**Decision Trees Regression**

Now we’d like to consider how to construct a Decision Tree when the prediction we’re making is not categorical, but numerical, as is this case below.

Table

Description automatically generated

**SSE, Prediced Outcome, and Information Gain**

Much is the same as before, but we will need a new statistic, analogous to entropy and information gain, or impurity and impurity loss, which handles numerical data, instead of categoricals. Apparently we use the sum of squared errors about the mean.



where yi enumerates all the data, and is the mean of all the data. The predicted outcome of this set of data would be:



We will also define SSEm(Y|Ai), the sum of squared errors about the means, given A = Aa. This is just saying that we are restricting attention to the rows of data for which A = Aa. And this would be given by:



And out of this reduced set of outcomes, the predicted oucome would be:



And if we want (and we do) to calculate the SSEm measure of spread over all values of A, then we do:



In words, we are taking all output values, and putting them in subgroups sorted by their A values, Aa. The output values so sorted are referred to as yAa,j (Aa denotes the subgroup with comman value Aa, and j the element within that subgroup). Then we find the mean values within that subgroup, which I’m calling Aa, and computing the sum of squared errors within that subgroup. Then we add up the sum of squared errors across all subgroups. And finally, we could define an information gain kind of quantity,



And whichever column of data has the largest information gain would be the one most correlated with the outcome. Like in the previous file, we can run into the problem where a column has purely numerical values, and so doesn’t naturally segregate itself into a reasonable number of values upon which we can split a node in the regression tree. So all things being equal, do analogous to what we did in the last file. We split the data into two ‘values’: those less than a critical value, and those greater than a critical value. To find the possible critical values, we arrange the numerical data in order, and then calculate the average of adjacent pairs. Then we calculate SSEm(Y|A) for each critical value, choosing whichever has the smallest value.

**Constructing the Decision Tree**

So below I’ll describe the decision tree. So diagrammatically, we’ll start our decision tree by putting all the data in the root node. And then we’re looking to categorize the data into categories/leaves, that maximize the IG or whatever.

Diagram, schematic

Description automatically generated

So we want to find leaves, Lℓ, which maximize the information gain:



In this diagam, L = {A1, B1, B2}. We will typically use a greedy algorithm to find these leaves. It won’t be guaranteed to produce the correct result, but I don’t think we actually care, because we won’t usually fit a tree precisely to the data anyway (that’d be overfitting). The way we use the greedy algorithm is as follows. Say we’re at some node, A2 (and this could be the root of course). In this node are all rows with value A = A2. These rows have outcomes Yj. We can calculate the SSE of this node SSEm(Y|A2) = Σj=1n\_A2 (yA2\_j – fA2)2 as shown. And the prediction of this node would be fA2 = (1/nA2)Σj=1n\_A2 yA2\_j, i.e., the average value of Y in this subset A2. To make further progress classifying the data, we can split Yj’s into two groups according to the B values: B1, B2. And we can calculate the SSEm(Y|B1) of the group in B1, and the entropy SSE(Y|B2) of the group in B2. Summed together this gives the total SSE(Y|B) = SSE(Y|B1) + SSE(Y|B2). And then we calculate the information gain for this split: IG(B) = SSE(Y|B) – SSE(Y|A2). And we choose the column B for which IG(B) is largest. And we continue in like fashion until we’ve broken the tree down as far as we want to go. And certainly if we should find that the information gain were ever negative, then we wouldn’t bother making the split.

Note, BTW, that R, A2 are internal nodes, also called split nodes. And A1, B1, B2 are called terminal, or leaf nodes.

The depth of a decision tree is the length of the longest path from the root to a leaf node. So the depth of the tree above would be 2.

**Caution**

Like we mentioned in the previous file, we don’t want to overfit the data. So we generally won’t split a node which has fewer than 20 data points. Or maybe we’ll require the information gain to be past some threshold.

**Example**

Let’s try this example.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

The average effectiveness is 47, and the SSEm is:



So our root is:

A picture containing graphical user interface

Description automatically generated

Moving on to the,

**First Level**

So first we have to compute SSEm(E|D). I’ll limit the size of the branch to 2. So let Dcrit = (10+20)/2 = 15. This will split the data into two sets: white and purple.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

Then,



and now try Dcrit = (20 + 25)/2 = 22.5. This will split the data as follows,

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

and we’ll have:



**So if we’re to split the data by dosage, we’d use Dcrit = 15**. Now let’s do by age. The first critical age we’ll use is Acrit = (25 + 44)/2 = 37.5. This will split the data up as follows,

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

and,



whereas the other is possibility is to use Acrit = (44 + 54)/2 = 49, illustrated below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

which gives us:



**So if we’re to split the data by age, we’d use Acrit = 49**. Now let’s do by gender. We have only the male/female split, so data will look like this:

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

and,



**So it is apparent that separating by age, with Agecrit = 49 does the best job.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

So our decision tree will look like this so far:

Diagram

Description automatically generated

Moving on to the …

**Second Level**

So we’ll split our data according to the critical Age = 49.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

And now we want to analyze what we have left. What we *would* do is take the Age > 49 data (purple) and see which category, will result in least sum of squared errors, and split the Age > 49 data off accordingly. And then we’d do the same with the Age < 49 data (clear). But, according to my prescription that we cannot split a category into groups less than 2, we’ll actually have to stop now. So there.

Diagram

Description automatically generated

**Another example**

Say we have data like this:

Chart, scatter chart, bubble chart

Description automatically generated

I don’t know what the average effeciveness is, since I’m copying StatQuest video here. Let’s say it’s 40%. And I don’t know what SSEm(E) is either, consequently. But we don’t need to, to carry out the algorithm. So our root node would be:

A picture containing table

Description automatically generated

**First Level**

If we run a decision tree algorithm on this, we would test Dcrit\_1 values between 0 and 40,

Chart, scatter chart, box and whisker chart

Description automatically generated

and calculate the concomittant SSEm(E|Dcrit\_1):



(where < is the average for Ei < Dcrit\_1 and y> is the average for i > Dcrit\_1) until we found the one for which SSEm(E|Dcrit\_1) is smallest. Turns out our Dcrit\_1 = 14.5. So we’d split the data up like this:

Chart

Description automatically generated

So then our Decision Tree would look something like,

Diagram

Description automatically generated

Moving on the second level,

**Level 2**

Then we’d repeat the process for the data < Dcrit\_1 = 14.5, finding the split Dcrit\_2a for which SSEm(E|Dcrit\_2a) is smallest. Or well actually, since we have so few data points in the data < 14.5 category, we’d just quit right here and output the average effectiveness for this category. But we *would* repeat our level 1 calculation for the data > Dcrit\_1, finding the split Dcrit\_2b for which SSEm(E|Dcrit\_2b) is smallest. Turns out this is Dcrit\_2b = 29. So now our data is split lke this:

Chart

Description automatically generated

and our Decision Tree looks like this:

Diagram

Description automatically generated

Moving on,

**Level 3**

Then we’d repeat the process for the Dcrit\_1 = 14.5 < data < Dcrit\_2b = 29, finding the split Dcrit\_3a for which SSEm(E|Dcrit\_3a) is smallest. This will be found to be Dcrit\_3a = 23.5. And we would do the same for the data > Dcrit\_2b = 29 category, but we have fewer than the threshold in this category, so we’d just quit and output the average effectiveness here. So our data now looks like this:

Chart, histogram

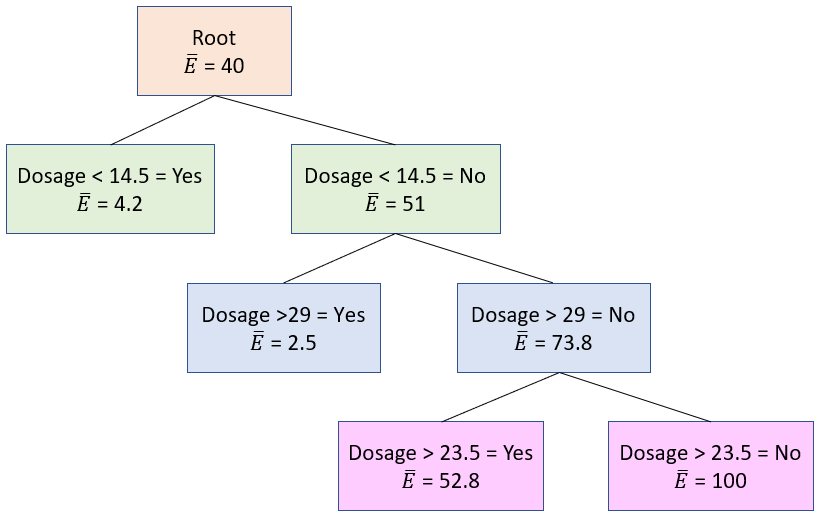
Description automatically generated

and,

Chart

Description automatically generated

And now our Decision Tree looks like this:



And since all of our categories now have ≤ the minimum threshold of points (7 I think in his example), we stop here.

**Goodness of Fit: R2 value**

R2 is a measure of the goodness of fit, of the regression curve. Just as for linear regressions, it’s given by:



where fi are the regression curve/tree data points, which one could say is a random variable since it depends in known way (if have formula/tree) on the random variable Yi, the data points we’re trying to fit. SSEf = sum of squares about f, and SSEm = sum of squares about mean. This is actually a general formula that applies to all curve fits. Apropos the fraction, the numerator is the ‘variation around the line, or f’, and the denominator is the ‘variation around the mean’). Can see R2 = 1 if there is no variation around the line, i.e., if the curve fits the data exactly. On the other hand, if the curve fits the data no better than the mean, then we get R2 = 0. If you have an R2 = 0.75, then you can say that 75% of the variation of the data is explained/predicted by the regression curve f.

**Feature Importance**

The importance of a feature is more or less given by how correlated it is with the outcome. The way it’s assessed is by looking at the tree and calculating the information gain when splitting on that feature. For instance, consider the following classification tree (too lazy to do regression tree)

A diagram of a number

Description automatically generated with medium confidence

splitting on thai\_7.0 < 0.5 results in an information gain of,



But for a regression tree we’d calculate IG via the change in the SSEm (see formula at top of page). We can do likewise for all the features, and then assign each feature a weight, given by:



This normalizes the feature importance to 1 basically. If a feature’s importance is low, then we should consider eliminating that variable from the model. It might be irrelevant to the underlying ‘physics’, and just be fitting noise.

**Training and testing on datasets with few instances per category**

Sometimes we have datasets with a category, like house location, which can take on multiple values, even hundreds. And there may be hundreds of rows corresponding to one location, and only one row corresponding to another. When we split the dataset for training and testing, our training dataset may have instances that our testing dataset does not have, and vice versa. For instance, say, we split the dataset below into training (grey-blue) and testing (white).

|  |  |  |  |
| --- | --- | --- | --- |
| **Dosage** | **Age** | **Sex** | **Effectiveness** |
| 10 | 25 | Female | 98 |
| 20 | 73 | Male | 14 |
| 35 | 54 | Female | 6 |
| 5 | 12 | Male | 44 |
| 25 | 44 | Female | 73 |

Our training algorithm would only make decisions based on Dosage and Age, since Sex is female for all. So then when we try to make predictions on our test dataset, the fact that Sex is all male, rather than female, simply won’t matter, since Sex never enters into those calculations. What if we had Sex = Female, Male, Nonbinary. And in our training dataset, we only encountered Sex = Female, Nonbinary? Well, our training decision tree algorithm would ask a question like something like Sex = Female?, or Sex = Nonbinary? So it would still be able to make a decision on our test dataset, as it could still answer those questions when Sex = Male. That being said, it might not make very good decisions, if being Male matters a whole lot to the outcome of the experiment.

**Exploring the Model and Hyperparameters**

Here’s the decision tree with pure data (N = 100) on a linear curve, and on a quadratic curve, with outliers in red. The black curve is the regression curve for the blue dots.

A graph of a tree

Description automatically generated A graph of a tree

Description automatically generated

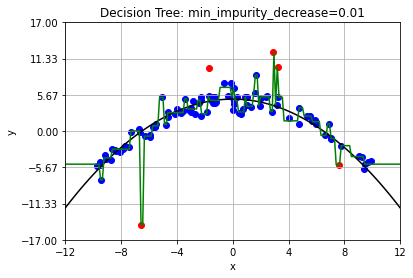
So without tuning hyperparameters, it basically will connect all the dots, as one would expect. Will note that outside the data range, on the edges of the graph, the decision tree would make increasingly horrible predictions. This is probably always going to be the case for decision tree models.

**Hyperparameter: loss**

So we can specify the loss function as *criterion = squared\_error, absolute\_error, etc*. I don’t imagine that would drastically change the performance here, apropos overfitting, and it doesn’t.

**Hyperparameter: min\_impurity\_decrease**

So when determining how to split a node, we look for the way which will maximize the (gini?) impurity decrease. If the gain isn’t substantial, then we may just forego doing so, as this might mean we’re following too closely the data. And so we can set a lower bound for this. Here’s a few examples.

 A graph with a line graph and a line graph

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

Description automatically generated

The higher the min is, the less overfitting we get. Can see that there is a Goldilocks zone.

**Hyperparameter: min\_samples\_leaf**

Another thing we can do is set a minimum on the number data points a leaf can contain.

A graph with a line graph and numbers

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

Description automatically generated A graph with red and blue dots

Description automatically generated

This does a pretty good job of tamping down on outliers. We’re basically fitting a decision tree to the local averages. A similar parameter is **min\_samples\_split**, which prescribes how many samples a leaf must have before it can be split.

**Hyperparameter: max\_depth**

This guy sets an upper limit on the depth of the tree.

A graph of a tree

Description automatically generated A graph with a line graph and numbers

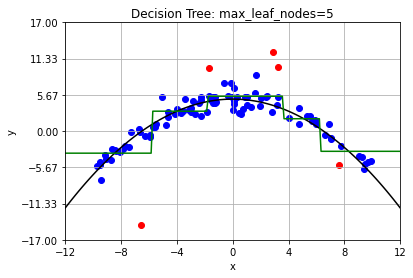
Description automatically generated with medium confidence A graph of a graph with a line graph

Description automatically generated with medium confidence

Doesn’t seem to help too much.

**Hyperparameter: max\_leaf\_nodes**

This guy sets an upper limit on the number of leaf (i.e. terminal) nodes.

 A graph with red and blue dots

Description automatically generated A graph with a line graph and a line graph

Description automatically generated with medium confidence

So anyway, yeah we can see that decision trees are apt to overfit. And of course there are many other hyperparameters.