**Decision Trees + Cat Boost**

When we have categorical data that can take more than two values, one usually resorts to one-hot-encoding. But if there are a lot of categorical values, this can really slow things down, and it’d be faster to assign the categorical values to numbers. Target-Encoding and Cat Boost Target-Encoding are ways in which we can do this. So let’s run through how we’d build a regression tree based on this type of encoding. We’ll take as our example here, the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **Weight (kg)** |
| 1.6 | Blue | Male | 88 |
| 1.6 | Green | Female | 76 |
| 1.5 | Blue | Female | 56 |
| 1.8 | Red | Male | 73 |
| 1.5 | Green | Male | 77 |
| 1.4 | Blue | Female | 57 |

We want to write a decision tree to estimate the outcomes Y = weights. We already discussed the basic concepts of SSE, predicted outcome, and Information Gain, when we covered normal regression trees. So I’ll just jump into how we construct the tree. Those concepts are a little modified from before though, as well as how we construct the tree. I think it’s easiest to just to it all together.

**Constructing the Decision Tree**

I’m not sure how to phrase this in the form of trying to minimize an overall loss function. So I’ll just jump into the methodology.

***0th Tree***

Before using any trees, we start with a simple classification whereby we predict everything to be a single constant parameter, so fi = f(0), and this is:



That was easy!

***1st Tree***

So we’d like to construct a tree to improve this classification. So we’ll look for an increment, Δfi(1), that we can add to our present prediction, f(0), to get a new prediction: fi(1) = f(0) + Δfi(1). Like I said, not sure how to fashion this from a minimization principle to start. So let’s just say, we will start by randomizing the rows. And then in the output Y column, for the purposes of Target Encoding, split the data about a critical value, ycritical, so that data below it is assigned the categorical label 0, say, and data above it the categorical label 1 (StatQuest guy says you can go to CatBoost website for instructions on how to use more finesse than a strict binary y column labeling, which is probably something to look into vis a vis regression). So then you split the data into leaves, L. I think it’s taken for granted that you will just bifurcate the data in such a way as to maximize the the cosine similarity at each split. The cosine similarity is defined as:



where **y**Lℓ = {yLℓ,i} is the vector of outputs in a leaf, **f**Lℓ(0) = {fLℓ,i(0)} is vector of former predictions of the outputs in that leaf. Of course this is zero presently. And **y**Lℓ - **f**Lℓ(0) is the vector of residuals in that leaf. Finally, Δ**f**Lℓ(1) = {ΔfLℓ,i(1)} is the vector of incremental pseudo-predictions for each of those elements in **y**Lℓ. Note the incremental pseudo-prediction of the ith row in a leaf is the average of all the residuals *preceding* that one in the leaf. Should contrast the increment pseudo-predictions ΔfLℓ,j(1) from the actual prediction of the leaf. The latter is not index-dependent. It’s given by:



where nLℓ is the number of terms in the leaf (again, fLℓ,j(0) = 0). Therefore it is just the average of all the residuals in the leaf. 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. And I’ve only put labels on the right side of the tree for space considerations.

A picture containing text, diagram, line, screenshot

Description automatically generated

We will typically use a greedy algorithm to find these leaves, just looking for the split that will maximize the cosine similarity, at each level. There is a caveat though, whatever split we use one one node, the Cat Boost algorithm mandates that we use the same split on the other node. Once we’re done, we’ll have a set of predictions for the leaves:



But we won’t wholly endorse the jump, as that would overfit. So we say,



***2nd Tree***

So we’d like to construct a tree to improve this classification. So we’ll look for an increment, Δf(2), that we can add to our present prediction, f(1), to get a new prediction: fi(2) = fi(1) + Δfi(2). Before we construct the new tree, we’ll randomize the positions of the rows. This will randomly change their numerical encodings, which we have to recalculate. Then we’ll put all the data in a root node and then start breaking it down by maximizing the cosine similarity again.



where **y**Lℓ = {yLℓ,i} is the vector of outputs in a leaf, **f**Lℓ(1) = {fLℓ,i(1)} is vector of former predictions of the outputs in that leaf. This will *not* be zero now. And **y**Lℓ - **f**Lℓ(1) is the vector of residuals in that leaf. Finally, Δ**f**Lℓ(2) = {ΔfLℓ,i(2)} is the vector of incremental pseudo-predictions for each of those elements in **y**Lℓ. Note the incremental pseudo-prediction of the ith row in a leaf is the average of all the residuals *preceding* that one in the leaf. Should contrast the increment pseudo-predictions ΔfLℓ,j(1) from the actual prediction of the leaf. The latter is not index-dependent. It’s given by:



where nLℓ is the number of terms in the leaf. Therefore it is just the average of all the residuals in the leaf. 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. And I’ve only put labels on the right side of the tree for space considerations.

A picture containing text, diagram, screenshot, line

Description automatically generated

We will typically use a greedy algorithm to find these leaves, just looking for the split that will maximize the cosine similarity, at each level. There is a caveat though, whatever split we use one one node, the Cat Boost algorithm mandates that we use the same split on the other node. Once we’re done, we’ll have a set of predictions for the leaves:



But we won’t wholly endorse the jump, as that would overfit. So we say,



***3rd Tree, etc.***

And we’d proceed likewise for all other trees. Presuming we stop after n trees, we’d have:



We’d stop when the predictions fi(n+1) and fi(n) are ‘close enough’.

***Making Predictions***

To make predictions on testing data, we need to know how to encode the testing row’s category. Turns out we continue to use the same formula as before. We mentally append the row to the training data, and use the formula,



Note that the encoding of the testing row will have nothing to do with the testing row itself, only with the training data. But it also has nothing to do with the *order* of rows in the training data, as we’ll get the same result regardless.

**Example**

For example, let’s take this table, where we try to predict weight from height, favorite color, and gender.

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **Weight (kg)** |
| 1.6 | Blue | Male | 88 |
| 1.6 | Green | Female | 76 |
| 1.5 | Blue | Female | 56 |
| 1.8 | Red | Male | 73 |
| 1.5 | Green | Male | 77 |
| 1.4 | Blue | Female | 57 |

*Zeroth Tree*

So the initial prediction is:



I’ll put this result in a tree stump, even though it’s not formally part of a tree,

A close-up of a sign

Description automatically generated with low confidence

And we’ll move on to the first tree,

*First Tree*

The first order of business is to encode the Color column. So we’ll assign the W column binary values. The average value is 72, but I guess I’ll go off of the median rather, and say W < 75 → 0, and W ­­ > 75 → 1. So,

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **Weight** |
| 1.6 | Blue | Male | 88 (1) |
| 1.6 | Green | Female | 76 (1) |
| 1.5 | Blue | Female | 56 (0) |
| 1.8 | Red | Male | 73 (0) |
| 1.5 | Green | Male | 77 (1) |
| 1.4 | Blue | Female | 57 (0) |

Then according to Cat Boost Target Encoding, we’d translate the colors to:



where m is number of previous rows of the value Ai, and k is a user defined number we’ll take to be k = 0.05. And Y is W.



So,

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **Weight** |
| 1.6 | Blue = 0.05 | Male | 88 (1) |
| 1.6 | Green = 0.05 | Female | 76 (1) |
| 1.5 | Blue = 0.525 | Female | 56 (0) |
| 1.8 | Red = 0.05 | Male | 73 (0) |
| 1.5 | Green = 0.525 | Male | 77 (1) |
| 1.4 | Blue = 0.35 | Female | 57 (0) |

And now we’ll start our tree. Can see that we will be fitting our decision tree to the residuals W – W(0). So let’s replace the weight column with residuals. This is the same obviously:

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **W – W(0)** |
| 1.6 | Blue = 0.05 | Male | 88 (1) |
| 1.6 | Green = 0.05 | Female | 76 (1) |
| 1.5 | Blue = 0.525 | Female | 56 (0) |
| 1.8 | Red = 0.05 | Male | 73 (0) |
| 1.5 | Green = 0.525 | Male | 77 (1) |
| 1.4 | Blue = 0.35 | Female | 57 (0) |

And now we’ll start our tree. Our root node would be where we place all the data initially, and output ΔWR(1) = (1/n)Σi(W – W(0)) = 72. So,

A picture containing text, font, screenshot, number

Description automatically generated

Now we want to do a little better and split the root node. So we want to figure out the column most closely correlated with W – W(0). That’ll be the one with the largest cosine similarity, i.e., largest ΔCS(1)(W|ΔW(1)→L) value. Not going to look at all the possible splits. Apropos height, it looks like the best split is at Hcrit = 1.55m. By Color, we’ll try Ccrit = 0.20. And gender is obviously either Male or Female. Let’s try height first. And the output is the average weight of rows above it in that category:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **W – W(0)** | **ΔWH,i(1) = (1/i)Σj<i (WH,j -WH,j(0))** |
| 1.6 | Blue = 0.05 | Male | 88 (1) | 0 |
| 1.6 | Green = 0.05 | Female | 76 (1) | 88 |
| 1.5 | Blue = 0.525 | Female | 56 (0) | 0 |
| 1.8 | Red = 0.05 | Male | 73 (0) | (88 + 76)/2 = 82 |
| 1.5 | Green = 0.525 | Male | 77 (1) | 56 |
| 1.4 | Blue = 0.35 | Female | 57 (0) | (56 + 77)/2 = 67 |

Not that it matters right now, or perhaps at all, but the output of the two H leaves would be the averages of the values in the leaves. So,



And cosine similarity is:



Now let’s look at Color. If we make the split at C­crit = 0.20, then we have, well, the same thing, actually.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **W – W(0)** | **ΔWC,i(1) = (1/i)Σj<i (WC,j - WC,j(0))** |
| 1.6 | Blue = 0.05 | Male | 88 (1) | 0 |
| 1.6 | Green = 0.05 | Female | 76 (1) | 88 |
| 1.5 | Blue = 0.525 | Female | 56 (0) | 0 |
| 1.8 | Red = 0.05 | Male | 73 (0) | (88 + 76)/2 = 82 |
| 1.5 | Green = 0.525 | Male | 77 (1) | 56 |
| 1.4 | Blue = 0.35 | Female | 57 (0) | (56 + 77)/2 = 67 |

Not that it matters right now, or perhaps at all, but the output of the two C leaves would be the averages of the values in the leaves. So,



And cosine similarity is:



And now let’s do Gender,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **W – W(0)** | **ΔWG,i(1) = (1/i)Σj<i (WG,j -WG,j(0))** |
| 1.6 | Blue = 0.05 | Male | 88 (1) | 0 |
| 1.6 | Green = 0.05 | Female | 76 (1) | 0 |
| 1.5 | Blue = 0.525 | Female | 56 (0) | 76 |
| 1.8 | Red = 0.05 | Male | 73 (0) | 88 |
| 1.5 | Green = 0.525 | Male | 77 (1) | (73 + 88)/2 = 81 |
| 1.4 | Blue = 0.35 | Female | 57 (0) | (56 + 76)/2 = 66 |

Not that it matters right now, or perhaps at all, but the output of the two H leaves would be the averages of the values in the leaves. So,



And cosine similarity is:



So looks like the split by Height, or Color, has the greatest cosine similarity. Let’s just split by Color then, and we’ll say:

A picture containing text, font, screenshot, line

Description automatically generated

Going to go down one more level. So an idiosyncrasy of the Cat Boost algorithm is that we use the same decision to split both leaves of the present level. This is because it’s faster to run the algorithm that way, and also that it shouldn’t matter too much since that just makes each decision tree an even weaker learner (which is okay because we’re only endorsing each tree by small increment so we’ll have to use more increments to get a good fit). Not sure how we decide which decision o make though. Perhaps it doesn’t matter too much. Let’s split by Gender. Then we need to recalculate the ΔW(1) for the subcategories. We find,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **W – W(0)** | **ΔWC,G,i(1) = (1/i)Σj<i (WC,G,j – WC,G,**j**(0))** |
| 1.6 | Blue = 0.05 | Male | 88 (1) | 0 |
| 1.6 | Green = 0.05 | Female | 76 (1) | 0 |
| 1.5 | Blue = 0.525 | Female | 56 (0) | 0 |
| 1.8 | Red = 0.05 | Male | 73 (0) | 88 |
| 1.5 | Green = 0.525 | Male | 77 (1) | 0 |
| 1.4 | Blue = 0.35 | Female | 57 (0) | 56 |

The output of the leaves would be:



So, our Tree would now be:

A picture containing text, font, diagram, line

Description automatically generated

And our decision tree’s predictions would be (going to set α = 0.5 for illustration´s sake):



*Second Tree*

Now to do the second tree, the first thing we do is randomize the rows. This will change the encodings of the colors. So say we reorder the columns like this:

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **Weight** |
| 1.8 | Red | Male | 73 (0) |
| 1.4 | Blue | Female | 57 (0) |
| 1.6 | Green | Female | 76 (1) |
| 1.6 | Blue | Male | 88 (1) |
| 1.5 | Green | Male | 77 (1) |
| 1.5 | Blue | Female | 56 (0) |

Then using the same formula as before,



color encodings will become,



Filling these in, looks like only one color value changes:

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **Weight** |
| 1.8 | Red = 0.05 | Male | 73 (0) |
| 1.4 | Blue = 0.05 | Female | 57 (0) |
| 1.6 | Green = 0.05 | Female | 76 (1) |
| 1.6 | Blue = 0.025 | Male | 88 (1) |
| 1.5 | Green = 0.525 | Male | 77 (1) |
| 1.5 | Blue = 0.35 | Female | 56 (0) |

Then we need to update the residuals. This contradicts his video, but I’d think we should rather use the last tree’s prediction, for the values run through the next decision tree. That’s what we’ve done for all the other decision tree algorithms. And that’s what I’m going to say is going on. Assuming so, then we will use the formula,



So,

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **W – W(1)** |
| 1.8 | Red = 0.05 | Male | 73 – 41 = 32 (0) |
| 1.4 | Blue = 0.05 | Female | 57 – 38 = 19 (0) |
| 1.6 | Green = 0.05 | Female | 76 – 38 = 38 (1) |
| 1.6 | Blue = 0.025 | Male | 88 – 39 = 49 (1) |
| 1.5 | Green = 0.525 | Male | 77 – 39 = 38 (1) |
| 1.5 | Blue = 0.35 | Female | 56 – 29 = 27 (0) |

And now we’ll start our tree. Our root node would be where we place all the data initially, and output ΔWR(2) = (1/n)Σi(W - W(1)) = 34 So,

A picture containing text, font, screenshot, number

Description automatically generated

Now we want to do a little better and split the root node. So we want to figure out the column most closely correlated with W – W(1). That’ll be the one with the largest cosine similarity, i.e., largest ΔCS(2)(W|ΔW(2)→L) value. But not going to look at all the possible splits. And for practice, I’m just going to presume it’s by color, using Ccrit = 0.20 again.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **W – W(1)** | **ΔWC,i(2) = (1/i)Σj<i (WC,j - WC,j(1))** |
| 1.8 | Red = 0.05 | Male | 32 (0) | 0 |
| 1.4 | Blue = 0.05 | Female | 19 (0) | 32 |
| 1.6 | Green = 0.05 | Female | 38 (1) | (32+19)/2 = 25 |
| 1.6 | Blue = 0.025 | Male | 49 (1) | (32+19+28)/3 = 26 |
| 1.5 | Green = 0.525 | Male | 38 (1) | 0 |
| 1.5 | Blue = 0.35 | Female | 27 (0) | 38 |

And the output of the leaves would be:



And cosine similarity is:



Next we would try other splits, to ascertain the one with the largest cosine similarity, but I’m going to stop here. We’ll just split by Color:

A diagram of a root

Description automatically generated with low confidence

If we stop here, then using α = 0.5 again, our predictions after the second tree would be:



Making Predictions on Testing data

To make predictions on testing data, we need to know how to encode the testing row’s color. Turns out we continue to use the same formula as before,



treating our testing row as an addition to the training data.

|  |  |  |  |
| --- | --- | --- | --- |
| **Height (m)** | **Favorite Color** | **Gender** | **Weight** |
| 1.6 | Blue = 0.05 | Male | 88 (1) |
| 1.6 | Green = 0.05 | Female | 76 (1) |
| 1.5 | Blue = 0.525 | Female | 56 (0) |
| 1.8 | Red = 0.05 | Male | 73 (0) |
| 1.5 | Green = 0.525 | Male | 77 (1) |
| 1.4 | Blue = 0.35 | Female | 57 (0) |
| new height | new color | new Gender | new weight |

Note that the encoding of the testing row will have nothing to do with the testing row itself, only with the training data. But it also has nothing to do with the *order* of rows in the training data, as we’ll get the same result regardless. So a testing Blue, Green, or Red column would have the following encodings,



I assume all the testing rows with color Blue, Green, Red would have those same values. So if we were to test a Male, Blue person, we’d guess their weight to be:

