Lecturer notes\_Sesi 4 \_ Decision Tree

**https://towardsdatascience.com/build-better-decision-trees-with-pruning-8f467e73b107**

**Pruning**

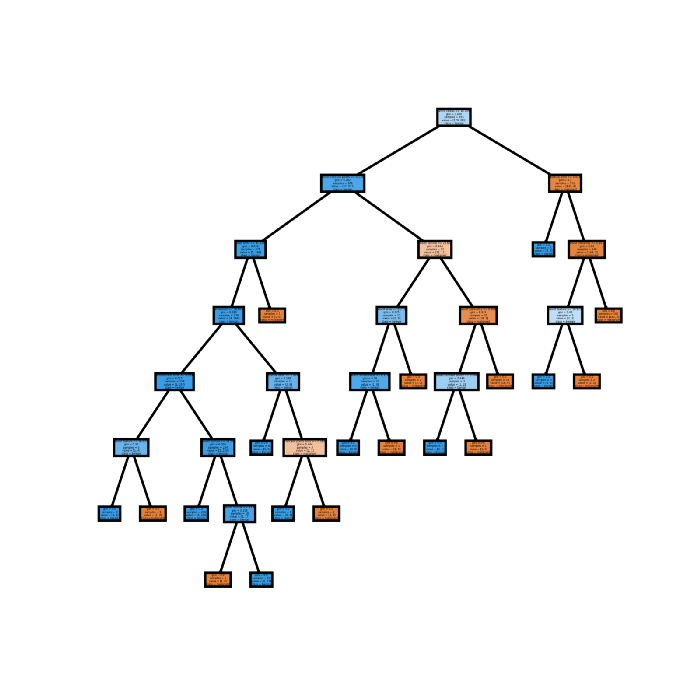
Overfitting and Decision Trees

Decision Trees are prone to over-fitting. A decision tree will always overfit the training data if we allow it to grow to its max depth.

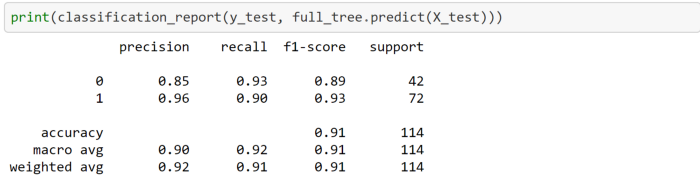
A decision tree is overfit when the tree is trained to fit all samples in the training data set perfectly. You can tweak some parameters such as [min\_samples\_leaf](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)to minimize default overfitting. This type of tweak is called pre-pruning, but is out of the scope of this article.

The deeper you allow your tree to grow, the more complex the sequence of decision rules becomes. Assigning a maximum depth to a tree can simplify it and combat overfitting.





Below is the classification report for this tree when it encounters the **test data**.



Notice the accuracy, 91. Keep this in mind as we build our two next trees.

**How to Simplify a Decision Tree with an Optimal Maximum Depth**

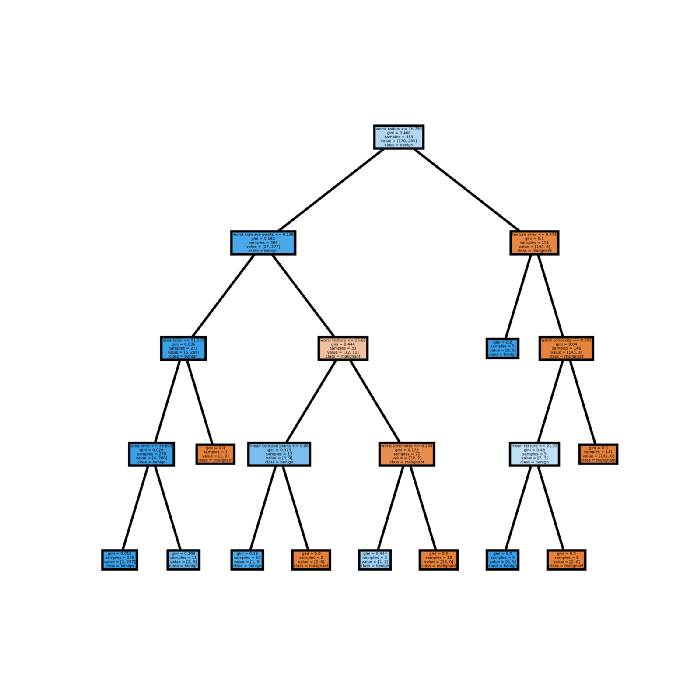
Now let's build a tree and limit its maximum depth.



In the first cells above, we find the depth of our full tree and save it as max\_depth. We do this to build a grid search from 1 → max\_depth.

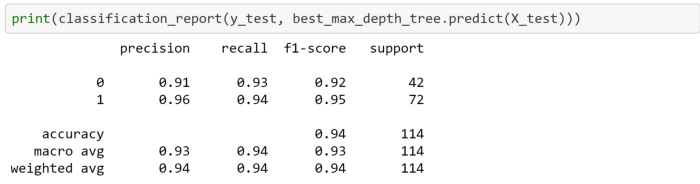
This grid search builds trees of depth range 1 → 7 and compares the training accuracy of each tree to find the depth that produces the highest training accuracy.

The most accurate tree has a depth of 4, shown in the plot below.



This tree has 10 rules. This means it is a simpler model than the full tree.

How does it perform on the test data?



Look at that! Fewer rules and higher accuracy. We definitely built a better model when we limited our max depth.

**What is Pruning?**

**Pruning** is a technique that is used to reduce overfitting. Pruning also simplifies a decision tree by removing the weakest rules. Pruning is often distinguished into:

* **Pre-pruning** (early stopping) stops the tree before it has completed classifying the training set,
* **Post-pruning** allows the tree to classify the training set perfectly and then prunes the tree.

We will focus on **post-pruning** in this article.

Pruning starts with an unpruned tree, takes a sequence of subtrees (pruned trees), and picks the best one through cross-validation.

Pruning should ensure the following:

* The subtree is optimal — meaning it has the highest accuracy on the cross-validated training set. (Trees can be optimized for whatever parameter is most important to the engineer — not always accuracy)
* The search for the optimal subtree should be computationally tractable.

In scikit-learnsDecisionTreeClassifier, ccp\_alphaIs the **cost-complexity parameter.**

Essentially, pruning recursively finds the node with the “weakest link.” The weakest link is characterized by an effective alpha, where the nodes with the smallest effective alpha are pruned first.

Mathematically, the **cost complexity measure** for a tree T is given by:



* **R(T)**— Total training error of leaf nodes
* **|T|** — The number of leaf nodes
* **α**— complexity parameter(a whole number)

As alpha increases, more of the tree is pruned, which increases the total impurity of its leaves.

If we only try to reduce the training error **R(T)**, it will lead to relatively larger trees (more leaf nodes), resulting in overfitting.

Cost complexity pruning generates a series of trees where cost complexity measure for sub-tree **Tₜ**is:



The parameter **α**reduces the complexity of the tree by controlling the number of leaf nodes, which eventually reduces over-fitting.

Which subtree is selected eventually depends on α . If **α=0**, then the biggest tree will be chosen because the complexity penalty term is essentially dropped. As α approaches infinity, the tree of size 1, i.e., a single root node, will be selected.

To get an idea of what values of ccp\_alpha will work to reduce the tree size, scikit-learn provides a functioncost\_complexity\_pruning\_path that returns the effective alphas and the corresponding total leaf impurities at each step of the pruning process.

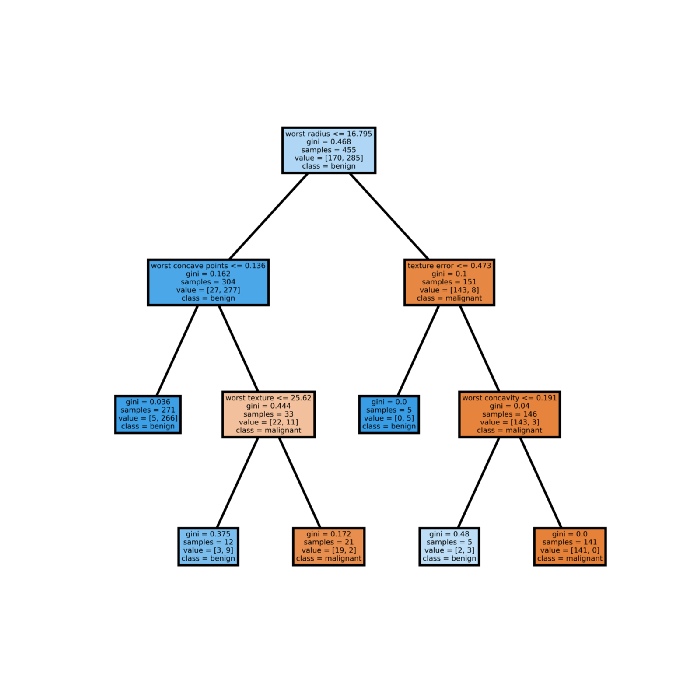
Let’s build our final tree model and see how it performs.



Each ccp\_alpha above represents an optimal subtree. Once again, we build a grid search to compare the various trees. Here the grid search is comparing the training accuracy for each optimal subtree.

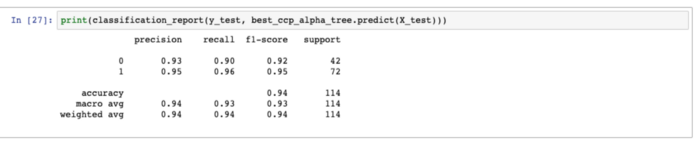
We see that the most accurate subtree is that generated by the ccp\_alpha, 0.0059340658…

Now let’s plot this pruned tree.



This model only contains 5 rules!

The pruned model is less complex, easier to explain and easier to understand than the previous decision tree plots.



With half the rules of the max-depth-limited tree, we have achieved the same accuracy. Once again, we have improved our model! This time we did so by reducing complexity while maintaining performance.

**Advantages of Pruning a Decision Tree**

* Pruning reduces the complexity of the final tree and thereby reduces overfitting.
* Explainability — Pruned trees are shorter, simpler, and easier to explain.

**Limitations of Pruning**

Similar to Lasso regularization, there is no real disadvantage. However, pruning does come with a high computational cost.

**Resources**

* Scikit-learn documentation for **c**[**ost complexity pruning**](https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html#sphx-glr-auto-examples-tree-plot-cost-complexity-pruning-py)**.**
* PennState STAT 508 | Applied Data Mining and Statistical Learning <https://online.stat.psu.edu/stat508/lesson/11/11.8/11.8.2>