**Decision Trees:**

**DTs** are a non-parametric supervised learning method used for [classification](https://scikit-learn.org/stable/modules/tree.html#tree-classification) and [regression](https://scikit-learn.org/stable/modules/tree.html#tree-regression). The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Some advantages of decision trees are:

* Simple to understand and to interpret. Trees can be visualized.
* Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
* The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
* Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See [algorithms](https://scikit-learn.org/stable/modules/tree.html#tree-algorithms) for more information.
* Able to handle multi-output problems.
* Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
* Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
* Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

* Decision-tree learners can create over-complex trees that do not generalize the data well. This is called over fitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
* Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
* The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
* There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
* Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

## Tips on practical use

* Decision trees tend to overfit on data with a large number of features. Getting the right ratio of samples to number of features is important, since a tree with few samples in high dimensional space is very likely to overfit.
* Consider performing dimensionality reduction ([PCA](https://scikit-learn.org/stable/modules/decomposition.html#pca), [ICA](https://scikit-learn.org/stable/modules/decomposition.html#ica), or [Feature selection](https://scikit-learn.org/stable/modules/feature_selection.html#feature-selection)) beforehand to give your tree a better chance of finding features that are discriminative.
* [Understanding the decision tree structure](https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html#sphx-glr-auto-examples-tree-plot-unveil-tree-structure-py) will help in gaining more insights about how the decision tree makes predictions, which is important for understanding the important features in the data.
* Visualise your tree as you are training by using the export function. Use max\_depth=3 as an initial tree depth to get a feel for how the tree is fitting to your data, and then increase the depth.
* Remember that the number of samples required to populate the tree doubles for each additional level the tree grows to. Use max\_depth to control the size of the tree to prevent overfitting.
* Use min\_samples\_split or min\_samples\_leaf to ensure that multiple samples inform every decision in the tree, by controlling which splits will be considered. A very small number will usually mean the tree will overfit, whereas a large number will prevent the tree from learning the data. Try min\_samples\_leaf=5 as an initial value. If the sample size varies greatly, a float number can be used as percentage in these two parameters. While min\_samples\_split can create arbitrarily small leaves, min\_samples\_leaf guarantees that each leaf has a minimum size, avoiding low-variance, over-fit leaf nodes in regression problems. For classification with few classes, min\_samples\_leaf=1 is often the best choice.
* Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant. Class balancing can be done by sampling an equal number of samples from each class, or preferably by normalizing the sum of the sample weights (sample\_weight) for each class to the same value. Also note that weight-based pre-pruning criteria, such as min\_weight\_fraction\_leaf, will then be less biased toward dominant classes than criteria that are not aware of the sample weights, like min\_samples\_leaf.
* If the samples are weighted, it will be easier to optimize the tree structure using weight-based pre-pruning criterion such as min\_weight\_fraction\_leaf, which ensure that leaf nodes contain at least a fraction of the overall sum of the sample weights.
* All decision trees use np.float32 arrays internally. If training data is not in this format, a copy of the dataset will be made.
* If the input matrix X is very sparse, it is recommended to convert to sparse csc\_matrix before calling fit and sparse csr\_matrix before calling predict. Training time can be orders of magnitude faster for a sparse matrix input compared to a dense matrix when features have zero values in most of the samples.

## Tree algorithms: ID3, C4.5, C5.0 and CART

What are all the various decision tree algorithms and how do they differ from each other? Which one is implemented in scikit-learn?

[ID3](https://en.wikipedia.org/wiki/ID3_algorithm) (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data.

C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. These accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule’s precondition if the accuracy of the rule improves without it.

C5.0 is Quinlan’s latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate.

[CART](https://en.wikipedia.org/wiki/Predictive_analytics#Classification_and_regression_trees_.28CART.29) (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

scikit-learn uses an optimised version of the CART algorithm; however, scikit-learn implementation does not support categorical variables for now.