1:44

Alexey

**This week, we'll talk about Bayesian modeling and probabilistic programming. We have a special guest today, Rob. Rob is a machine learning engineer and a data scientist. He's interested in deep generative models, as well as good statistical models. Previously, he was a research scientist at Indiana University, where he worked on the Hakaru probabilistic programming language. Sounds Japanese, right?**

2:07

Rob

Yes!

2:09

Alexey

**Sudoku Hakaru. [chuckles]**

2:11

Rob

Yeah, yeah. It's actually a verb. Hakaru means “to measure”. It was a system that actually took measured spaces with modeling events seriously. The name also, in some sense, advertises unique features of the library.

2:33

Alexey

**Only if you speak Japanese (or understand it).**

2:35

Rob

Well, yeah. [chuckles]

2:36

Alexey

**[chuckles] Okay. The questions for today's interview were prepared by Johanna Bayer. Thanks, again, Johanna, for your help. Let's start.**

# Rob’s background

2:46

Alexey

**Before we go into the main topic of Bayesian modeling and probabilistic programming, let's start with your background. Can you tell us about your career journey so far?**

2:56

Rob

Yeah, sure. I basically started as a fairly regular software engineering and computer science person. I went to LA, worked in different startups and was getting involved just as machine learning was kind of really having… I don't want to say a renaissance yet, but it was starting to really pick up. What do I mean by that? What I mean is, open source machine learning tools were actually getting good. Because before, I would say, 2006–2007, lots of machine learning tools were very hit or miss. You had some nice stuff in the R ecosystem and you had libraries like OpenCV and…

3:55

Alexey

**[inaudible] Have you ever used it?**

3:57

Rob

But there really weren't… SciKit Learn was essentially only created in the late 2000s. It was actually very hard to do good machine learning work readily well. It was all researchers, so everyone was essentially just rewriting all the algorithms from scratch all the time. But they were starting to get pretty good and I was very much really engaging with machine learning. But I was often finding that a lot of the kinds of problems I wanted to work with didn't always fit things like SciKit Learn or things that you could just throw a random forest at.

So I got interested in Bayesian modeling. Through that, an opportunity showed up so I could work with a former university professor of mine, Ken Shawn, in Indiana. Over there, I did work on probabilistic modeling (Bayesian modeling) tools. Through that, I really came to really love and appreciate the power and flexibility of these tools. From there, I went and did a PhD program at Oxford, where I worked on more deep Bayesian modeling, so very much VAE sorts of things. After that, I moved to Berlin to kind of move back into industry to start actually applying these tools. In some sense, I was motivated by…

The problems I cared about with a lot of these tools are things that are very challenging to work on the side – they really are research problems that require dedicated time. I became very motivated to see these tools get widely adopted, and I also just enjoy solving actual problems. A lot of my trajectory has been this back and forth, where I sort of pop into industry, dip into academia, pop back into industry, pop back into academia. With this, I try to kind of push the technology forward.

6:30

Alexey

**You said you were… How did you phrase it? “A usual software engineer/computer science person”?**

6:38

Rob

Yeah.

# Going from software engineering to Bayesian modeling

6:40

Alexey

**I imagine that for a usual computer science/software engineering person, when they see all these mathematical formulas from the Bayesian modeling, they just freeze or freak out or whatever. I remember, for me, it was like, “Oh, all these integrals. I don’t understand what's happening here.”**

**How did it happen for you? Was it easy for you to get into the field with your software engineering/computer science background? Or did you have to do a lot of extra work to be able to work and not be afraid of all these formulas?**

7:23

Rob

Sure. I might be a little bit not the typical case, because I always had a love for machine learning and AI. I kind of always knew that I'd need to be comfortable with integrals and things like that. There are some people who were software engineers, they were used to making things like web servers and mobile apps, and then they're like, “Calculus?! I thought that was useless. You mean this math I was taught in secondary education is the most important thing I could have known? Why did my teachers not say that?” I didn't quite… [cross-talk]

8:10

Alexey

**I think they did. [chuckles] They did say that.**

8:12

Rob

If I wanted to use these methods, I had to at least be somewhat comfortable with optimization methods and linear algebra. There's kind of no way to do that without being at least a little comfortable with taking a few integrals, taking a few derivatives. So it was less of a challenge in that regard.

8:35

Alexey

**Did your job as a software engineer…? As I understood, you started your career as a software engineer, and then you transitioned to more Bayesian stuff? [Rob agrees] Did your work as a software engineer involve mathematics, or was it creating websites and doing backend stuff?**

8:56

Rob

No, no. It was very much doing things like “Scrape data from the web, clean the data, put it in a database.”

9:10

Alexey

**So fairly standard software engineering.**

9:12

Rob

Yeah. Not so much as even calculating a median.

9:17

Alexey

**Okay. [chuckles] So you kind of kept your, if I can call it “mathematical muscles” fit by doing some extra stuff, or did you just like mathematics so much?**

9:32

Rob

It was just dabbling. You're just sort of dabbling in the problems, you're reading about them. Because in some sense, if you're like, “Oh, I want to learn machine learning. Okay, I'm going to learn how to run a random forest or implement it.” You just sort of spend a lot of time on the side getting comfortable with things, making little programs.

This was back when there weren't these very beautiful boot camps, which really, I think, try to structure a lot of this knowledge. So I was just dabbling. I've told this to people – I've never taken a formal statistics class. Ever. I never did. I just basically kept buying statistics textbooks until I felt comfortable enough with the ideas.

10:24

Alexey

**Interesting. For me, I remember my first formal machine learning class was a bit of a shock – maybe not a bit of a shock, but actual shock. It was at TU Berlin. The professor started with Introduction to Bayesian modeling, with a lot of integrals and… the things we talked about, like marginalization, he just assumed we knew all this stuff. It was like, “Okay, what am I doing here?” [chuckles] But I decided to stay and see what happens. It was a good decision. But yeah, it was… It was scary.**

**For you, from the university times, you liked mathematics. You were curious about that. You were reading statistics textbooks. This is how you kept your mathematical skills, right? For you, this transition from software engineering to more mathematics-heavy stuff was not uncomfortable. Right?**

11:32

Rob

Right. I was already reading these things like, “This is how you marginalize. And this is…” [cross-talk]

11:38

Alexey

**I guess this is why you transitioned, right? Because you wanted to do more of that stuff – not scraping.**

11:43

Rob

Yeah, in some sense I kind of already did, but… This was the thing where it was, at that point… Thinking about 2006, 2007, 2008 – there weren't machine learning engineer jobs. That wasn't a thing that existed. There were analysis jobs. But this idea that you will be in a tech company doing machine learning was… Those jobs existed here and there. They might exist in Google, if you want to help with spam finding or working on the ranking algorithm.

That was a job where you use machine learning, but you'd be basically hired as a software engineer. You can sort of still see it in the culture of a lot of companies that haven't really come back and reflected on roles. It's just that they hire software engineers, and then they hope they can teach you statistics later. That's a culture that I think still persists quite a bit.

12:50

Alexey

**I'm pretty sure about that. I see that [as well]. Then there is an opinion that it's easier to learn the math you need than the engineering. So it's easier to hire an engineer and then teach the math (the basics), and then let them work on ML, rather than getting them in the position to be able to engineer. I don’t know if it's true – I've seen good people…[cross-talk]**

13:14

Rob

Yeah. It’s not clear if it's true, because I've gotten to experience the opposite, which is – you have a bunch of, to put it bluntly, unemployed physicists that know the math, and basically are taught software engineering. In my personal experience – they do just fine.

13:33

Alexey

**I'm sure that if you know all the… Physics involves a lot of mathematics, right? All this gravitational stuff. The variety of symbols there, and Greek letters, and all that stuff – it's even scarier than integrals, right? And they understand all that. If they understand all that, and can make sense of that, and spend five years defending PhDs about that – for them, learning Java is not that difficult. Right?**

14:02

Rob

Right. Right. It feels like it’s such a silly… I don't know. I suspect that the reason this attitude is taken is more because software companies are run by people who used to be software engineers. That's what they value. So their attitude is, “Software is what I know, so software is not what I expect to teach. Statistics is something I don't know. I kind of had to dabble and learn. So that's what I'll expect from the people I hire.” I feel like it actually just flows downstream from those kinds of unstated values.

# Frequentist vs Bayesian modeling approach

14:43

Alexey

**What is Bayesian modeling? What are we talking about today?**

14:47

Rob

Right. Okay. Yeah. Also, I've been saying the words “probabilistic modeling” and “Bayesian modeling” – they're not quite the same thing. There are differences. If you take a statistics class… Maybe this has changed. But if you take a statistics class, at some point… It feels a little bit silly to say this considering I've never actually formally taken one, but I read a bunch of textbooks that are used for these classes. So hopefully, I'm not coming off too presumptuous. Okay, let me say this – if you open up a statistics textbook, in the introduction one, they'll often say, “There are these different schools of how to do statistics.” What they'll basically say is that there's this sort of frequentist modeling approach and a Bayesian modeling approach.

15:36

Alexey

**My statistics class started with that, yeah.**

15:39

Rob

Okay, good, good, good. Alright. I don't want to be like, “Oh, this is how class starts.” And everyone’s like, “No. No one does that anymore.” They shouldn't. But, you know. [chuckles]

15:49

Alexey

**It was a long time ago, though. But yeah, I confirm.**

15:53

Rob

So they'll say there's frequentist modeling and there's Bayesian model. What do they mean by this? What they mean is, in a frequentist modeling approach, there's an assumption that you have a parameter that you're interested in. So you're interested in, say, something like annual rainfall in Germany. This is an actual quantity that exists in the world, but there's no way to directly measure that. But what you can do instead is measure things that are sort of noisy… the result of processes downstream of that.

You can do things like measure soil, or measure the increase of water in sewer systems. These aren’t annual rainfall, but these are essentially quantities that you can then use to estimate annual rainfall. There's, of course, some noise or some variants. A lot of what frequentist modeling is about is, “We're going to essentially do these data collecting processes for this, and then use that to estimate. Because there's some noise in that, we can't say this is what annual rainfall is. We’ll essentially give you a confidence distribution, meaning here's what we think it is and here are some bars of where we think the value is.” So you'll say things like, “We're 95% confident that the true estimate of annual rainfall will be between these two values.”

And frequentist statistics is nice in that regard, in the sense that these confidence values mean something. What they mean is, “If you do 100 experiments where you gather data and then estimate the rainfall, 95% of the time, your estimate will be between those two bars.” There's a really good guarantee there. But that's frequentist statistics, which I don't really do. [chuckles] But there's also Bayesian statistics. Bayesian statistics actually takes a very different attitude. In Bayesian statistics, there is no point estimate at any part of the analysis.

18:41

Alexey

**By “point estimate,” do you mean we make an experiment?**

18:46

Rob

There’s just a number where you say, “This is what I think the annual rainfall is.”

18:49

Alexey

**Yeah. So we get a number and this is our point estimate of the rainfall? [Rob agrees] And then we put some bars around it to say, “Okay, this is the interval.” So then it's still a point estimate, right?**

19:06

Rob

Right. There's a point estimate, and then there's a confidence interval around it. There are things that use distributions in frequentist statistics, but for Bayesian modeling, the fundamental object is a distribution. You almost never are doing anything with point estimates. Everything has a distribution. Before you even begin, you don't even talk about “Oh, I want annual rainfall.” You say, “I have a prior distribution. I have a starting distribution of what I think annual rainfall is.” What you then do in Bayesian modeling is collect data. And then, through basically applying Bayes’ rule, you end up with a posterior distribution of rainfall, given the data you've now observed. So it's always distributions – you're always working with distribution. What’s the advantage…

20:21

Alexey

**So let's say I have some assumptions about the annual rainfall in Germany. I could have zero clue of what it is – it could be, “Okay, it's a uniform distribution between X and Y (some numbers).” I have no clue. But this is the starting point, right? [Rob agrees] And then I go and dig a hole and then measure something – I get some data. I go look into the sewers and learn some other variables, right. I collect data, and then from the flatline (from the uniform distribution) I start getting something else. This is what I do with Bayesian modeling.**

21:02

Rob

Yeah, with a Bayesian model, exactly. You would start with some examples like that. I would hope you wouldn't use a uniform, because you've collected some data, you've looked at some annual rainfall estimates over the years, and you said, “Oh, it seems that sometimes there's less, sometimes there's more.”

21:27

Alexey

**It all depends on what kind of data we already have – what kind of knowledge we already have.**

21:31

Rob

Yeah, of course. But the important thing is that it's always a process from one distribution to another. Now, why would someone prefer that? It seems like more work. And it is work – doing this Bayes’ rule over and over again often involves computing a lot of these annoying integrals that are often not efficient. What's the actual advantage here? Because it's distribution in, distribution out, it's a very composable framework. It's very easy to extend your analysis and build up an analysis. Everything is just distribution in, distribution out. So it's very easy to sort of…

Here's the thing – I have this distribution on data. That can be the starting point for a future analysis. I want to add more variables into the model later? I can start with this one, add in a little bit more data. It's very composable in that way, where it's very easy to sort of start from a simple analysis, and incrementally build into a more complicated one. Whereas, often, in frequentist statistics, you can't really do that – you often have to kind of throw away a lot of your old work. What this often leads to is, because you can build up your analysis, you can write tools that do this compositional work for you. It becomes much more amenable to these sorts of universal solutions where you can write a model, derive a sampler from it, and get to work. So in that sense, it's very useful in that regard.

23:41

Alexey

**And how is it related to probabilistic programming?**

23:45

Rob

Probabilistic programming are effectively tools to make doing Bayesian modeling significantly easier by removing all of the tedious steps.

24:00

Alexey

**The tedious steps are computing integrals?**

24:02

Rob

It's computing the integrals, or approximating the integrals by sampling, it's figuring out which integrals you have to compute when you write your model – it's all these little steps that are sort of straightforward to do, but they're very tedious if you have to do them every time you make a little change to your model.

# About integrals

24:29

Alexey

**Why is computing integrals such a big deal? Why don't we just compute them? What's the problem with them?**

24:39

Rob

The issue with integrals is –if you go into your Introduction to Calculus class, there's this impression of, “You're going to write this integral, you're going to write this formula, you write an integral – Look, it has this nice clean form.” But, of course, the tragedy there is that this is actually not the common case. The idea that I could just write a function and get a nice integral from it, is not what actually occurs. They just don't often exist. In some sense, it’s the opposite of the case you have in derivatives, where there are very nice ways to automatically calculate derivatives. When you use PyTorch, you can just write a function and be handed, essentially, another function that computes the derivative. No such thing really exists in integrals. So, what people have to often do is approximate that. What do I mean by approximate? If you remember, when people talk about Riemannian, where you make these little bars, and then you sum the bars. People effectively have to do things like that.

26:02

Alexey

**You mean compute the integral numerically, instead of computing it analytically, when deriving the formula [cross-talk]… You cannot do this because for some cases, there is no rule that you can apply here. So the only thing you can do here is just use numerical methods, which involves summing from one thing… I remember, usually integrals would be from minus infinity to infinity, and then you’re like, “Okay, how do I compute numerically? It can be computationally intensive. Correct?**

26:40

Rob

Right. But effectively, this is what we're kind of doing. We're going in, and computing numerically. Often, what happens when you're sampling – you're approximating an integral, because you're like, “Okay, I have this curve. I'm going to use probabilities to land on different parts of the curve and I'm going to count.” Effectively, by counting them, it's almost like building a histogram, right? You're making each of the bars, and you're counting where each of the bars are. A lot of times when you're sampling, you're essentially just approximating this integral. That's a lot of what the regime is. In Bayesian modeling, because most of these integrals are intractable, we devised the sampling methods.

27:37

Alexey

**By intractable, do you mean that we cannot compute them on a piece of paper?**

27:42

Rob

We can take a formula, get back another formula, and then, essentially, pass the bounds of integration and do minus and then repeat if we have nested ones. That's an option that really doesn't exist for the kind of integrals most people care about, sometimes provably so. If you look at some of the integrals that are involved for these probabilistic models, you're getting things like the error function. People are like, “What's the error function?” “It's this integral.” “Does have a closed form?” “We've never found one.” Or “We've proven that one can't exist.” Or “They use these nice set of functions.” So you have to do these numerical methods – you have no choice.

# Probabilistic programming and samplers

28:35

Alexey

**Again, if we go back to our discussion of Bayesian versus frequentist – in frequentist, we get a number and then a confidence interval, and that's all we get, right? [Rob agrees] In Bayesian, on the other hand, we start with a distribution, we end with a distribution, but because we deal with distributions, we get all these nasty integrals that we have to deal with somehow.**

**This is why we use probabilistic programming – because otherwise, if we just directly use numerical methods to compute these integrals with these things you mentioned, like histograms, it just takes too much time. That's why we have approximation methods? To do it faster?**

29:17

Rob

Right. Well, there's two things. One, without programs that do integrals, for every model, you have to essentially figure out what the sampler is going to be and then write it. If you make your model a little different, you have to change the sampler – you have to rewrite it.

29:37

Alexey

**What’s a sampler?**

29:40

Rob

A sampler is a way to sort of figure out these integrals by basically… You're going to sample from the distribution in a particular way from the posterior distribution and that's essentially going to a bunch of draws, which you can then take the empirical average of. That's going to essentially let you approximate the integrals that are involved for…

30:10

Alexey

**So posterior distribution is the result – the thing we're interested in. We cannot easily calculate it because there is no formula, so we need to somehow figure out what the value is, right? That's why we need sampling. There are different approaches for computing this integral These are the samplers. [Rob agrees] The Monte Carlo simulation is one of the possible samplers. Right?**

30:37

Rob

Yeah. What happens is… Sampling is its own challenge. What you quickly realize is, if you try to do this the naive way, most of your time – if you just said, “I don't know where my distribution is. I'm just gonna guess randomly and then check,” You will find that, particularly in high dimensions, most of your guesses are going to be in low probability regions.

31:02

Alexey

**I have no idea what this sentence meant. [laughs]**

31:05

Rob

Well, if you're trying to guess rainfall, and you know that it's some number between zero and infinity, you might find that you often spend a lot of your time generating a guess that has low probability.

31:23

Alexey

**So I think, “Okay, maybe it's 1000 (whatever unit it is).” But it's a very slim chance that it's actually 1000. Regardless of what number I choose, the chance that I'm correct to still… [cross-talk]**

31:41

Rob

Right. You could use your prior distribution, like that uniform example, and use that as a guess. Then, you would essentially do a reweighting. That's better than guessing completely arbitrarily. But if you think about it, that’s just in one dimension. Now imagine in, say, 1000 dimensions or a million dimensions.

32:01

Alexey

**A dimension, in this case, would be… Rainfall would be one dimension, right?**

32:09

Rob

Well, you're trying to estimate rainfall for not Germany, but every city in Germany.

32:16

Alexey

**Oh, okay.**

32:17

Rob

And there are clearly relationships between them, so a guess for one is probably not a bad guess for others. But if you're just guessing arbitrarily, the chance that you're going to pick a thing with reasonably high probability just goes down tremendously.

32:36

Alexey

**If we would pick only the five largest cities in Germany, then the probability that we pick the rainfall for each of them correctly is non-existent. Okay, we might get lucky, but if we add another city to that, with each city…**

32:52

Rob

Right – as you keep adding, as you increase the dimensionality, the chances that you're gonna do okay… Of course, you can keep running samples and hope that eventually you hit high probability regions. But that takes computation time and [cross-talk].

33:10

Alexey

**By running sampling, you mean something like, “Okay, we have these five cities. Let's just roll five dice.” We just open our NumPy and ask NumPy, “Okay, give us five random numbers between zero and infinity.” And we'll repeat until it's correct, right? That's what running sampling means?**

33:35

Rob

Yeah, it could take a while to actually get answers that have high probability.

# MCMC and Hakaru

33:43

Alexey

**How do we even know that these are correct if we just guess them?**

33:48

Rob

Our model essentially returns to us, “What's the probability the parameter has this value, given the data we return?” So we have these probabilities there. But we can't know in advance what the high probability regions are going to be. There are tricks. You can sort of take the maximum – the maximum a posteriori of this function – and say, “Okay, what particular set of parameters gives me the top one?” But again, that's a point estimate. You still need to look at what the other likely values are. Also, the maximum a posteriori can be deceptive for lots of problems. All of these fancy sampling algorithms are all tricks to make it so that when you sample, you're likely to be sampling from a high probability region. That's where you get things like MCMC. What does MCMC do? It says, “Well, pick a starting point and make a sample from that.”

35:00

Alexey

**Monte… What was it?**

35:03

Rob

Markov chain Monte Carlo. So “Monte Carlo” because it's your sampling, it's gambling – you're throwing dice, and the “Markov chain” because each sample that you pick is dependent on the previous one. There’s this kind of Markov chain of all the samples. Why does Markov Chain Monte Carlo work fairly well, particularly high dimensions? Because you assume that your high probability points are near each other. So, for annual rainfall, if we think something like “50 centimeters of annual rainfall” is a good estimate, and has a good probability, we expect that 49 is not too bad either.

You start there, and then your next sample is near there, and that you make a move that's in the region. In that way, you sort of hope to stay in the high probability regions of the space. You can say, “Well, this sampling algorithm has shortcomings, too,” so we make a better one, and so on, so forth. But the whole development of sampling algorithms is all for the purpose of getting points that are in high probability regions of the space without spending too much computational time doing so.

36:33

Alexey

**And that's what you did too, when you developed the Hakaru language, right?**

36:39

Rob

Oh, we didn't actually develop new sampling algorithms for Hakaru. We just took some of these algorithms that were out there and made them so that the users didn't have to run the algorithms. The user just ran the model, and they got the algorithms. That's the main thing that every probabilistic programming language is doing. It's just about letting someone talk about their model and writing the sampling algorithm for them.

37:12

Alexey

**And what do we actually mean when we say it's a programming language? There’s Java, there’s R, there is Python, there is Hakaru. Right? So you actually write code there. [Rob agrees] It’s an interpreter – a different environment. It’s not Python, right?**

37:31

Rob

Right. Well, it's not quite Python, but it's…

37:36

Alexey

**[It’s] for these sorts of things that we have been talking about, right?**

37:40

Rob

Yeah. What we mean by that is – when we think about statistical models, sometimes you'll think about the Professor writing some notation on the board, saying, “I have some parameter – it comes from this distribution. If I see this value, it comes from this.” But if you just kind of pause and step back a bit and think about what a statistical model is – think about… Let’s just keep using just the annual rainfall example. I might say, “I expect a certain amount of rainfall in a region just because it's in a particular latitude and longitude, and I expect some regional variation.” Why do I expect some regional variation? “Well, some towns are near a mountain and mountains affect things. Some towns are just maybe a little more south, so it's a little warmer, so they get more [rain].”

If you think about how you would write that model, thinking about, “I have a loop, I iterate over cities, I have, if/can statements that I check, whatever mountain. Some of these parts of the model might be tedious, I might create subroutines. Oh my god, we have a programming language. This is clearly a programming language.” Just by virtue of trying to talk about what you're modeling, you inevitably are using a sort of language to do so. So from my perspective, the way we talk about modeling *is* the programming language. So what do we do with problems…

# Language vs library

39:28

Alexey

**In the same sense that we can say SciKit Learn is a programming language because there are… What's the difference between a library and a programming language is, I guess, what I’m trying to ask.**

39:38

Rob

I mean, that's a very beautiful question. If you talk to some people out of the University of Utah or some of the racket scheme people they'll say, “This distinction between a library and a language is not very black and white. There is a little bit of an in-between.” So I think the distinction can be a little bit… But I'll make the case that there's this notion of “ifs” and “subroutines,” and “looping primitives,” these are things that we associate with languages, not libraries. A looping primitive isn't a thing you use within SciKit Learn, it's the thing you use within Python, which you then call “SciKit Learn From”.

So when we talk about a statistical model, it's not just that we're using an “if” in Python. When I talk about, “Oh, I think the rainfall should be higher if a town is near a mountain,” that actually affects how we compute the probability of there being rainfall in a particular town. So it's not just that we've run a simulation, it actually has consequences into how you compute the log probability. And if it has consequences like that, then it's like, “Okay, we're not really evaluating this if we're interpreting it differently.”

But if you're interpreting differently, you're not in Python anymore, right? Clearly this is living in a different space at that point. So I think it's actually useful to say, “Oh, this is its own language, which we can embed in Python, or R, and what have you. But it's definitely its own language, with its own semantics.”

41:45

Alexey

**And Scikit Learn is not, because we don't use “ifs”… [cross-talk]**

41:49

Rob

We're just doing these API calls. There's no notion of a new “if” or anything or a unique notion of a control flow.

42:00

Alexey

**Well, it's still a little bit abstract for me, to be honest. Maybe I should read those Git docs. You said that there is the University of Utah, who are really into programming languages. [cross-talk]**

42:14

Rob

Yeah. In papers there were things like “libraries of languages” and things like that. You import a library, and you get a new language or you have a library that slowly turns into a language. So there are these notions that you can… The distinction isn't… There is a gradient between these two. But from my perspective, Scikit Learn has an API – it has function calls, it has data types, and they can kind of be moved together. It is a language in a certain sense, but it's a very restricted language. Because it's so restricted, I think most people are comfortable calling SciKit Learn a library more than calling it a language.

PyMC

43:05

Alexey

**Okay. What's PyMC?**

43:08

Rob

Right. PyMC is one of the major policy programming languages/libraries in Python.

43:18

Alexey

**So just libraries. [chuckles] Interesting. Okay. [chuckles]**

43:20

Rob

Well, it's a library because it’s a Python library, but it very much has its own language. When you write a model in PyMC, you get a computational graph back – you get an AST. What you can inspect… When you write models you aren't… You use Python code to write them, but what that Python code piping code is actually doing is building up this computational graph, which is effectively an AST.

43:52

Alexey

**Okay, I am lost. How does PyMC… Can I say program… look like? Let's say we want to estimate the annual rainforest in Germany and we want to use PyMC for that. What does our program look like?**

44:14

Rob

The way it would look is, you would say… Let's just pick a very simple example. Let's use a simple model just as an example for now. We're gonna assume that there's a certain annual rainfall that we expect in Germany, and we're gonna assume that all the cities are essentially that rainfall plus a little noise (which isn't the worst assumption). The way you would write that in PyMC is, “Germany rainfall equals normal, (the mean rainfall, the standard deviation we expect on the rainfall”. Now, that's not exactly how you write it because the syntax in PyMC requires that you put a string, so you name it… [cross-talk]

45:17

Alexey

**It’s very difficult to explain it just by talking.**

45:20

Rob

At a high level, this is effectively what you do. Now, you might say, “Rainfall is never a negative number,” but I will handwave that and say that, “We’re going to use a good sampler, so we're going to try never to sample negative numbers.” [cross-talk]

45:37

Alexey

**So you say “If the sample is less than or equal than zero, then sample one more time.”**

45:43

Rob

Yeah, we can do something with that. Also, PyMC has a thing called the “half-normal,” which is just the normal but a part of it is just chopped off. We can do that as well. But we're going to do that, and then the next line is going to be, “We're going to sample annual rainfall for every city.” That's just going to be rainfall for cities equals normal, left parenthesis mean (the global mean we just estimated), comma some variance (we have so much variance we expect for each of the cities to have) comma shape equals number of cities, right parenthesis.

At that point, we're then going to add another thing, which is the sewer levels for each city. We're gonna say the sewer levels – there's some functional relationship between them and we're gonna essentially call that annual rainfall thing – pass that through that function to get them. That's what we're going to actually observe. Because what we observed is how much sewer levels rose for each of the cities.

47:05

Alexey

**So what we're doing here is some sort of linear regression, but instead of just output for each city, we get a distribution, right?**

47:15

Rob

It's not quite a linear regression, right? Because we cannot…

47:20

Alexey

**We have some features – this is stuff from the sewer…**

47:24

Rob

Because we have this parameter that we're going to be using to try to keep all the rainfall values near this global value, because… [cross-talk]

47:32

Alexey

**That’s why it's kind of, sort of, conceptually, like linear regression – or more like just regression. Because we predict – we output a number (or a bunch of numbers, because it's a bunch of this). Conceptually, it's a regression, except it's not linear regression. Right? [Rob agrees] It’s a different sort of regression model that outputs probabilities – not probabilities, distributions. And we have this ability to say that, “This is the global mean and it shouldn't be too different from the global mean.**

48:10

Rob

Exactly. I wouldn't quite call it regression, because you can represent regressions in PyMC. You can say that you have some x, you have some y, and we have some relation between them (this function). This function has some weights. I don't know what the weights are – we're gonna put a distribution. So you can do a regression. This isn't quite that, but yeah, the intuition is that you can have this model that can say, “There is rainfall. There's a relationship between rainfall in the cities and the sewer tunnels. There's a relationship between rainfall between cities, because they're kind of near each other.” For all those, you can just readily encode them.

This is a three-line model, but there's already quite a bit of richness there. Of course, that's just the model. Then you probably want stuff – you want some quantities to come out. You can then recall PyMC, that sample, to now get estimates of all the city rainfalls. You might also care about the annual one – you might ask for that one as well, since you've already modeled it. When this runs as samples, you will now have a posterior distribution on rainfalls for each city, *and* for just this estimate of what the rainfall is in the country as a whole.

At that point, that's pretty good. In practice, you wouldn't just stop there. You might actually pull up statistics of actual previous estimates people had of rainfall, and if your estimate was too far from these government records, there's something probably wrong with your model and you would make the model slightly different, you might add other parameters than just measuring sewage rises. But over time, your estimates would start to match the kind of estimates that others had and you can start to have confidence that you've seemed to have captured something.

50:29

# Encoding dependencies and relationships into a model

Alexey

**From what I understood, from what you said, PyMC, and probably other probabilistic programming languages or frameworks – what they give us is the ability to express our problem in some sort of language, and express dependencies between different things. [Rob agrees] We can say there is a… can I say locality dependence? Cities that are close probably shouldn't have too different… than the global… [cross-talk]**

50:59

Rob

Right. The model I just said…

51:03

Alexey

**Yeah, so in your case, they shouldn't be too different from the global. [Rob agrees] Which kind of captures that. You can make it more complex, saying that cities that are close…**

51:17

Rob

Yeah. You can start doing like spatial models, where you say, “Cities that are close to each other should have similar annual rainfall.” And there are ways to sort of encode that.

51:29

Alexey

**Like how far it is from the sea, right?**

51:34

Rob

Right. You can encode how far you are from the sea – you can start quickly building up. What's nice is… What does it mean to “build it up”? You just add another line of code, hit “sample” again. The process is much more natural. You're not saying, “Oh, I want to model what it means to be ‘more far from the sea’. Now I have to sit, recalculate by hand. I have to check if I can still use the sampler from before. Am I doing something where suddenly my old sampler’s not good? Do I need to make a hack to get around that?” These are all things that used to be the norm. Now, people, by and large, write something in PyMC or other programming languages – things like Hakaru, Turing, Stan… There are many of these sorts of…

52:28

Alexey

**And if I compare it to, let's say, (classical?) frequentist approaches – when we, say, have a regression problem with random forest or whatever. Then, our features would include all the things we talked about plus, maybe, how far the sea is, right? So if we want to introduce one more thing in our model (in our program), like the distance from the sea – in the frequentist case, we just would add another feature “distance from the sea”. In the case of probabilistic programming, we would model it differently, right?**

53:12

Rob

Yeah. It might be [something] like you add another feature, because you're doing something like a Bayesian neural net, and you just add another weight – you put another prior on that weight – and then go from there. You just say, “This is just going to be ‘distance from the sea’ and how important this weight is.”. So you can do that as well. But if you're trying to do something where you're doing these groups, it actually becomes a little bit tricky to do. So if you're trying to do things like, say, dynamically turning on and off groups of features – that's a thing that's actually quite subtle to do using machine learning algorithms.

In a probabilistic programming language, that's just another line of code – you're not even thinking about it. You're just like, “Yeah, let's group fix! Who cares?” You're not even thinking about, “Oh, my God! This tiny change I made has severely changed the algorithm and I can't use the one in SciKit Learn anymore.” Also, we should keep in mind that, by and large, for these prediction algorithms, there's no distribution. It's just, “Here are my inputs. Here are my outputs.” Maybe, if you’re lucky, you might get a confidence score on that output. And if the confidence score is too low, you might say, “Okay, maybe we should be careful with the prediction.” You don't get this distribution where I say, “You've given me this input. Here's a distribution on the values.”

This becomes particularly relevant when you start getting these multimodal distributions, where you can say, “There isn't one good answer. There are two good answers that are very different from one another.” Because you might say things like, “Well, when there's a drought, the rainfall is very different than when there's not.” If you're just asking me what the rainfall is, I can say, “Well, in drought years, it tends to be this number – in normal years, it tends to be this number.” That is a thing that's very easy to express in a Bayesian modeling framework and very difficult to express outside of it.

# Stan, HMC (Hamiltonian Monte Carlo) , and NUTS

55:24

Alexey

**When you were talking about different probabilistic programming languages (frameworks), you mentioned Stan. There is a question from Johanna, “What do you think about Stan?” So, what is Stan and what do you think about it?**

55:41

Rob

Sure. Stan, I think I can comfortably say this, is the main and most predominant probabilistic programming system out there. It's a pioneer in many ways, which I'll explain shortly. It's a leader in the space. It's the “main” one. If you only look at one, you should look at Stan, because it is the main and dominant one. Why is that? Part of what that is – there were probabilistic programming systems before Stan, but Stan did two things. One, it introduced HMC (Hamiltonian Monte Carlo) into popular use. For the moment, I'm not gonna go into what makes HMC special, other than to say that it's a significantly better sampling algorithm than all the others that were previously in use. It is so much better, that if your problem doesn't fit HMC, the advice is “figure out a way to make it work in HMC”. It's that good of a sampler – it changes the gravity around the way you model things.

So if you're using Stan, you're just getting better estimates, better results than if you're using other tools before Stan came along. This trend continues – innovations in sampling algorithms often first appear in Stan before they appear in other systems. This isn't fully the case, but this is the case often enough that it’s true. The best practices established in Stan set the tone to best practices used elsewhere. If an algorithm gets introduced in Stan, it'll likely appear in PyMC within a few months, and others. There's a strong leader role there. The second reason, of course, is that the algorithms that Stan users tend to have a lot of their own hyperparameter tuning, so they work out of the box. You're not doing lots of fiddling.

There really is just this button called “sample” that you hit and you get samples out. You're not doing these little tweaks where you're playing with little parameter numbers to kind of make it work. HMC has existed for decades, but it requires you to know these very particular arguments that you had to pass in – and if you got the arguments wrong, it didn't work at all. But getting them right was very fiddly. Stan added features where they figured out what the correct arguments were to pass to HMC. So when you have things, it's called the “no U-turn sampler” (NUTS). The main thing NUTS is doing is taking HMC and figuring out the right parameters to use in HMC for your specific problem. Because of that, you essentially had a sampler that was very efficient and required basically little feedback to work. Yeah, I think Stan’s great. I'm not gonna say anything other than that. [chuckles]

59:28

Alexey

**Yeah, I was just checking the questions. The reason I was asking about PyMC is because you’ve contributed to it – you're a contributor to this library. Thanks a lot for doing that. Do you have a couple more minutes before we finish?**

59:45

Rob

Of course! Of course, yeah.

# Sources for learning about Bayesian modeling

59:46

Alexey

**Great. So we have a course – it's a machine learning engineering course – and the way we do it is half of the content is about machine learning and the other half of the content is about engineering. The half that’s about machine learning, as you can imagine, is about the frequentist approach – linear regression, then we talk about logistic regression, then we also cover tree methods, and then cover neural networks.**

**I imagine that some students will listen to our chat right now and some of them might wonder, “Okay, we now have learned these frequentist approaches, but I want to try something Bayesian.” What would you suggest they do? What's the easiest way for them with the background they already have to try something Bayesian, maybe with PyMC?**

60:47

Rob

Sure. There's a book I like… This answer has gotten much easier in recent years, because people have really started to produce good materials for this. Before, I would have said, “Oh, you should just buy a book on Bayesian statistics.” I'm gonna still say that, but the books are better now. I'll suggest two things. There's a book on Bayesian computation that was written by a lot of the PyMC people – there’s Ravin, Osvaldo Martin, and a few other people that I'm blanking on. The book’s freely available online and you can sort of work through it and use PyMC to explain how to do Bayesian statistics.

There's also an online book that's a little older, called Bayesian Statistics for Hackers – it also uses PyMC. But if people like a course environment, there's a really amazing course taught by Richard McCaffrey, who out of Leipzig, called Statistical Rethinking. He has an online course – previous versions of the course are freely available on YouTube so you can just watch the course online there. He has a book. The book isn't free, but the book’s very good. His book is really amazing. It's hard for me to… I’d be hesitant to say there’s anything wrong with the book for what it does. Because what it does is – it doesn't just teach how to do Bayesian modeling and Bayesian statistics, it teaches how to do statistics. A lot of the challenge of doing statistics is often not, “Oh, should you do frequentist or should you do Bayesian?” It's often, “Are you capturing causality properly? Are you collecting data in a good way, where you'll actually be able to learn something and prove your models?” It really kind of takes this big, holistic view ,which I think is… [cross-talk]

63:01

Alexey

**It doesn’t explain why one is better than the other, it just says, “Okay. Here are the problems. Here are the things you need to think about. Here are the solutions (the tools you can use).” Right?**

63:12

Rob

It really teaches the workflow, which is really where I think statistics really can be lacking. Too much of the old ways it was taught were cookbooks, effectively, “Here's your problem at the factory. Use this recipe from the cookbook.” Which, because of the dynamic way we work with data… The idea that you could do a cookbook for data scientists, I feel it's overly optimistic, for a lot of the challenges people face. Where I think this book shows its strength and it says, “Here's the attitude you should take for how you should be doing statistics. And if you have this attitude, you'll figure out the correct steps you need to do to get an analysis that's useful and actually informative.” So I recommend the Statistical Rethinking book/class, and particularly for PyMC, I recommend the Bayesian Computation book. [cross-talk] All the programs that were written in Stan for the Statistical Rethinking book have been ported to PyMC by enthusiastic volunteers.

64:40

Alexey

**I found… The first book is called Bayesian Modeling and Computation in Python, with the authors Martin, Kumar, and Lao. I posted the link. Then the second one, Statistical Rethinking, the author is Richard McCaffrey. [Rob agrees] I just posted the link. The examples are in R and Stan.**

65:16

Rob

Right.

65:19

Alexey

**As you said, enthusiasts ported everything from that book into PyMC.**

65:22

Rob

Yeah. The book is so good that students find it and then port the examples to PyMC.

# Reaching out to Rob

65:32

Alexey

**Yeah, we should be wrapping up. It was amazing. Thanks, Rob. You made some things clear to me, personally. Sometimes you would say something and I would just think, “What is that?” But now it's clear. Thanks. Before we wrap up, is there anything you want to mention?**

65:53

Rob

Yeah. Maybe a little bit of a plug. I do statistical consulting – that is kind of the main thing I do these days. I write software as well to help with that. For anyone who's listening to this, if you or your company have statistical challenges, are actually interested in maybe applying Bayesian modeling for your problems, reach out to me. I'm always interested in helping people out on this.

66:24

Alexey

**What's the best way?**

66:26

Rob

The best way is just to email me – rob@zinkov.com.

66:31

Alexey

**Okay. We will also include the email in the description. I posted two links in the live chat – I will also post them in the description. I guess that's all for today. Thanks a lot, Rob, for joining us today. And thanks, everyone, for joining us today too.**

66:46

Rob

Thank you.

66:49

Alexey

**Have a great week. See you around.**