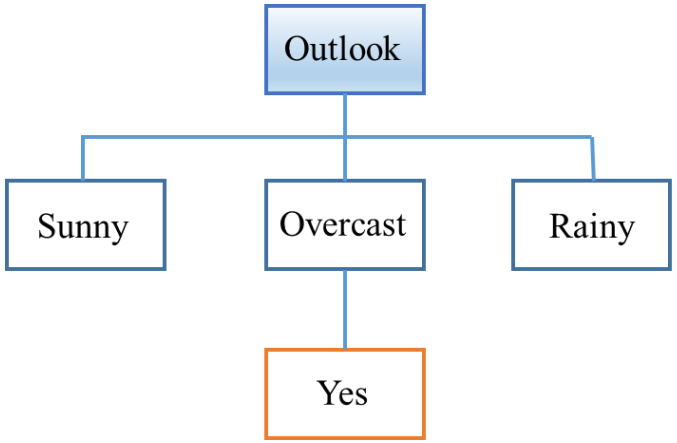
Step 4a:

Calculate the entropy of the target for each branch.

A branch that its entropy of target equals 0 is a **leaf node**.



Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Hot	High	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

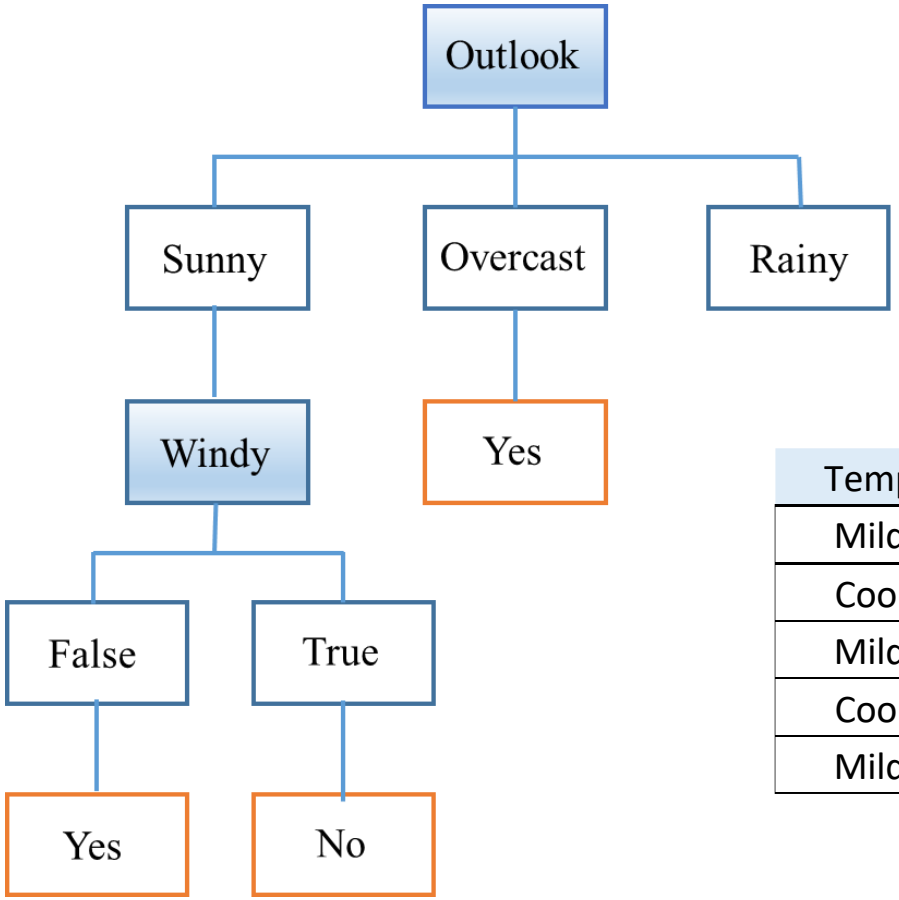


Decision Tree

Step 4b:

A branch that its entropy of target is more than 0 needs further splitting.

Calculate the information gain for non-leaf branches.

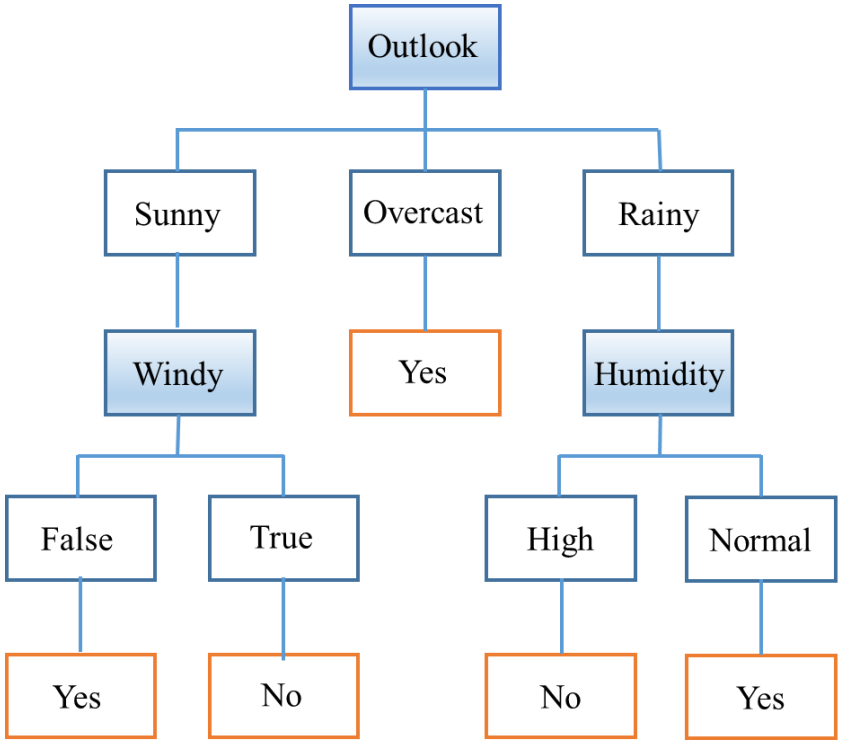


Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No

Decision Tree

Step 5:
The algorithm is run recursively on the non-leaf branches until all data is classified.

1. If (outlook==sunny) and (windy==false) then play=yes
2. If (outlook==sunny) and (windy==true) then play=No
3. If (outlook==overcast) then play=yes
4. If (outlook==rainy) and (humidity==high) then play=No
5. If (outlook==rainy) and (humidity==normal) then play=yes

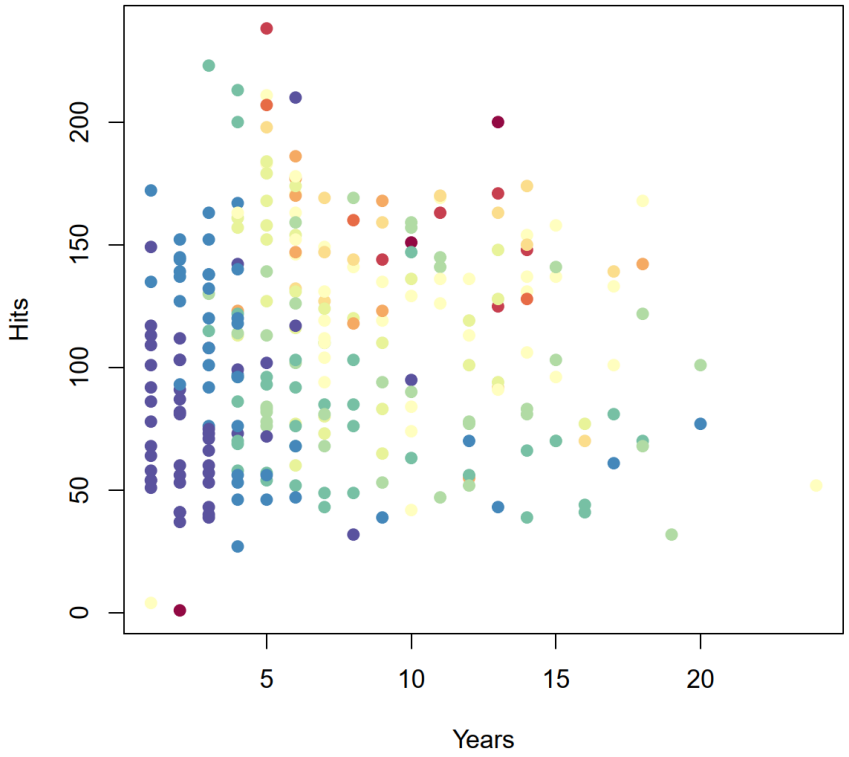


REGRESSION TREES

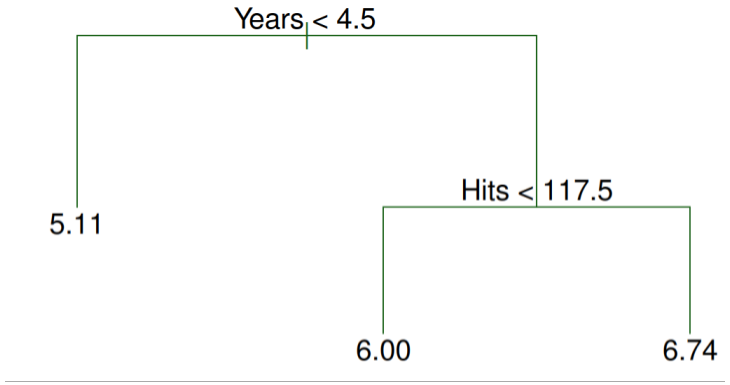
Example

We begin with a simple example.

- We use the *Hitters* data set to predict a baseball player's (log) 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).
- Salary is color-coded from low (blue, green) to high (yellow, red)



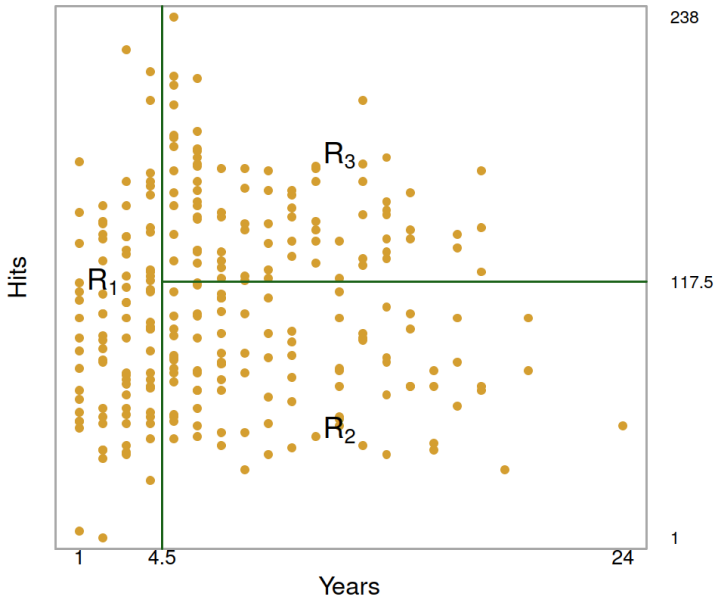
Decision tree for these data



The regression 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 (cont'd)

- The top split assigns observations having Years<4.5 to the left branch. The predicted salary for these players is given by **the mean response value** for the players in the data set with Years<4.5. For such players, the mean log salary is 5.107, and so we make a prediction of $e^{5.107}$ thousands of dollars, i.e., \$165,174.
- Players with Years>=4.5 are assigned to the right branch, and then that group is further subdivided by Hits. Overall, the tree stratifies or segments the players into three regions of **predictor space**:



R1 ={X | Years< 4.5}
R2 ={X | Years>=4.5, Hits<117.5}
R3 ={X | Years>=4.5, Hits>=117.5}

The predicted salaries for these three groups are:

\$1,000×$e^{5.107}$ = \$165,174
\$1,000×$e^{5.999}$ = \$402,834
\$1,000×$e^{6.740}$ = \$845,346

Terminology for Trees

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes*
- 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 are referred to as *internal nodes*
- Refer to the node segments of the trees that connect the nodes as branches.

Tree-building Process

Roughly speaking, there are two steps for the process of building a regression tree.

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

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.

Tree-building Process

- How do we construct the regions $R_1, R_2, \ldots, R_J$?
- 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 $R_1, R_2, \ldots, R_J$ 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.

Tree-building Process

- Unfortunately, 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.

Recursive Binary Splitting

- To perform recursive binary splitting, we first select the predictor X_j and the cut-point **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.
- That is, we consider all predictors $X_1, X_2, \ldots, X_p$, and **all possible** values of the cut-point **s** for each of the predictors, and then choose the predictor and cut-point such that the resulting tree has the lowest RSS.
- For any *j* and *s*, we define the pair of half-planes

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

and we seek the value of *j* and *s* that minimize the equation

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

where $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j, s)$,

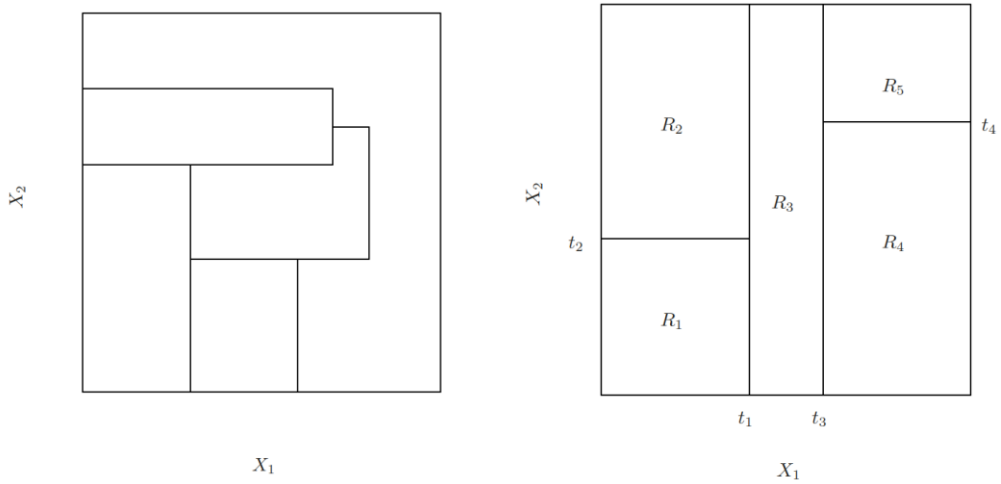
And $\hat{y}_{R_2}$ is the mean response for the training observations in $R_2(j, s)$.

Tree-building Process

- Next, we repeat the process, looking for the best predictor and best cut-point in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified 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.
- Once the regions $R_1, R_2, \ldots, R_J$ 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.

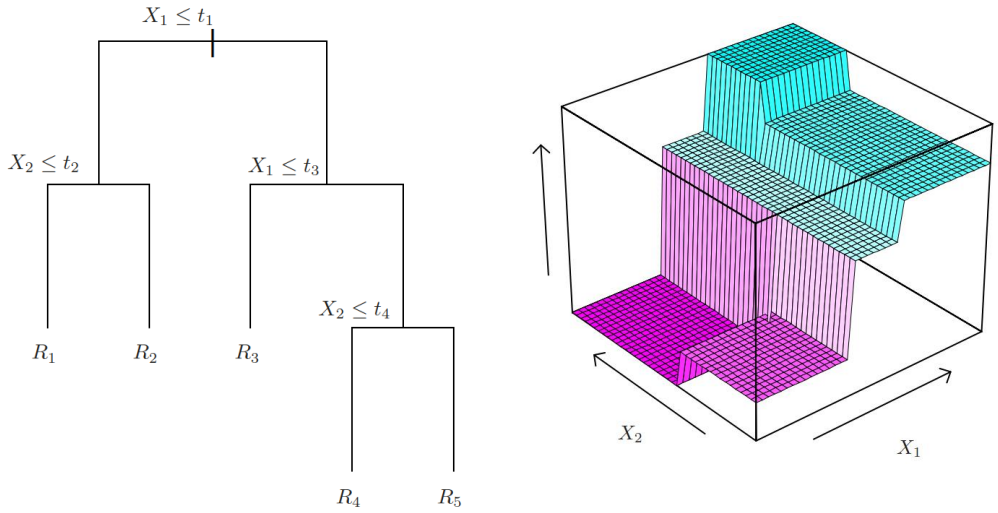
Example

- A five-region example of this approach is shown below.



- 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 corresponding to 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 set performance. This is because the resulting tree might be too complex.
- 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.
- A better strategy is to grow a very large tree T_0, and then prune it back to obtain a subtree.

Pruning a Tree

How do we determine the best prune way to prune the tree?

- Intuitively, our goal is to select a subtree that leads to the lowest **test error** rate.
- *Cost complexity pruning* — also known as *weakest link pruning* — is used to do this
- Rather than considering every possible subtree, 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

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

is as small as possible. Here |T| indicates the number of terminal nodes of the tree T, R_m is the rectangle (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.
- When $\alpha = 0$, then the subtree T will simply equal T_0. However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the objective function will tend to be minimized for a smaller subtree.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Regression Tree Algorithm

USC Viterbi School of Engineering

Algorithm 8.1 *Building a Regression Tree*

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 $k = 1, \ldots, K$:

   (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.

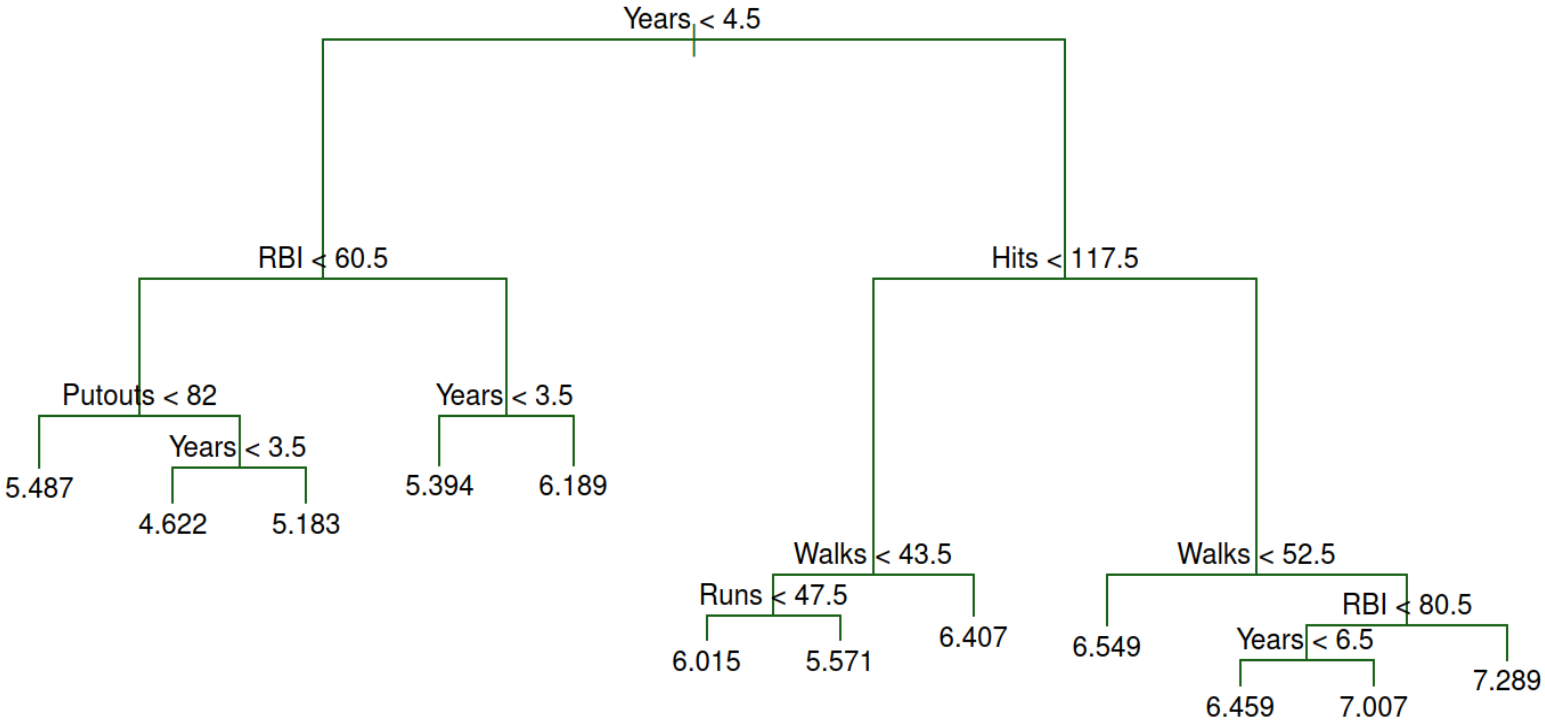
   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Baseball Example (cont'd)

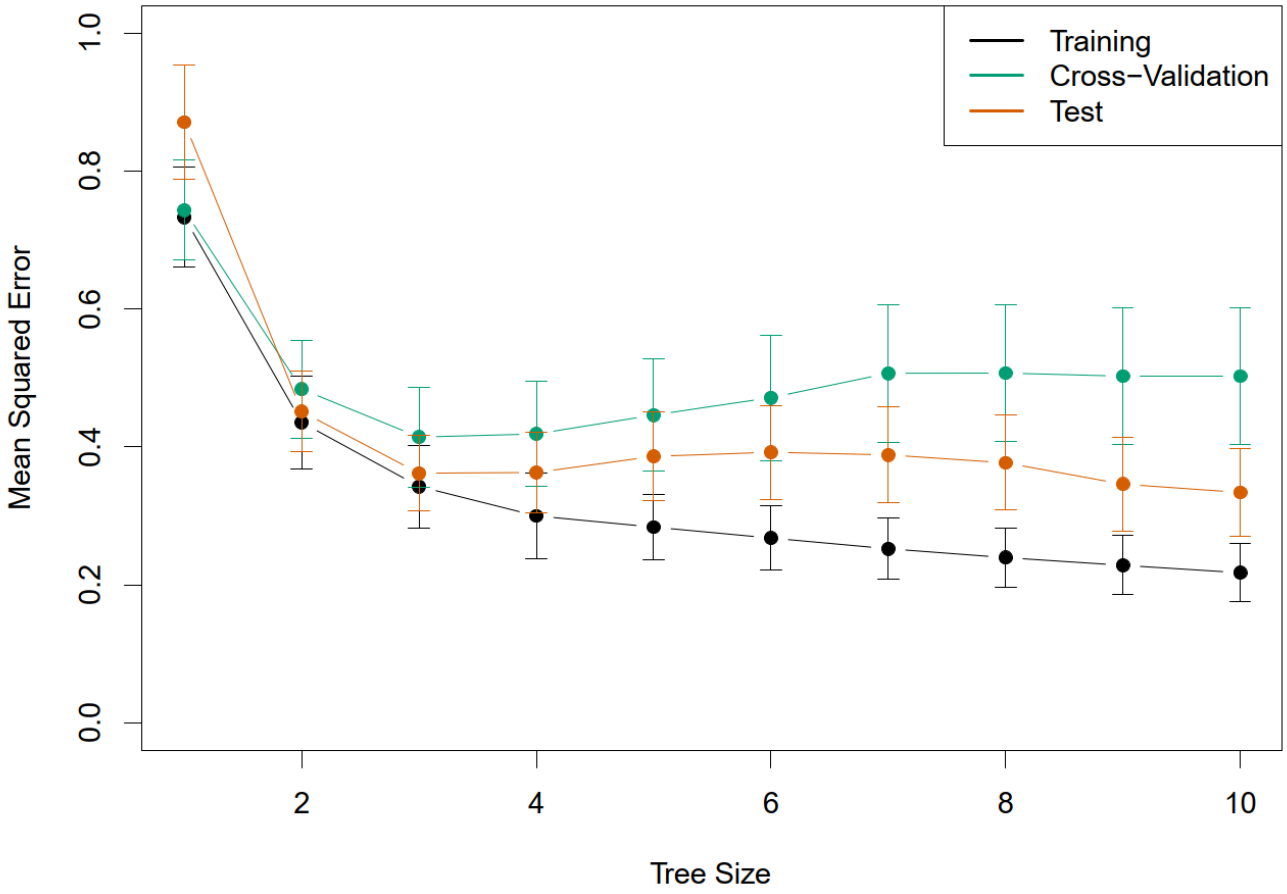
- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied α to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold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

Baseball Example (cont'd)



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

Baseball Example (cont'd)



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. The minimum cross-validation error occurs at a tree size of three.

CLASSIFICATION TREES

Classification Trees

- A classification tree is very similar to a regression tree, except that it is 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.
- 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.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits

Classification Trees

- A natural alternative to RSS is the *classification error rate*. This 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, classification error rate is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable, i.e., *Gini Index* or *Entropy*.

Gini Index & Cross-Entropy

- The Gini Index is defined by

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

a measure of *total variance across the K classes*. The Gini Index takes on a *small value* if all of the $\hat{p}_{mk}$ are close to zero or one.

- For this reason, 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.

Gini Index & Cross-Entropy

- An alternative to the Gini Index is *Cross-Entropy*, given by

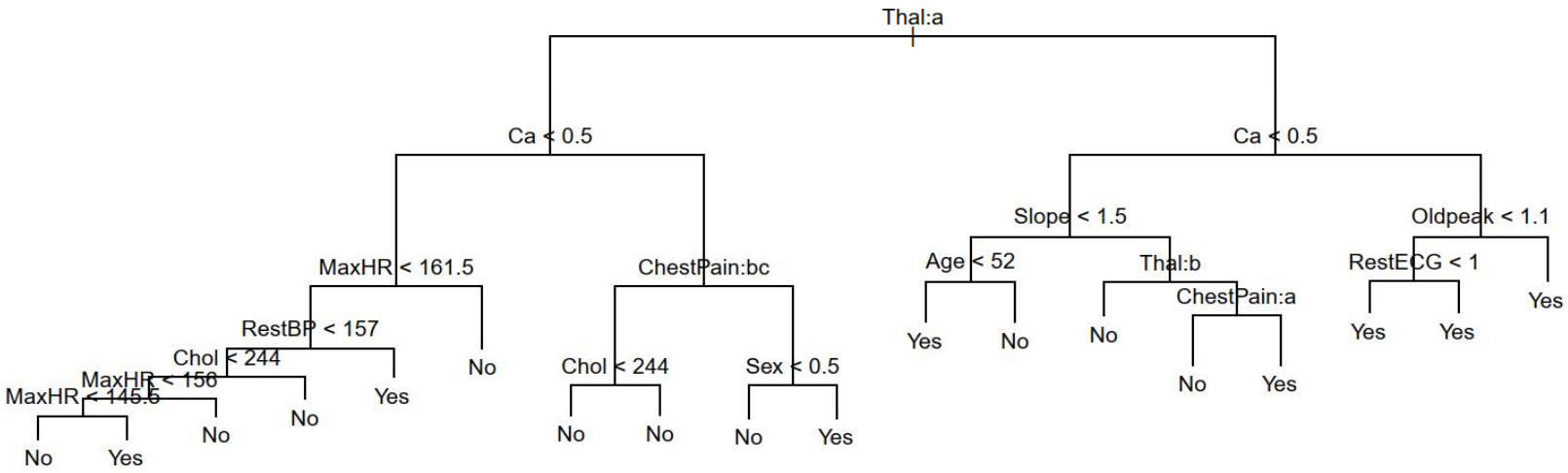
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

- Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $-\hat{p}_{mk} \log \hat{p}_{mk} \geq 0$.
- The entropy will take on a value near zero if the $\hat{p}_{mk}$ are all near zero or near one. Therefore, the entropy will take on a small value if the *m*th node is pure.
- It turns out that the Gini Index and the Cross-Entropy are very similar numerically.
- 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 sensitive to node **purity**.
- 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.

Example: Heart data set

- These data contain a binary outcome heart disease (HD) for 303 patients who presented with chest pain.
- An outcome value of *Yes* indicates the presence of 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 results in a tree with six terminal nodes. See next figure.

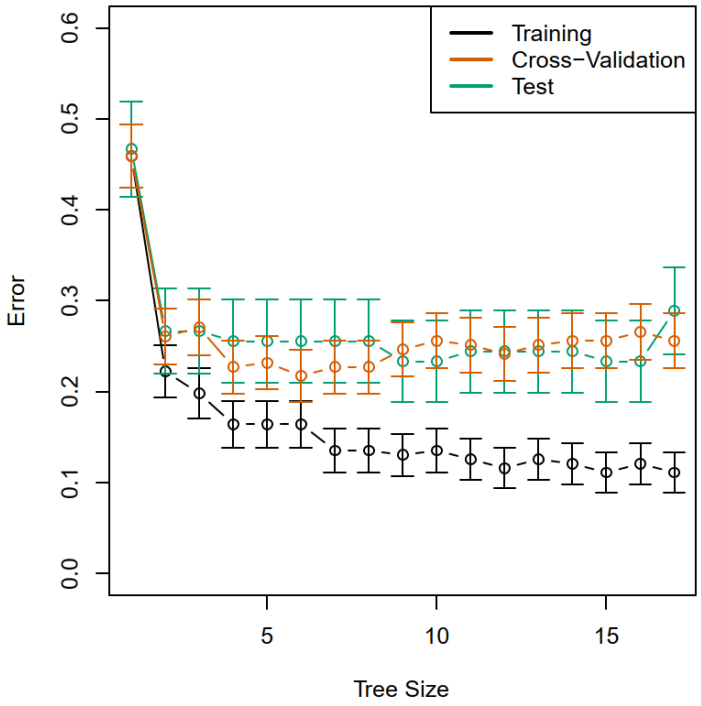
Example: Heart data set (cont'd)

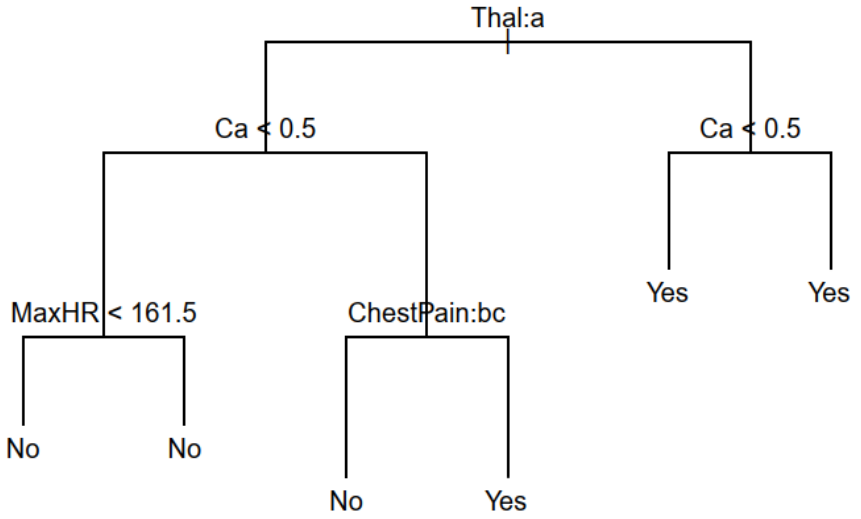


The figure shows the **unpruned tree**. Some of the predictors, such as *Sex*, *Thal* (Thallium stress test), and *ChestPain*, are qualitative. For instance, the left-hand branch come from the first value of the *Thal* variable (normal), and the right-hand branch from the remaining values (fixed or reversible defects) of the *Thal* variable .

A surprising characteristic: some of the splits yield two terminal nodes that have the same predicted value. The split is performed because it leads to increased *node purity*. For instance, consider the split RestECG < 1. That is, all 9 of the observations corresponding to the right-hand leaf have a response value of *Yes*, whereas 7/11 of those corresponding to the left-hand leaf have a response value of *Yes*. (**certain or less certain)**

Example: Heart data set (cont'd)





Left: Cross-validation error, training, and test error, for different sizes of the pruned tree.

Right: The pruned tree corresponding to the minimal cross-validation error.

Advantages and Disadvantages

- 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.
- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.

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