Introduction

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

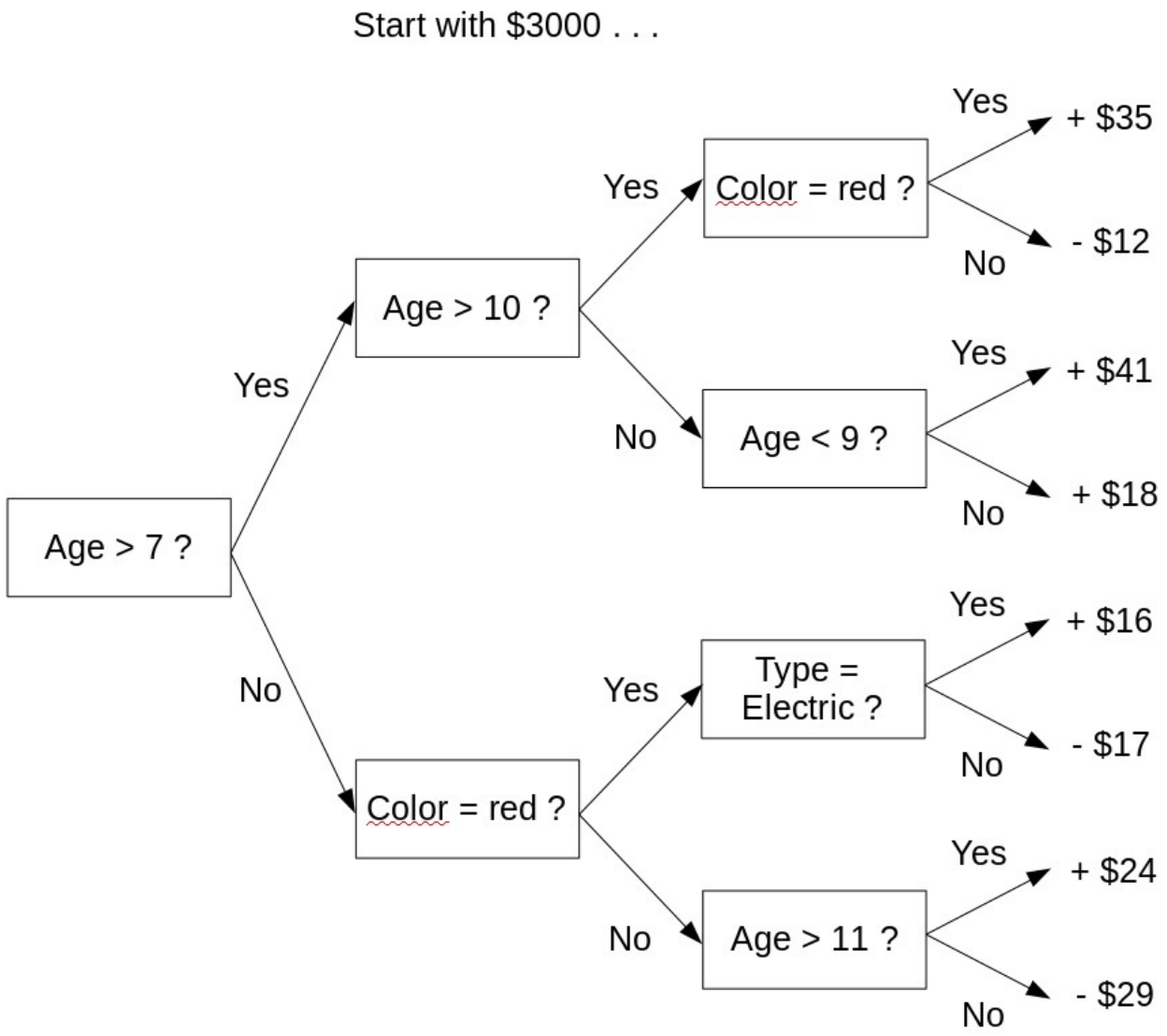
Price = $3000 + $2000 * electric - $1000 * diesel + $50 * red + $100 * economy - $450 * years

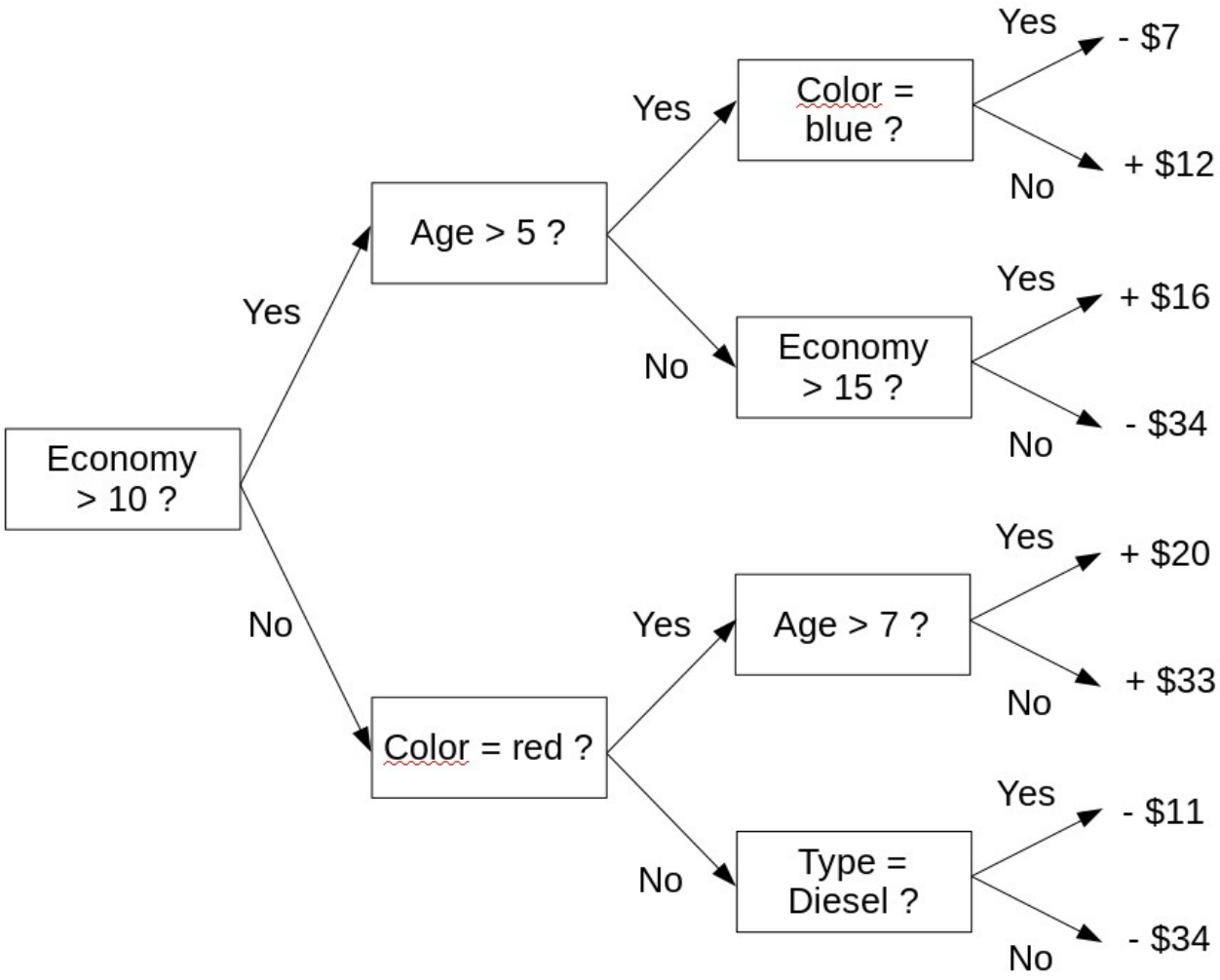
Or, in ordinary language:

“When estimating the price of a car, start with $3000, add $2000 if it’s electric, subtract $1000 if it’s diesel, add $50 if it has a red paint job, add $100 for every point in its fuel economy rating, and subtract $450 for every year it’s been on the market.”

You can see what each term is doing to the overall prediction, and roughly how important each of them is. 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 with its default treedepth of 3, looks like this:

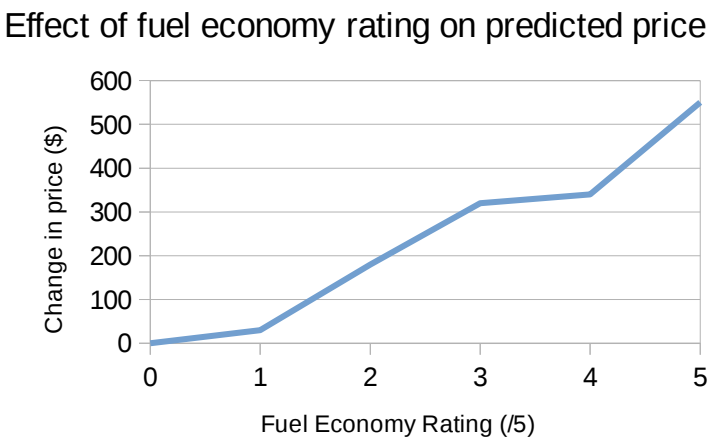


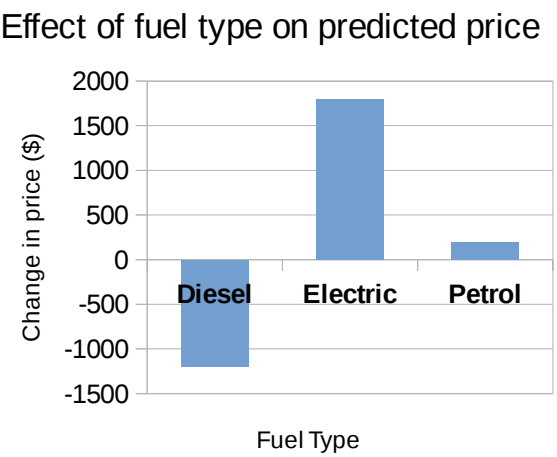


. . . and so on for 98 more trees.

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 all the decision trees that comprise this model had only one question (max_depth=1, also known as 'decision stumps'), then no matter how many of them there were, you could sum up the effect of each variable with graphs like this:





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 could be addressed without using treedepth>1.

By providing these existence proofs, I aim to support the following implications:

- These problems are present in at least some real-world datasets to which tree-based models have been applied, 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	1.967	2.382
max_depth = 2	1.916	2.304
max_depth = 3	1.886	2.248

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 a 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 effect I've shown is still 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 entangled 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 not mention this phenomenon 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 pricing system

	Size = Small	Size = Big
Color = Red	$100	$300
Color = 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 -tives. 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 responses

Scenario Three: You're back flipping coins; this time, four per row. Start with a value of 1; then, double it for every heads in the first two coins or tails in the second two, and halve it for every tails in the first two coins or tails in the second two. Once the halvings and doublings are all applied, use the result as the lambda-value for a Poisson random number generator, and take the output as your A-response. Then do the same thing with the same coin results to get B-response and C-response, but with the double-on-heads-halve-on-tails coins as one-and-three and one-and-four respectively. Your total response – that is, your value for y – is the sum of the A-, B-, and C-responses.

That sounds complex and arbitrary. Did you engineer a dataset that would never occur in the real world, just so you could get whatever effect you're hunting for?

Any situation where your response variable can be decomposed into multiple processes which deserve their own models, but where all you have access to is the sum of the outputs from those processes, will end up resembling this.

For example: let's say you're predicting staff injuries caused by lions at a zoo, and bites and scratches and headbutts should all be modelled separately, but you're only given the total injury count per lion per year. Or you're building an insurance model predicting claims from customer data, and some of the claims are fraudulent and some aren't, but you only know how many claims each customer made in total. Or you're predicting human height, which can be decomposed into leg height plus torso height plus head height, but you're only given the head-to-toe measurement.

So, in other words, any situation where the thing you're modelling contains more than one type of thing?

Not quite any situation: if we'd made the effects of the explanatory features on the responses additive instead of multiplicative, they'd be able to combine simply, and there'd be no benefit to increased treedepth. But in our log-linked setup, despite no interactions in the response generation process, treedepth=3 models consistently beat treedepth=1 models.

[TODO: give table real results]

Table 3a: Example results

	MAE	**RMSE**
max_depth = 1, 100 trees	2.9414	4.0419
max_depth = 1, 200 trees	2.9413	4.0596
max_depth = 3, 100 trees	1.9583	2.6376

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 (and, crucially, not significantly better) results is sufficient to prove that improvement caused by increased treedepth isn't just compensation for underfit.

Sounds kinda sketchy. Speaking of sketchiness, is there a reason you keep not including seeds and environments for your code? How am I supposed to check you didn't fabricate these results?

Seeds are irrelevant because I'm confident you'll get similar results every time you run the linked code. And including an environment would be like saying the phenomenon is fragile enough that using XGB version 1.3.0 instead of 1.2.0 might make it point the other way; it isn't, so I won't. Any other issues, dear Reader?

Yes; I was just getting the procedural stuff out of the way before bringing up my real concerns. How can you say there are 'no interactions in the response generation process'? They pretty obviously affect each other's effects on the total output in ways that can't be summed up in an appropriately-linked linear system.

True, but there are no interactions between the explanatory features in each individual response.

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

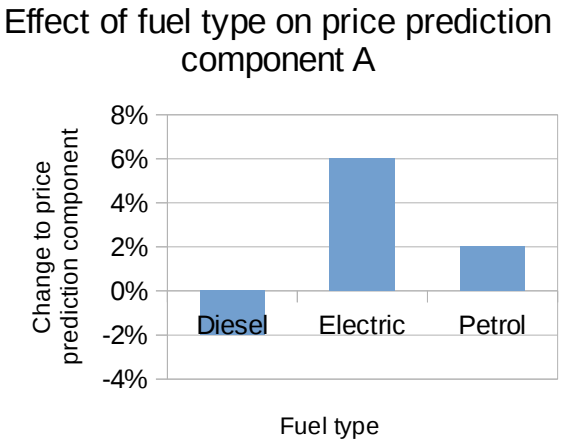
As in the previous section, if you want to approach the system using the framework of interactions, you can go ahead and do that and you'll get the right answer. But also as in the previous section, the system can be analyzed on its own terms, and therefore be presented much more simply without sacrificing model quality.

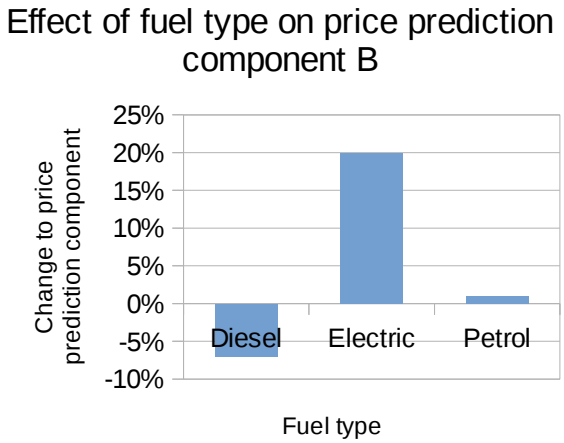
If we ignore the randomness, we have a system which can be written like this:

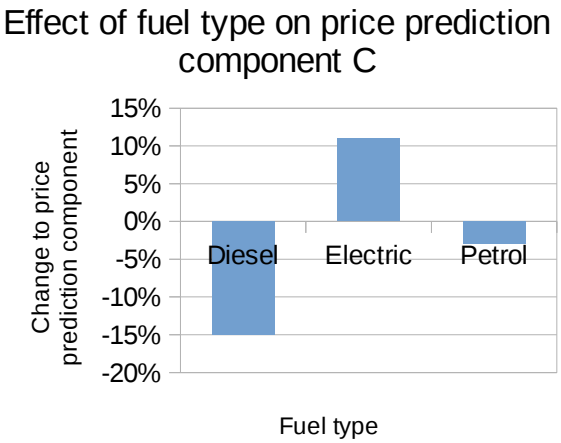
$y = e^{(k_{0a}+k_{1a}x_1+k_{2a}x_2+k_{3a}x_3+k_{4a}x_4)} + e^{(k_{0b}+k_{1b}x_1+k_{2b}x_2+k_{3b}x_3+k_{4b}x_4)} + e^{(k_{0c}+k_{1c}x_1+k_{2c}x_2+k_{3c}x_3+k_{4c}x_4)}$

Where *y* is the dependent variable, the *x*s are the coin flips, and the *k*s are the coefficients which define the system. So we can set up a model of that form, apply an appropriate error distribution, and use gradient descent to vary the *k*s until they reach an optimum set of values.

In this simplified case, we end up with a model which is defined by the fifteen k-values, so you can still see at a glance how each feature affects the predicted value of y. In a scenario with categorical and quantitative features as well as binary ones, we'd have the same kinds of graphs as shown in the introduction, just N times as many (where N is the number of parallel models fit, in this case 3).







That's great assuming you know what N should be, but what if you don't have any idea what sub-responses your response is made from?

The lovely thing is that you don't have to. A person fitting two sub-responses to their lion injury data doesn't need to know that some are bites and some are scratches; they can just set N to 2 and let the model do the rest. And they don't need to know a priori that N is 2, either: they can treat it like people treat treedepth, and just increase it until doing so stops helping.

Huh. And this optimum set of k-values you mentioned a few paragraphs back: is that a global optimum? Because gradient descent models tend to tend to local minima and get stuck, especially when only given a few degrees of freedom.

The fact that gradient descent is a viable way to solve this isn't a core part of my argument, but as it happens, when I compare the results of the canonical modelling code to equivalent results from tree-based models . . .

Table 3b: Example results from tree-based and canonical models

	MAE	**RMSE**
max_depth = 1, 100 trees	2.9390	4.0973
max_depth = 1, 200 trees	2.9370	4.1085
max_depth = 3, 100 trees	**2.0208**	**2.7222**
Canonical model, N=1	2.9370	4.1087
Canonical model, N=2	2.4667	3.4335
Canonical model, N=3	**2.0207**	**2.7223**

. . . it sure doesn't look like it got stuck in a suboptimal place.

In this one idealized handcrafted situation.

True, I have no guarantee it wouldn't get stuck in a local minimum under other circumstances. But the points I'm trying to make rely on existence proofs, so showing it works somewhere is enough.

High treedepth can compensate for multiple populations

Scenario Four: For each row, roll a four-sided die and flip twelve coins. Rolling a 1 makes it an A-row, and otherwise it's a B-row: don't keep that information in the explanatory variables, but do keep all the flips. The value for the response variable y starts at 1; each heads on flips 1-3 doubles the value of y for the row; each heads on flips 4-6 doubles the value of y for an A-row but halves it for a B-row; each heads on flips 7-9 halves it for an A-row but doubles it for a B-row; and each heads on flips 10-12 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. Despite no interactions between the features provided, the dataset generated thus will consistently get better scores with treedepth=3 than treedepth=1.

Table 4a: Example results

	MAE	RMSE
max_depth = 1, 1000 trees	1.1661	1.7386
max_depth = 1, 2000 trees	1.1661	1.7386
max_depth = 3, 1000 trees	1.1425	1.6981

You know, I have the strangest feeling of deja vu.

You're right, this is pretty similar to the previous section. But where that section addressed how multiple effects could be present in the same row, this addresses how multiple effects could be present across different rows.

Keeping with the examples we used last section: let's say you're trying to predict injuries caused by the exotic felines at a zoo, and some of them are lions and some are tigers, and the species differences are profound enough that they should have different models, but none of your explanatory variables outright tell you which is which. Or you're building an insurance model predicting claims from customer data, and some of your customers are Honest Citizens while others are Contemptible Frauds, but you can't tell in advance who's who. Or you're predicting people's heights from their other characteristics, but the collaborators who gave you the dataset didn't record what gender everyone is.

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

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

For this scenario, the system on its own terms looks like this:

$y = e^{(k_{0a}+k_{1a}x_1+k_{2a}x_2+ \ldots +k_{12a}x_{12})}$ with probability f
$y = e^{(k_{0b}+k_{1b}x_1+k_{2b}x_2+ \ldots +k_{12b}x_{12})}$ with probability (1-f)

So we can use gradient descent to find the best k-values, and end up with only twice the complexity of a normal linear model.

Just the k-values? What about f? Isn't it part of the model?

As a practical matter, I find varying f alongside the k-values tends not to work well, so it's best to use it as a hyperparameter instead of a regular parameter; then, we can gridsearch it, similar to how we'd get the best value for N or learning rate.

Table 4b: Example results from tree-based and canonical models (‘true’ value of f is 0.25)

	MAE	RMSE
Canonical model, N=2, f=0.05	1.3554	2.5590
Canonical model, N=2, f=0.15	1.3789	2.4893
Canonical model, N=2, f=0.25	**1.4160**	**2.4615**
Canonical model, N=2, f=0.35	1.5154	2.6799
Canonical model, N=2, f=0.45	1.5386	2.7053
max_depth = 1, 1000 trees	1.4628	2.6322
max_depth = 3, 1000 trees	**1.4181**	**2.4657**

That sounds awkward, especially for high values of N. If you wanted to fit ten subpopulations, you’d need nine degrees of freedom, which implies nine fs to gridsearch.

Setting N as high as 10 would be a bad idea anyway; the entire point of this is to let people understand the effect each feature has in a model, and making people look at ten numbers per feature (or, for the quantitative and categorical analogues of these binary variables, ten graphs per feature) would only be a slight improvement over making them look over hundreds of trees.

So what if there genuinely are ten distinct subpopulations in a dataset, and you genuinely could and should model them as distinct to get the best result?

Then we’re screwed, dear Reader; not all scenarios have inherently interpretable solutions. Speaking of which . . .

High treedepth can compensate for absent explanatory columns

Scenario Five: For each of 10000 rows, toss a coin and roll six ten-sided dice. Record the opposite of the coin flip on rolling a 1, and record the true result on any other result. For your dependent variable y, raise 2 to the power of the number of heads you got, then feed the result into a Poisson random number generator as in the last two scenarios. For your explanatory variables, take *only* the first three recorded coin flips. When using these explanatory variables to predict this dependent variable with an appropriately-linked and -distributed XGB model, treedepth=3 outperforms treedepth=1.

Table 5: Example results

	MAE	RMSE
max_depth = 1, 100 trees	7.4127	11.4690
max_depth = 1, 200 trees	7.4125	11.4692
max_depth = 3, 100 trees	7.0407	11.0973

I hope you realize I haven’t been paying attention to these complicated generation procedures.

Of course; I’m just providing them as an alternative/supplement to reading my hacky, uncommented code. What’s important is that you grasp what the setup is supposed to simulate: in this case, what

happens when you use trees to model an idealized log-linked system where some of the explanatory variables are absent, but are correlated with those present.

Uh huh. And I'm guessing you built some workaround where you use the explanatory variables you do have to predict the ones you don't, then use them all together to predict the response?

Actually, I was going to, but then I realised that would be a waste of time. For one thing, it'd just be impractical: you couldn't know a priori how many absent features there are, or whether they're categorical, continuous, or ordinal; and if any were categorical or ordinal, you'd have to gridsearch the number of categories or ranks for each absent feature.

But more importantly, even if you got it right, you'd end up with graphs deducing missing features from present features, and graphs using absent and present features to predict the response. This would obscure and complicate the effect of each individual feature on the final prediction to the point where I wouldn't feel comfortable calling the model inherently interpretable, and interpretability is the entire point of this exercise.

Instead, my solution to extra complexity caused by absent features in a log-linked system is "try to make sure important features aren't absent when modelling a log-linked system".

As plans go, that's up there with "try discovering the response variable directly instead of using models to predict it". If people could have obtained information that can help with model-building, wouldn't they have done it already?

Probably, though "improved performance!" and "reduced complexity!" together might open doors that "improved performance!" on its own wouldn't. And this effect isn't obvious: it's pretty counter-intuitive that using more features – even/especially features highly correlated with those present – can make models *less* complex.

Summary, Analysis, and Wild Speculation

So, we have our five results. What do they suggest?

Didn't you already say what you're trying to suggest with this essay in the introduction?

No, that list of intentionally weak statements are what these results *prove*. This section is about what they *suggest*, in a wink-wink nudge-nudge sort of way: all the wild speculations and grand theorising they lend credence to.

First, higher treedepth in the absence of real interaction occurs as a result of fucking up. Consider our findings:

- "High treedepth can compensate for insufficient learning": if you fuck up by undertraining the model, higher treedepth will help.
- "High treedepth can compensate for incorrect linkages": if you fuck up the model's linkage, higher treedepth will help.
- "High treedepth can compensate for multiple responses": if you fuck up by assuming single responses in a log-linked system where response is composite, higher treedepth will help.
- "High treedepth can compensate for multiple multiple populations": if you fuck up by assuming homogeneity in a heterogeneous population, higher treedepth will help.
- "High treedepth can compensate for absent explanatory columns": if you fuck up by omitting relevant data in a log-linked system, higher treedepth will help.

You realize that, except for the first one, that's all kind of inevitable? Like:

- *If you fuck up the model's linkage, higher treedepth will help. (All linkages are approximations)*
- *If you fuck up by assuming single responses in a log-linked system where response is composite, higher treedepth will help. (All responses are arbitrarily decomposable)*
- *If you fuck up by assuming homogeneity in a heterogeneous population, higher treedepth will help. (All models elide relevant heterogeneity)*
- *If you fuck up by omitting relevant data in a log-linked system, higher treedepth will help. (You will never have all relevant data) (If you did, you wouldn't need predictive models)*

I grant all of this: fucking up is inevitable and ineradicable. But we can still do better, and end up with less complex models, by fucking up less.

Second, if I've found five ways fucking up adds treedepth to a conventional-interaction-free model, that implies there are probably at least five ways I haven't found. There are many more bullets to dodge than those listed here.

Third, everything I've said about treedepth is probably true of model complexity in general. I've been sharpening my axe with decision trees in mind, but there are likely some neural nets, deep learners, and other monstrosities out there in Production which could have been linear models if their builders hadn't fucked up.

Fourth and finally, dodging and/or mitigating enough fuckups could mean a "decision stumps" GLM might only need a small number of legible interactions sprinkled on top to let it match or outperform much more complex models.

(At least, I hope it could, because I plan to spend the next year or so investigating that possibility, and I'll be sad if it doesn't work out.)

Other Panaceas

Dear Reader, permit me one last synthetic scenario before we part ways.

Table 6a: Hypothetical gemstone pricing system

	Size = Small	Size = Medium	Size = Big
Color = Red	$100	$200	$300
Color = Blue	$200	$400	$600
Color= Green	$300	$600	$2700

Start with a price of $100. Blueness multiplies it by 2. Greenness multiplies it by 3. Mediumness doubles price, and bigness triples it. Finally, there is an interaction in this log-linked system: on top of the other effects, bigness-and-greenness triples price again.

This system can – and probably should – be modelled in an interaction paradigm; it's exactly what XGBoost and its kin were built for. But as I've spent twelve pages of A4 demonstrating, you can model with the wrong paradigm and still get better results just from adding more degrees of freedom.

So, let's model this as a multiple/composite response problem, as in the third scenario. First, have one of the responses be "non-interaction response", modelling the system without the interaction:

Table 6b: Gemstone model, non-interaction response

	Size = Small	Size = Medium	Size = Big
Color = Red	$100	$200	$300
Color = Blue	$200	$400	$600
Color= Green	$300	$600	$900

Then, for "interaction response", start with $1800*$\varepsilon^2$, where ε is an arbitrarily small quantity. Greenness multiplies this part of the price by 1/epsilon; bigness does the same.

Table 6c: Gemstone model, interaction response

	Size = Small	Size = Medium	Size = Big
Color = Red	$1800 * ε^2	$1800 * ε^2	$1800 * ε
Color = Blue	$1800 * ε^2	$1800 * ε^2	$1800 * ε
Color= Green	$1800 * ε	$1800 * ε	$1800

Add these responses together and – ignoring the effects of arbitrarily small quantities, which are by definition ignorable – you have the system you started with.

Table 6d: Gemstone model, combined response

	Size = Small	Size = Medium	Size = Big
Color = Red	$100 + $1800 * ε^2	$200 + $1800 * ε^2	$300 + $1800 * ε
Color = Blue	$200 + $1800 * ε^2	$400 + $1800 * ε^2	$600 + $1800 * ε
Color= Green	$300 + $1800 * ε	$600 + $1800 * ε	$2700

The moral of this story isn't that you would be well-advised to model interactions this way. If nothing else, adding any penalisation to this modelling process would wreck it; and penalisation is important, especially when aiming for interpretability. Rather, I want to show that there are other ways to crank the complexity – and hence accommodate unaddressed fuckups – that can (at least theoretically) do everything trees do. If model complexity is a panacea, that doesn't mean a trilemma between low performance, illegible complexity, and having to perfectly anticpate every irregularity: we can apply extra epicycles in whatever forms best suit our needs.

Appendix: Failed Approaches

Briefly, here are two things I thought would increase optimal treedepth but didn't.

First, I couldn't get higher treedepth to cause improved performance when modelling highly nonlinear effects from continuous features. While I proved my hunch that high-treedepth models would learn the inner parts of a w-shaped feature effect at the same rate as the outer parts and

lower-treedepth models fixated on the edges, I couldn't engineer a scenario where that had a meaningful or reliable impact on model performance.

Second, while the presence of outliers – or, equivalently, the use of incorrect error distributions – appeared to benefit from higher treedepth, the benefit was only consistent for some metrics, and only when treedepth was already >1. That is: MAE for treedepth=10 would be consistently lower than for treedepth=5 in the presence of sufficiently severe outliers, but I couldn't engineer a scenario where it was consistently lower than for treedepth=1, and extra complexity didn't help with RMSE at all.

(Note: the ability of extra model complexity to lower MAE but not RMSE – by enabling it to ignore outliers – can explain some of the weirdness in table 4b)

I mention these failures mostly in case they're useful and to avoid publication bias, but also to show that there exist problems which do *not* increase optimal treedepth.