Introduction

Some models are inherently interpretable. An example of this is a linear model, like the one below:

[Model]

Or, in ordinary language:

[Explanation of model]

You can see what each term is doing to the overall prediction, and (conditional on being familiar with the quantities in question) roughly how important each of them are. A proper discussion of the benefits of interpretability deserves its own post, but for now I'll treat it as axiomatic that – all else equal – it is usually better to be able to see what you're doing.

However, most models aren't that straightforward. A typical tree-based model, as produced by the popular model-building algorithm XGBoost, looks like this:

[Picture of 5 XGB trees with treedepth 3]

. . . and so on for 95 more trees.

This is not inherently interpretable. No human could hold it in their head and make sense of it. People who use models like this cannot understand their behaviour, and are forced to analyze their output to get a (limited) sense of how they work.

The main cause of uninterpretability here is interaction. If the decision trees that comprise this model were of depth 1 ('decision stumps'), then no matter how many of them there were, you could sum up the effect of each variable with a graph like this:

[PDP graph]

This wouldn't be quite as interpretable as a linear model, but it would still be pretty good; while you couldn't take in the entire model at a glance, you would still be able to see exactly how each individual feature affects the outcome.

However, in a complex system, effects of variables are modified by other variables. While we should regard an algorithm that says "I've found a three-way interaction in this dataset, please use it to help inform your decisions" with some fraction of the skepticism we'd show a sociologist saying the same thing, frequently datasets are large enough and signals strong enough that we actually can meaningfully model interactions. So in these cases, models without the capacity to model interactions will underperform those which can.

Does this doom inherently interpretable models to inferiority? Probably. But there is *less* doom than you might think. This essay outlines five synthetic scenarios in which problems other than interaction cause high-treedepth models to outperform, and shows how they can be addressed without increasing treedepth.

I do this to support the following implications:

- These problems are present in at least some real-world tree-based models, and are at least somewhat responsible for at least some of the performance gap between high- and low-treedepth models.

- For a given dataset, the gap between the performance of conventional tree-based models and the best inherently interpretable model may be smaller than you think.
- For a given dataset, the maximum treedepth required for optimum performance by a tree-based model may be lower than you think.
- At least a few real-world models which appear to benefit from treedepth>1 can probably have their performance matched by inherently interpretable models once these problems are addressed.

So, what non-interaction reasons might a tree-based model have for high treedepth?

High treedepth can compensate for insufficient learning

Scenario One: Generate a dataset by flipping ten (virtual) coins a thousand times apiece. The explanatory variables are the results of the ten coin filps; the response variable is the number of heads in each set of ten flips, plus some Gaussian random noise. Then model it, using XGBoost with varying treedepths. Note that, despite no interactions between variables, and despite the form of this process being ideal for unity-linked Gaussian modelling, higher-treedepth models consistently get better MAE and RMSE scores when tested on identically-generated outsamples.

Table 1a: Example results from Scenario 1

Scenario	Mean Average Error	Root Mean Square Error
max_depth = 1	1.617	2.016
max_depth = 2	1.524	1.888
max_depth = 3	1.482	1.825

Oh, come on!, you may reasonably exclaim. *Did you think I would just buy that without looking at the code you linked to? You set the number of trees in the model to four! Even if you got the learning rate exactly right, it is literally impossible to fit ten effects with four degrees of freedom. Of course four high-depth trees will do better than four stumps!*

Terribly sorry, dear Reader. Here is a version of the modelling code which addresses that particular concern; you will notice that the results displays the same effect.

Table 1b: Example results from Scenario 1, with 100 trees this time

Scenario	Mean Average Error	Root Mean Square Error
max_depth = 1	1.617	2.016
max_depth = 2	1.524	1.888
max_depth = 3	1.482	1.825

No shit it displays the same effect! You set the learning rate to 0.01, and only gave it 100 trees!

And why is that a problem?

Because you-

Because the total amount of learning – that is, the quantity which makes a model underfit when too low and overfit when too high – has an arbitrarily complex but monotonically increasing relationship with treedepth, treecount, and learning rate? And because, when the other hyperparameters are too low, increasing treedepth will make the model better because it moves it away from being underfit, even if there are no interactions to model?

Don't interrupt! But yeah, that's pretty much it.

Well, for what it's worth, you're right: if you get the amount of learning correct, modelling complexity that isn't there will only make things worse. But the exact reasoning is important.

How? All I've learned from this is to not blatantly undertrain a model.

You've also learned not to subtly undertrain a model. For example, if you tried to find the best treedepth for a model by varying only treedepth, you'd get the "what's the best treedepth?" question mixed up with the "what's the best amount of learning?" question. In order to actually find the best treedepth, you'd need to gridsearch by varying it and at least one of the other hyperparameters simultaneously, and comparing the best value for each treedepth.

Table 1c: values of MAE for different values of treedepth and learning rate

	max_depth = 1	max_depth = 2	max_depth = 3
learning_rate = 0.05	1.0948	0.9323	0.8541
learning_rate = 0.1	0.9053	0.8289	**0.8164**
learning_rate = 0.2	0.7994	0.8124	0.8166
learning_rate = 0.3	0.789	**0.8099**	0.8255
learning_rate = 0.4	**0.7883**	0.813	0.825
learning_rate = 0.5	0.7884	0.811	0.836
Best	**0.7883**	**0.8099**	**0.8164**

And even that wouldn't be good enough if you picked the wrong range(s) for the other hyperparameter(s) . . .

You know everything you said works backwards, right? All your reasoning about how subtle undertraining can cause too-high treedepth counts as reasoning for why subtle overtraining can cause too-low treedepth.

That's true. But this post is about ways people can end up with artificially high treedepth, and it'd be remiss of me to mention this even when the reverse can be true.

High treedepth can compensate for incorrect linkages

Scenario Two: You are using the colour and size of gemstones to predict their price. The pattern you are attempting to model is shown in the table below.

Table 2: Hypothetical gemstone setup

	Size = Small	Size = Big
Colour = Red	$100	$300
Colour = Blue	$200	$600

If you fit an additive model – or, to use the proper terminology, a "unity-linked" model – you would need treedepth of two or more to fully model price. Start with $100, add $100 for blueness, add $200 for bigness, and add a further $200 for the interaction of blueness and bigness.

But the effects in this situation aren't additive. If you fit a multiplicative – or "log-linked" – model, you would only need treedepth of 1: blueness multiplies price by 2, bigness multiplies price by 3, and that explains the entire table.

That's a cute solution to a toy problem. But when dealing with the kind of data people hire people to deal with – as opposed to the kind of data you can just use physical or mathematical laws to predict – behaviour will be complex enough that effects from a given feature will be additive AND multiplicative AND probably some other stuff. Where does that leave you?

It guarantees that a model with a conventional approach to linkage will never be perfect. But we can minimise the impact of this phenomenon – and hence the treedepth required – by using the least incorrect linkage.

High treedepth can compensate for multiple populations

Scenario Three: You're back flipping coins; this time, thirteen per row. The first determines whether it's an A-row or a B-row: don't keep that one as an explanatory variable, but keep the others. Your value for the response variable y starts at 1. A heads on flips 2-4 doubles the value of y for the row; a heads on flips 5-7 doubles the value of y for an A-row but halves it for a B-row; a heads on flips 8-10 halves it for an A-row but doubles it for a B-row; and a heads on flips 11-13 halves it for any row. Once the halvings and doublings are all applied, y is replaced with the output of a Poisson random number generator with lambda=y.

That sounds complex and arbitrary. In what real-world contexts could you get a dataset like that?

Let's say you work for a zoo and you're analyzing staff injuries caused by the exotic felines: each row represents a lion or a tiger. You have three binary variables which – when true – raise the chance of injuries from both lions and tigers; you have three more which raise it for lions but lower it for tigers; three more which raise it for tigers but lower it for lions; and three which would lower it for both (you, of course, know none of this: you just have an inkling that the data you have might translate to injuries differently in the context of a different species). You also have a count of injuries caused by each animal. But what you don't have is any information that tells you which animal is a lion or a tiger; the owner just marked them all as "BIG CATS". Using this data, you have to predict the number of injuries that would be caused by incoming BIG CATS.

Or, less fancifully, let's say you work in insurance and you're trying to model the number of claims future customers will make. You know a priori that some customers in the dataset you're modelling from are Despicable Frauds, and some are Honest Citizens; but it's impossible, or at the very least prohibitively difficult, to tell them apart based only on the data you have. Some of your explanatory variables – when true – raise the average number of claims made by both Frauds and Citizens, some lower it for both, and some raise it for one subpopulation but lower it for the other. Using this data, you have to build a model that uses the same explanatory variables to predict the number of claims that will be made by new customers.

Or let's say you're modelling the height of crabs in millimetres. Some are male and some are female, but you don't know which are which. Some of your observational data would suggest the same change in height regardless of sex, and some would suggest an increase in height for one sex but a decrease for the other. Using this data, you have to discover how observational data will predict the height of unseen crabs.

Basically: any situation where there are multiple subpopulations in the dataset you're analyzing, but you don't necessarily know which row belongs to which subpopulation, is a situation which will be somewhat isomorphic to this.

So, in other words, any situation where the datapoints being modelled can't be approximated as homogenous.

Pretty much. And I think that's always going to be a possibility when the rows represent humans; at time of writing, there are around 7.7 billion types of person.

Anyway, we generate this dataset, use identical parameters to generate another as a test set, and train tree-based models (with appropriate linkage and error distribution) on the former to test on the latter. Despite no interactions in the data generation process, we consistently get results where treedepth=3 does better than treedepth=1:

Table 3a: Example results

	MAE	**RMSE**
max_depth = 1, 100 trees	128.08	222.27
max_depth = 1, 200 trees	128.17	222.82
max_depth = 2, 100 trees	127.75	222.87
max_depth = 3, 100 trees	122.08	222.12

Shouldn't you be gridsearching if you want to find the best treedepth? Didn't you make that pretty clear two sections ago?

In a situation like this, doubling the treecount and getting similar results is sufficient to prove that improvement caused by increased treedepth isn't just compensation for underfit.

Sounds sketchy, but that's just the most obvious problem with what you're showing me. More importantly, how can you say there are no interactions? The effects of flips 5-10 are completely different depending on the value of flip 1!

Yes, but there are no coded-in interactions between features provided to the modeller.

Semantics! If there weren't de facto interactions between the explanatory features, high-treedepth models wouldn't outperform low ones.

That’s the point I’m making. Like in the previous section, the behaviour can be analyzed on its own terms [TODO: build a mixture-modeller, apply it, and then link to it], and therefore be presented much more simply. This is called a “mixture model”, and the output in this context looks like this:

[TODO: rest of this section]

High treedepth can compensate for multiple responses

[TODO: this section]

High treedepth can compensate for missing explanatory columns

[TODO: this section]

Summary, Analysis, and Wild Speculation

[TODO: Starry-eyed ranting]

Other Panaceas

[TODO: use multiresp to handle a problem where trees would have been a more obvious solution]

Appendix: Failed Approaches

[TODO: talk briefly about how jag and outliers didn’t make the cut]