

Estimating Class Probabilities

A Decision Tree can also estimate the probability that an instance belongs to a particular class k : first it traverses the tree to find the leaf node for this instance, and then it returns the ratio of training instances of class k in this node. For example, suppose you have found a flower whose petals are 5 cm long and 1.5 cm wide. The corresponding leaf node is the depth-2 left node, so the Decision Tree should output the following probabilities: 0% for Iris-Setosa (0/54), 90.7% for Iris-Versicolor (49/54), and 9.3% for Iris-Virginica (5/54). And of course if you ask it to predict the class, it should output Iris-Versicolor (class 1) since it has the highest probability. Let's check this:

```
>>> tree_clf.predict_proba([[5, 1.5]])
array([[ 0. ,  0.90740741,  0.09259259]])
>>> tree_clf.predict([[5, 1.5]])
array([1])
```

Perfect! Notice that the estimated probabilities would be identical anywhere else in the bottom-right rectangle of [Figure 6-2](#)—for example, if the petals were 6 cm long and 1.5 cm wide (even though it seems obvious that it would most likely be an Iris-Virginica in this case).

The CART Training Algorithm

Scikit-Learn uses the *Classification And Regression Tree* (CART) algorithm to train Decision Trees (also called “growing” trees). The idea is really quite simple: the algorithm first splits the training set in two subsets using a single feature k and a threshold t_k (e.g., “petal length ≤ 2.45 cm”). How does it choose k and t_k ? It searches for the pair (k, t_k) that produces the purest subsets (weighted by their size). The cost function that the algorithm tries to minimize is given by [Equation 6-2](#).

Equation 6-2. CART cost function for classification

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

where $\begin{cases} G_{\text{left/right}} & \text{measures the impurity of the left/right subset,} \\ m_{\text{left/right}} & \text{is the number of instances in the left/right subset.} \end{cases}$

Once it has successfully split the training set in two, it splits the subsets using the same logic, then the sub-subsets and so on, recursively. It stops recursing once it reaches the maximum depth (defined by the `max_depth` hyperparameter), or if it cannot find a split that will reduce impurity. A few other hyperparameters (described in a

Download from finelybook www.finelybook.com
moment) control additional stopping conditions (`min_samples_split`, `min_samples_leaf`, `min_weight_fraction_leaf`, and `max_leaf_nodes`).



As you can see, the CART algorithm is a *greedy algorithm*: it greedily searches for an optimum split at the top level, then repeats the process at each level. It does not check whether or not the split will lead to the lowest possible impurity several levels down. A greedy algorithm often produces a reasonably good solution, but it is not guaranteed to be the optimal solution.

Unfortunately, finding the optimal tree is known to be an *NP-Complete* problem:² it requires $O(\exp(m))$ time, making the problem intractable even for fairly small training sets. This is why we must settle for a “reasonably good” solution.

Computational Complexity

Making predictions requires traversing the Decision Tree from the root to a leaf. Decision Trees are generally approximately balanced, so traversing the Decision Tree requires going through roughly $O(\log_2(m))$ nodes.³ Since each node only requires checking the value of one feature, the overall prediction complexity is just $O(\log_2(m))$, independent of the number of features. So predictions are very fast, even when dealing with large training sets.

However, the training algorithm compares all features (or less if `max_features` is set) on all samples at each node. This results in a training complexity of $O(n \times m \log(m))$. For small training sets (less than a few thousand instances), Scikit-Learn can speed up training by presorting the data (set `presort=True`), but this slows down training considerably for larger training sets.

Gini Impurity or Entropy?

By default, the Gini impurity measure is used, but you can select the *entropy* impurity measure instead by setting the `criterion` hyperparameter to “entropy”. The concept of entropy originated in thermodynamics as a measure of molecular disorder: entropy approaches zero when molecules are still and well ordered. It later spread to a wide variety of domains, including Shannon’s *information theory*, where it measures

² P is the set of problems that can be solved in polynomial time. NP is the set of problems whose solutions can be verified in polynomial time. An NP-Hard problem is a problem to which any NP problem can be reduced in polynomial time. An NP-Complete problem is both NP and NP-Hard. A major open mathematical question is whether or not $P = NP$. If $P \neq NP$ (which seems likely), then no polynomial algorithm will ever be found for any NP-Complete problem (except perhaps on a quantum computer).

³ \log_2 is the binary logarithm. It is equal to $\log_2(m) = \log(m) / \log(2)$.