**Pruning Decision Trees**

**Pruning Regression Trees**

Now would like to give a little more thought to problem of overfitting our Trees to the data. So we used a general criterion that the number of data points in a leaf has to be greater than or equal to some value. He used 7 in this simple example. Often 20 is used as the threshold. This is to prevent overfitting the data. Nonetheless, it can be the case that data is still overfit. If our training data looked like this above, and our testing data looked like this:

Chart

Description automatically generated

Then it would seeem that we have overfit our training data. Might have been better to prune the last two leaves of our final Decision Tree,

Diagram

Description automatically generated

as that prior tree seems to fit the test data better,

Chart

Description automatically generated Diagram

Description automatically generated

So how do we know when to stop branching? One way is to take our overfit tree, DT(Lmax) with Lmax leaves, and then successively snip off branches to get a new DT(L), where L < L­max­. And then run the new decision tree through the usual cross validation techniques and compare its performance on the testing data to the original DT(Lmax) tree. And we’d just keep snipping as long as our DT(L)’s performance keeps getting better, i.e., either the SSE(L) keeps getting smaller, or the confusion matrix score CMS(L) = Tr(CM)/|CM| keeps getting larger, etc. Now I considered our decision tree as being parameterized by the number of terminal leaves, L, that it has. But there are different ways to prune a tree to the same number of leaves, L. A better way to parameterize the pruned tree is with the cost-complexity parameter, α. So we consider a decision tree with L leaves, calculate Loss(L), and define a Tree Score:



where *Loss* is some loss function associated with the tree, α is ‘some’ non-negative parameter complexity parameter, and L is the number of leaves in our tree. The loss function for a regression tree would likely be SSE, and for a classification tree is often the TER (Total Error Rate), but could also be the Gini Impurity, or Entropy. Then for values of α ranging from 0 to some αmax, we determine which pruning minimizes the tree score T(α,L). When α is 0, then no pruning would be done, as there is no cost associated with having a large number of leaf nodes, L. And so we’d just want to minimize the Loss as much as possible. For super large α > αmax, then we’d have only a root node, the cost of having more leaves woud be too high, even though it would certainly reduce Loss. For intermediate values of α, some pruning of the tree would be done to minimize the tree score. So α is just a parameter that can be put into a one-to-one (mostly?) correspondance with how much pruning has been done to the initial (overfit) tree. So we can sort of write our pruned tree as a function of α, i.e., DT\_pruned = DT(α), instead of DT(L). And DT(α=0) = overfit, unpruned Tree, and DT(α>αmax) = root/stump tree. And as we increase α from 0 to αmax, DT(α) will presumably run through every way we can prune the tree. Should also note that L will be roughly inversely proportional to α. Here’s a list of α’s I found in one decision tree analysis I did. These are the α’s where the DT changed its shape, i.e., was pruned in some way.

Text

Description automatically generated

And here I graphed L(α) for this decision tree,

Chart, line chart

Description automatically generated

Interesting that L doesn’t hit every integer between Lmax and 1. And it even goes up at one point. So just know that we can characterize the complexity of tree with the parameter α. FWIW, we’ll also illustrate that the gini impurity of our model goes up as we prune the decision tree. And of course this ought to be the case as we prune away leaves.

Chart, line chart

Description automatically generated

Then of all the models, we find the one with α = α\* for which SSE(α\*) [of the testing data] is smallest. For instance, I calculated the confusion matrix score (accuracy score) of the training data set and testing data set, as a function of α,

Chart, line chart

Description automatically generated

The accuracy of the model on the training data starts off at 1, as expected. As we prune away leaves, the accuracy goes down, as does the gini impurity score. But the accuracy of the model on the testing data goes up, since the model fits less noise as we prune away leaves. So it appears our α\* ~ (0.16 – 0.37). Not sure which, yet. Seems that in general, α\* will be on the minimum side of the range? So we’d guess α\* ~ 0.16. And then use the tree with α = α\*, i.e., DT(α\*).

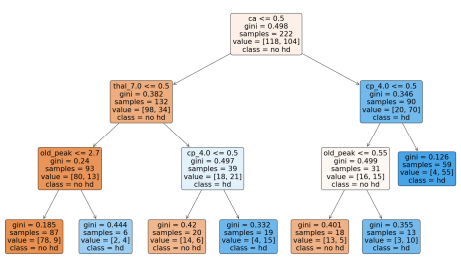
On a technical note, there is a slight problem with this procedure. We found the α\*, presumably, which best matched the testing data. But there are many ways we can split the data up into training and testing groups. And we will probably find a different α\* on a different training/testing set. To address this problem, we’ll use cross-validation. So below, I used

five-fold cross validation I think, and for each value of α, calculated the average CMS score across the five different test data sets.

Chart, line chart

Description automatically generated

And can see here the peak is at the α (in the list) given by α\* = 0.0142. So then to deploy, we use this value of α = α\* for the entire data set, meaning, we use DT(α\*) for the testing data. Here’s what the pruned decision tree looked like, for what it’s worth:



The boxes are more orange the further they lean towards “no heart disease” and more blue the further they learn towards “heart disease”. So the hue gets darker as gini impurity decreases. Note that samples = Σ values. Left value is number in that node that will ultimately be classified as “no hd”, and right value is number in that node that will ultimately be classified as “hd”. Note that at any node, samples = Σ daughter samples. Class is “no hd” if left value is greater than right value, and “hd” if left value is smaller than right value. BTW, the unpruned tree was *big*.

**Example**

Consider our classification tree above. Let’s say we wanted to keep pruning it, for illustration’s sake. The gini impurity of each leaf is given, and I put the probabilities of each branch on the respective lines.

A diagram of a data flow

Description automatically generated with medium confidence

Now let’s say we were to minimize the following tree score,



What is the tree corresponding to α = 0, i.e., DT(α=0)? Well for α = 0, L will be the unpruned tree. And FWIW, its gini impurity value is:



What is αmax? Well let’s consider starting with the root node. We have GIroot = 0.498. αmax is the value for which T(α,L) increases for all L. That means that each leaf we add, αL increases more than adding that leaf can decrease GI. Well the most that GI could possibly decrease is GIroot = 0.498. So if α > 0.498, then to minimize T(α,L) we’ll always want to make L as small as possible, i.e., L = 1. So we can say αmax = 0.498. And therefore DT(α>αmax) = just the root node. Now what about intermediate cases? For what minimum α will it behoove us to prune our tree? Well we can minimally increase GI by pruning one of those three leaf pairs. The GI gain (GIG) obtained by pruning the bottom leaf pairs is respectively (see the Appendix of the Decision Tree Classification file):



Up to this level of pruning, we can write T(α,L) as:



where θ(x) = 1 if x is True, and 0 otherwise. Now when α = 0, we can minimize T by setting all the θ’s (or their conditions) to 0. When α is positive but very small, the minimum value of T is still obtained by taking out zero pairs, so that:



But there is a lower bound for α beyond which we can get the smallest value of T by taking out a pair. The pair we’d take out is the left pair as this results in the smallest increase in T, 0.027, for excising a pair. And the reduction in T is given by 2α. This will lower T when we have 2α > 0.027 → α > 0.0135. So αmin = 0.0135. And without getting into it further, let’s just say that as α is increased, it will become favorable to excise more and more pairs.

**Exploring the Model and Hyperparameters**

So we left off, until now, one of the hyperparameters in the DecisionTreeClassifier. And that is alpha of course,

**Hyperparameter: alpha**

It’s called ccp\_alpha in sklearn. Here’s the N = 500 guy with 10% outliers, and varying values of ccp\_alpa. Can see that as it increases, the tree is pruned more and more.

A diagram of a graph

Description automatically generated A diagram of a graph

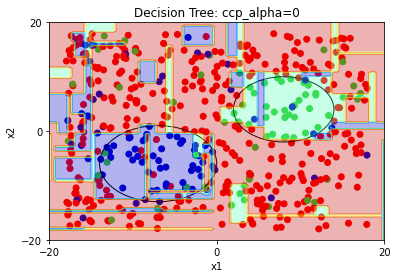
Description automatically generated

A diagram of a tree

Description automatically generated A diagram of a tree

Description automatically generated

There is clearly an optimum values somewhere around ccp\_alpha = 0.005 or something. Tuning this guy seems to be worth the effort. Let’s do same on tri-class circles guy,

 A diagram of a tree

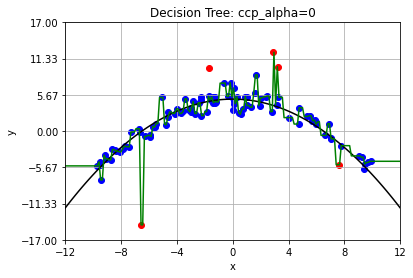
Description automatically generated

A diagram of a tree

Description automatically generated A diagram of a tree

Description automatically generated

Can see that small α overfits (i.e., picking up outliers), and large α is underfitting, i.e., not following the contours of the circular boundaries. We can also look at regression model,

 A graph with a line graph and numbers

Description automatically generated with medium confidence

A graph of a graph with a line graph

Description automatically generated with medium confidence A graph with red and blue dots

Description automatically generated

Again, there is a nice middle ground to be had, but still hard to keep it from doing the outliers.