**CART: Classification and Regression Trees**

# ****What category of algorithms does CART belong to?****

As the name suggests, CART (Classification and Regression Trees) can be used for both classification and regression problems. The difference lies in the target variable:

* With **classification**, we attempt to predict a class label. In other words, classification is used for problems where the output (target variable) takes a finite set of values, e.g., whether it will rain tomorrow or not.
* Meanwhile, **regression** is used to predict a numerical label. This means your output can take an infinite set of values, e.g., a house price.

# ****How do classification and regression trees work?****

## **Example**

Let’s start with a simple example. Assume you have a bunch of oranges and mandrins with labels on them, and you want to identify a set of simple rules that you can use in the future to distinguish between these two types of fruit.

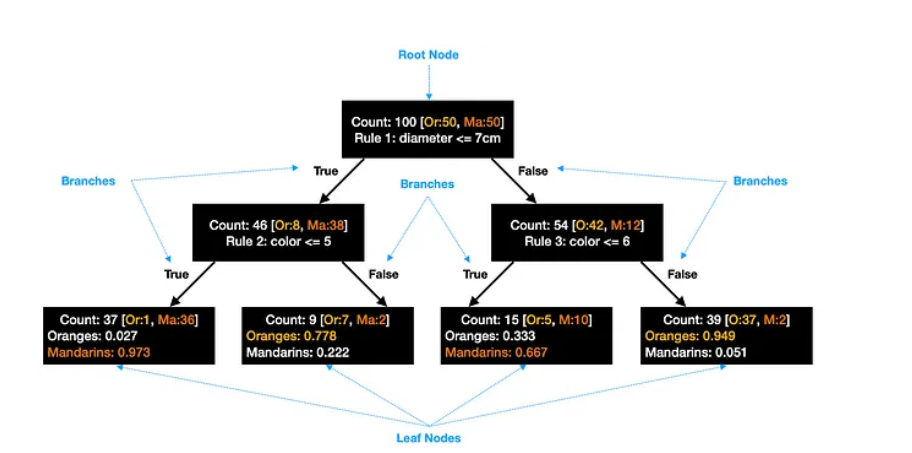
Typically, oranges (diameter 6–10cm) are bigger than mandarins (diameter 4–8cm), so the first rule found by your algorithm might be based on size:

* Diameter ≤ 7cm.

Next, you may notice that mandarins tend to be slightly darker in color than oranges. So, you use a color scale (1=dark to 10=light) to split your tree further:

* Color ≤5 for the left side of the sub-tree
* Color ≤6 for the right side of the sub-tree

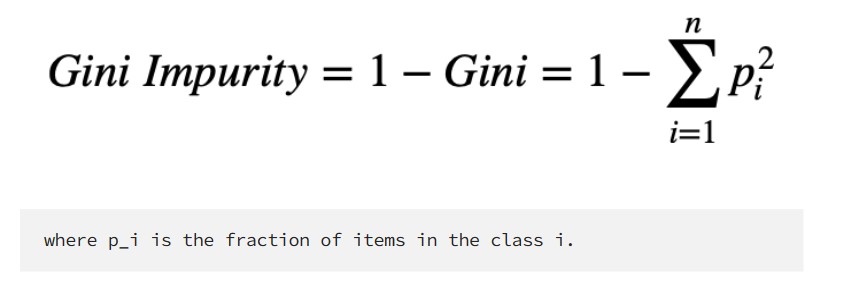
Your final result is a tree that consists of 3 simple rules that help you to correctly distinguish between oranges and mandarins in the majority of the cases:

****

## **How does CART find the best split?**

Several methods can be used in CART to identify the best splits. Here are two of the most common ones for classification trees:

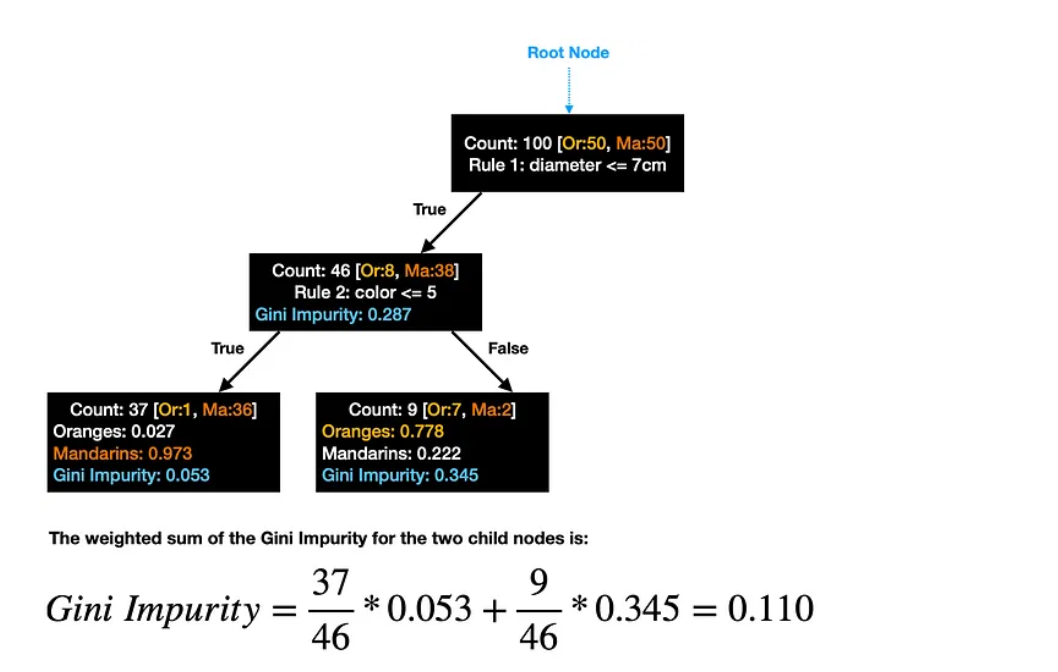
## Gini Impurity

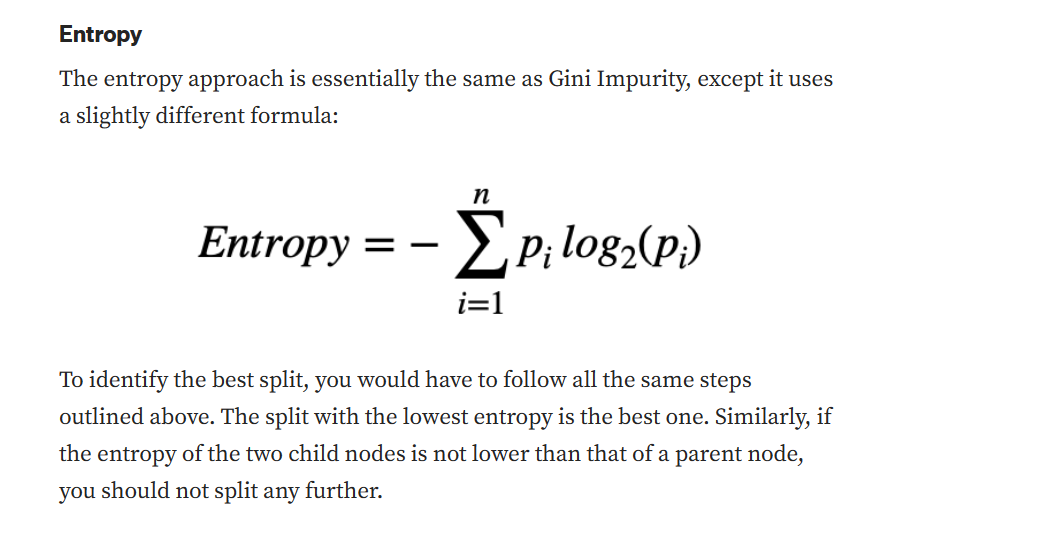
****

Using the above tree as an example, Gini Impurity for the leftmost leaf node would be:

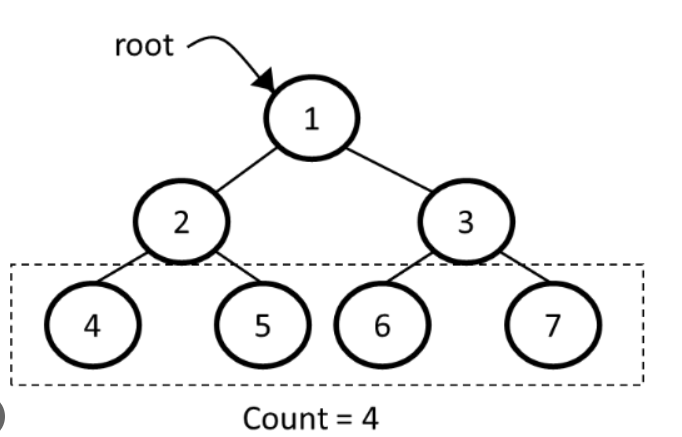
1 - (0.027^2 + 0.973^2) = 0.053

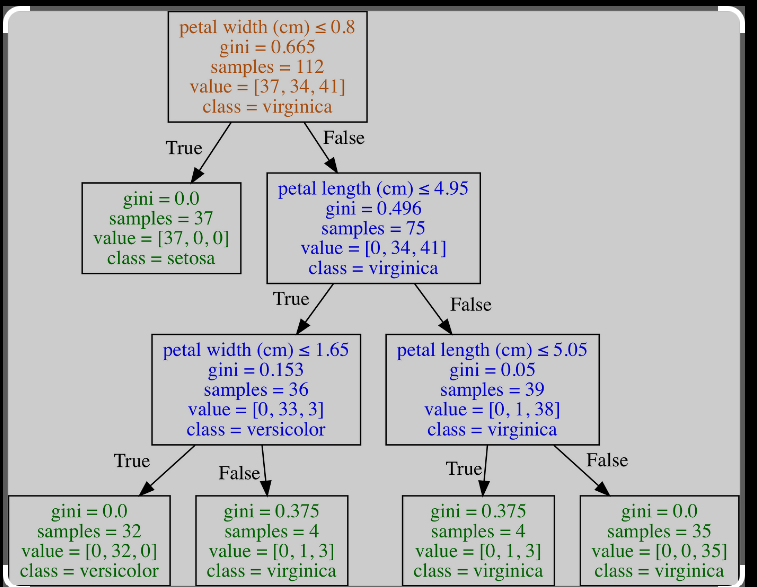
To find the best split, we need to calculate the weighted sum of Gini Impurity for both child nodes. We do this for all possible splits and then take the one with the **lowest Gini Impurity** as the best split.

****

****

**Number of leaf nodes in a tree:**

****

****

As you can see category chosen is on the leaf nodes by the majority of samples in certain category the gini impurity is used in order to check the depth of the three even if you type a maximum depth the model could cut off earlier **whether the best weighted Gini Impurity for the two child nodes is not lower than Gini Impurity for the parent node, you should not split the parent node any further.**

Using Sklearn to create CAST you do not have to put the rules manually, sklearn will do it choosing the best rules and features however there are many parameters that you can tune while you create you model:

**For Decision Tree Classifier:**

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

**Optimizing Decision Tree Performance criterion**: optional (default=”gini”) or Choose attribute selection measure. This parameter allows us to use the different attribute selection measure. Supported criteria are “gini” for the Gini index and “entropy” for the information gain.

**Splitter:** string, optional (default=”best”) or Split Strategy. This parameter allows us to choose the split strategy. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

**max\_depth:** int or None, optional (default=None) or Maximum Depth of a Tree. The maximum depth of the tree. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

**min\_samples\_leaf:** The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches.

**min\_weight\_fraction\_leaf**, the minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample\_weight is not provided.

**class\_weightdict, list of dict or “balanced”:** Weights associated with classes in the form {class\_label: weight}. If None, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y))

**For Decision Tree Regressor:**

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor

Parameters:

**Criterion** {“squared\_error”, “friedman\_mse”, “absolute\_error”, “poisson”}, default=”squared\_error”

**Splitter**{“best”, “random”}, default=”best”

**Max\_depth**: *int, default=None*

**Min\_samples\_split: int or float, default=2:** The minimum number of samples required to split an internal node.

**min\_samples\_leaf: int or float, default=1:** The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches.

**min\_weight\_fraction\_leaf:float, default=0.0:** The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample\_weight is not provided.

**max\_featuresint, float or {“auto”, “sqrt”, “log2”}, default=None:** The number of features to consider when looking for the best split.

**max\_leaf\_nodesint, default=None:** Grow a tree with max\_leaf\_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.