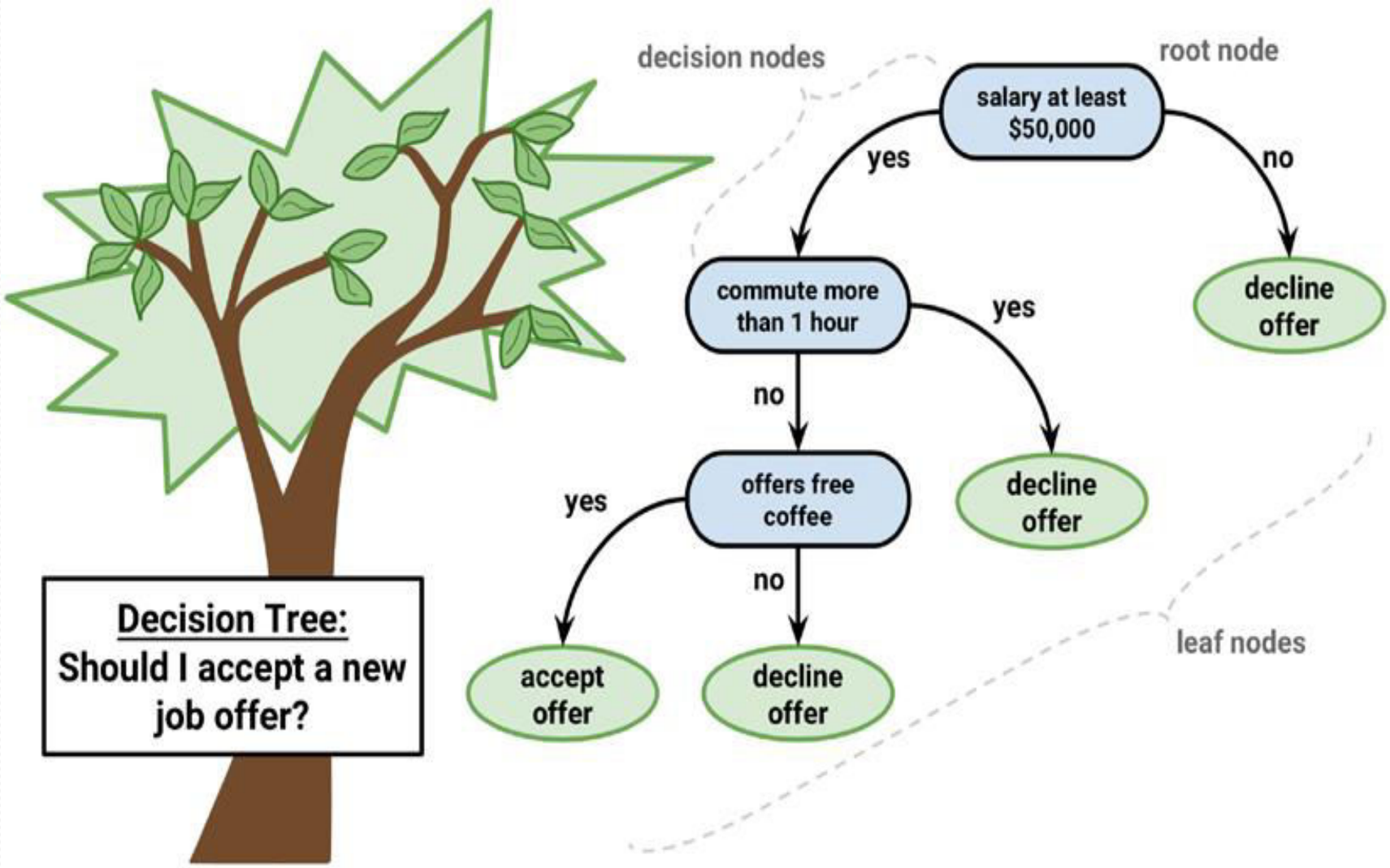
Divide and Conquer – Classification Using Decision Trees and Rules

Decision trees

- Decision tree learners are powerful classifiers, which utilize a **tree structure** to model the relationships among the features and the potential outcomes.
- Its name due to the fact that it mirrors how a literal tree begins at a wide trunk, which if followed upward, splits into narrower and narrower branches.
- In much the same way, a decision tree classifier uses a structure of branching decisions, which channel examples into a final predicted class value.

Decision trees



Decision trees

- Decision tree learners are powerful classifiers, which utilize a **tree structure** to model the relationships among the features and the potential outcomes.
- Its name due to the fact that it mirrors how a literal tree begins at a wide trunk, which if followed upward, splits into narrower and narrower branches.
- In much the same way, a decision tree classifier uses a structure of branching decisions, which channel examples into a final predicted class value.

Divide and conquer

- Decision trees are built using a heuristic called **recursive partitioning**.
- This approach is also commonly known as **divide and conquer** because it splits the data into subsets, which are then split repeatedly into even smaller subsets, and so on and so forth until the process stops when the algorithm determines the data within the subsets are sufficiently homogenous, or another stopping criterion has been met.

Divide and conquer

- Working down each branch, the algorithm continues to divide and conquer the data, choosing the best candidate feature each time to create another decision node, until a stopping criterion is reached.

Divide and conquer might stop at a node in a case that:

- All (or nearly all) of the examples at the node have the same class
- There are no remaining features to distinguish among the examples
- The tree has grown to a predefined size limit

The C5.0 decision tree algorithm

- This algorithm was developed by computer scientist J. Ross Quinlan as an improved version of his prior algorithm, **C4.5**, which itself is an improvement over his **Iterative Dichotomiser 3** (**ID3**)algorithm.
- The C5.0 algorithm has become the industry standard to produce decision trees, because it does well for most types of problems directly out of the box.

The C5.0 decision tree algorithm

Strengths	Weaknesses
• An all-purpose classifier that does well on most problems • Highly automatic learning process, which can handle numeric or nominal features, as well as missing data • Excludes unimportant features • Can be used on both small and large datasets • Results in a model that can be interpreted without a mathematical background (for relatively small trees) • More efficient than other complex models.	• Decision tree models are often biased toward splits on features having a large number of levels • It is easy to overfit or underfit the model • Can have trouble modeling some relationships due to reliance on axis-parallel splits • Small changes in the training data can result in large changes to decision logic • Large trees can be difficult to interpret and the decisions they make may seem counterintuitive

Choosing the best split

- The first challenge that a decision tree will face is to identify which feature to split upon.
- In the previous example, we looked for a way to split the data such that the resulting partitions contained examples primarily of a single class.
- The degree to which a subset of examples contains only a single class is known as **purity**, and any subset composed of only a single class is called **pure**.

Choosing the best split

- There are various measurements of purity that can be used to identify the best decision tree splitting candidate.
- C5.0 uses **entropy**, a concept borrowed from information theory that quantifies the randomness, or disorder, within a set of class values.
- Sets with high entropy are very diverse and provide little information about other items that may also belong in the set, as there is no apparent commonality.

Choosing the best split

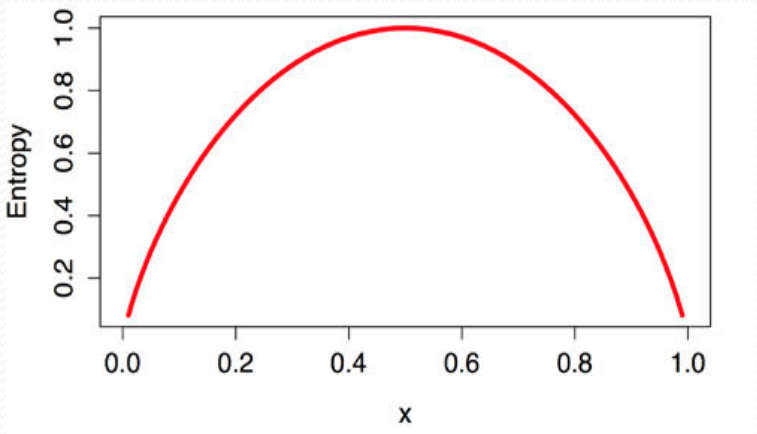
- The decision tree hopes to find splits that reduce entropy, ultimately increasing homogeneity within the groups. Typically, entropy is measured in **bits**.
- If there are only two possible classes, entropy values can range from 0 to 1.
- For *n* classes, entropy ranges from 0 to *log2(n)*. In each case, the minimum value indicates that the sample is completely homogenous, whilet he maximum value indicates that the data are as diverse as possible, and no group has even a small plurality.

Choosing the best split

- In the mathematical notion, entropy is specified as follows:
- Entropy (S) $= \sum_{i=1}^{c} -p_i \log_2(p_i)$
- In this formula, for a given segment of data *(S)*, the term *c* refers to the number of class levels and *pi* refers to the proportion of values falling into class level *i*. For example, suppose we have a partition of data with two classes: red (60 percent) andwhite (40 percent).

Choosing the best split

- he peak in entropy at $x = 0.50$, a 50-50 split results in maximum entropy. As one class increasingly dominates the other, the entropy reduces to zero.



Choosing the best split

- To use entropy to determine the optimal feature to split upon, the algorithm calculates the change in homogeneity that would result from a split on each possible feature, which is a measure known as **information gain**.
- The information gain for a feature *F* is calculated as the difference between the entropy in the segment before the split *(S1)* and the partitions resulting from the split *(S2)*:

$$\text{InfoGain}(F) = \text{Entropy}(S1) - \text{Entropy}(S2)$$

Choosing the best split

- One complication is that after a split, the data is divided into more than one partition. Therefore, the function to calculate *Entropy(S2)* needs to consider the total entropy across all of the partitions.
- It does this by weighing each partition's entropy by the proportion of records falling into the partition. This can be stated in a formula as:

$$\text{Entropy (S)} = \sum_{i=1}^{n} -w_i \text{ Entropy}(P_i)$$

Choosing the best split

- In simple terms, the total entropy resulting from a split is the sum of the entropy of each of the *n* partitions weighted by the proportion of examples falling in the partition (*wi*).
- The higher the information gain, the better a feature is at creating homogeneous groups after a split on this feature.
- If the information gain is zero, there is no reduction in entropy for splitting on this feature.

Choosing the best split

- On the other hand, the maximum information gain is equal to the entropy prior to the split.
- This would simply that the entropy after the split is zero, which means that the split results in completely homogeneous groups.

- Information gain is not the only splitting criterion that can be used to build decision trees.
- Other commonly used criteria are **Gini index**, **Chi-Squared statistic**, and **gain ratio**

Pruning the decision tree

- A decision tree can continue to grow indefinitely, choosing splitting features and dividing the data into smaller and smaller partitions until each example is perfectly classified or the algorithm runs out of features to split on.
- However, if the tree grows overly large, many of the decisions it makes will be overly specific and the model will be overfitted to the training data.
- The process of **pruning** a decision tree involves reducing its size such that it generalizes better to unseen data.

Pruning the decision tree

- One solution to this problem is to stop the tree from growing once it reaches a certain number of decisions or when the decision nodes contain only a small number of examples.
- This is called **early stopping** or **pre-pruning** the decision tree.

Pruning the decision tree

- An alternative, called **post-pruning**, involves growing a tree that is intentionally too large and pruning leaf nodes to reduce the size of the tree to a more appropriate level.
- This is often a more effective approach than pre-pruning, because it is quite difficult to determine the optimal depth of a decision tree without growing it first.
- Pruning the tree later on allows the algorithm to be certain that all the important data structures were discovered.

Pruning the decision tree

- One of the benefits of the C5.0 algorithm is that it is opinionated about pruning— it takes care of many decisions automatically using fairly reasonable defaults.
- Its overall strategy is to post-prune the tree.
- It first grows a large tree that overfits the training data. Later, the nodes and branches that have little effect on the classification errors are removed.

Pruning the decision tree

- In some cases, entire branches are moved further up the tree or replaced by simpler decisions.
- These processes of grafting branches are known as **subtree raising** and **subtree replacement**, respectively.
- Balancing overfitting and underfitting a decision tree is a bit of an art, but if model accuracy is vital, it may be worth investing some time with various pruning options to see if it improves the performance on test data. As you will soon see, one of the strengths of the C5.0 algorithm is that it is very easy to adjust the training options.

Boosting the accuracy of decision trees

- This is a process in which many decision trees are built and the trees vote on the best class for each example.
- By combining a number of weak performing learners, you can create a team that is much stronger than any of the learners alone.
- Each of the models has a unique set of strengths and weaknesses and they may be better or worse in solving certain problems.
- Using a combination of several learners with complementary strengths and weaknesses can therefore dramatically improve the accuracy of a classifier.

- Decision trees and rule learners are known as **greedy learners** because they use data on a first-come, first-served basis.
- Both the divide and conquer heuristic used by decision trees and the separate and conquer heuristic used by rule learners attempt to make partitions one at a time, finding the most homogeneous partition first, followed by the next best, and so on, until all examples have been classified.
-

- The downside to the greedy approach is that greedy algorithms are not guaranteed to generate the optimal, most accurate, or smallest number of rules for a particular dataset. By taking the low-hanging fruit early, a greedy learner may quickly find a single rule that is accurate for one subset of data; however, in doing so, the learner may miss the opportunity to develop a more nuanced set of rules with better overall accuracy on the entire set of data. However, without using the greedy approach to rule learning, it is likely that for all but the smallest of datasets, rule learning would be computationally infeasible.

Differences

- once divide and conquer splits on a feature, the partitions created by the split may not be re-conquered, only further subdivided.
- In this way, a tree is permanently limited by its history of past decisions. In contrast, once separate and conquer finds a rule, any examples not covered by all of the rule's conditions maybe re-conquered.