

Probabilistic programming makes Bayesian inference accessible. It allows you to build interpretable models that quantify uncertainty and make use of both data and domain knowledge. In this report we show how to use probabilistic programming to build tools and products for more effective decision making.

# Probabilistic Programming



*Fast Forward Labs*

# Probabilistic Programming



*To the future—*

*Copyright © 2016 by Fast Forward Labs*

*<http://www.fastforwardlabs.com>*

*New York, NY*

## Contents

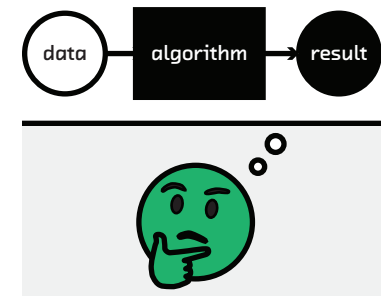
1 Introduction.....	7
2 Bayesian Data Analysis.....	11
2.1 Data and Probability.....	12
2.2 Incomplete and Imperfect Data.....	13
2.3 Bayesian Basics.....	14
2.4 Bayesian Inference.....	16
2.5 Bayesian Posteriors.....	19
2.6 In a Nutshell.....	21
2.7 Uses and Advantages.....	22
2.8 Conclusion.....	33
3 Probabilistic Programming.....	35
3.1 A Naive Sampling Algorithm: Rejection Sampling.....	36
3.2 A Better Sampling Algorithm: Metropolis-Hastings.....	41
3.3 A Contemporary Sampling Algorithm: Hamiltonian Monte Carlo.....	43
3.4 A Sampling Algorithm for the Future: ADVI.....	47
3.5 Putting It All Together: Probabilistic Programming.....	48
4 Prototypes.....	53
4.1 When to Use Probabilistic Programming.....	53
4.2 Loans.....	54
4.3 Real Estate.....	74
4.4 PyMC3.....	85
5 Tools and Products.....	87
5.1 Probabilistic Programming Tools.....	87
5.2 Probabilistic Programming in the Wild.....	94
6 Ethics.....	101
6.1 Choosing the Prior.....	102
6.2 Interpreting and Communicating the Posterior.....	105
6.3 Managing Data Privacy.....	106
6.4 Conclusion.....	108
7 Future.....	109
7.1 Probabilistic Sci-Fi: BayesHead 5000.....	110
8 Conclusion.....	115
Appendix: Frequentism.....	117

## CHAPTER 1

### Introduction

In the last several years we've seen remarkable advances in the capabilities of machine learning systems. Many of these advances, however, have come in the form of techniques that use very large data sets and immense computing power. Accuracy has been more important than interpretability, resulting in black box systems that can classify the world more accurately than ever before, but cannot explain how or why the world works the way that it does.

These black box techniques have yielded, and continue to yield, amazing results. But because it's not clear how they use data to make predictions, they have a number of limitations. It's difficult to feed in preexisting knowledge about the problem. When the prediction is wrong, it's often not possible to say why. These predictions don't necessarily come with associated confidence in the result. As a result, the trained model is unlikely to reveal much about the nature of the problem.



**FIGURE 1.1** *Black box techniques can be powerful but difficult to interpret*

These limitations make it difficult or impossible to make models that work with only a small amount of data and leverage domain-specific expertise. They also adversely affect models in dangerous or legally complicated contexts such as health or banking where models that yield predictions must come with confidences that allow one to assess risk. Lifting these limitations opens the door to new kinds of products and analyses, and is the subject of this report.

The solution is a statistical technique called **Bayesian inference**. This technique begins with our stating prior beliefs about the system being modeled, allowing us to encode expert opinion and domain-specific knowledge into our system. These beliefs are combined with data to constrain the details of the model. Then, when used to make a prediction, the model doesn't give one answer, but rather a *distribution* of likely answers, allowing us to assess risks.

Bayesian inference has long been a method of choice in academic science for just those reasons: it natively incorporates the idea of confidence, it performs well with sparse data, and the model and results are highly interpretable and easy to understand. It is simple to use what you know about the world along with a relatively small or messy data set to predict what the world might look like in the future.

Until recently the practical engineering challenges of implementing these systems were prohibitive, and required a large amount of specialized knowledge. Recently, a new programming paradigm, **probabilistic programming**, has emerged. Probabilistic programming hides the complexity of Bayesian inference, making these advanced techniques accessible to a broad audience of programmers and data analysts.

In this paradigm, ideas of probabilities and distributions of results are basic building blocks of models. Thanks to advances in the robustness and speed of inference algorithms such as NUTS and ADVI (discussed in detail in [3 Probabilistic Programming](#)), probabilistic programming has made these inference tasks that used to require months to years to implement into an afternoon of work.

Some innovations, like using Deep Learning for image analysis, are an entirely new capability and something that simply was not possible before. Others, like the probabilistic programming techniques discussed in this report, have historically been very difficult and have recently become widely available and easily accessible. This rapid change in capability is likely to be just as impactful.

Probabilistic programming languages make the process of specifying your beliefs and inferring the implications of your data in the context of these beliefs much simpler. It offers several advantages. First, you can actually use your existing domain knowledge in your model, allowing you to combine data with what you already know or believe. Second, it works well with small or noisy data sets, or multiple data sets that must be combined to give a single result. Third, it's interpretable, meaning that you will always understand why the system returned the result that it did. And finally, it's easy to use in online or active learning, meaning that you are not necessarily constrained by long training cycles.

This report explores these advantages in more depth, along with real-world applications, the commercial and start-up markets, the open-source community, and likely future directions.

## CHAPTER 2

# Bayesian Data Analysis

The main goal of this chapter is to offer a conceptual introduction to the Bayesian approach to data analysis, and to list its many advantages and use cases. We'll start with A/B testing as a concrete example in which we explain and apply the Bayesian method. We'll then survey use cases in a variety of fields. Bayesian data analysis is made faster and more accessible by probabilistic programming, which we'll discuss in the next chapter.

### 2.1 Data and Probability

Imagine that we are conducting an A/B test for an e-commerce website. Our goal is to determine whether layout A or layout B of our website yields more sales. We run the test for a week. During that time, 4% of the visitors shown layout A convert (i.e., sign up or buy something), while 5% of visitors shown layout B convert. Layout B's conversion rate is better, so it seems like layout B is better.

But this naive approach has the potential to go badly wrong. Perhaps we showed layout A to 100 people, 4 of whom converted, but we only showed layout B to 20 people, 1 of whom converted. One out of 20 is 5%, but would you be confident saying B is better than A now? Probably not! Even if we had run a larger experiment (perhaps showing layout A to 1,000

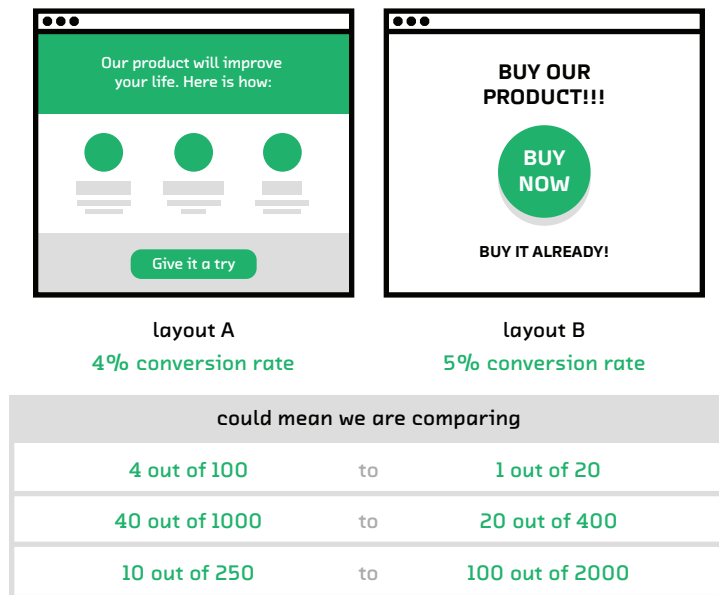


FIGURE 2.1 Confidence in the results of an A/B test should depend on the size of the sample

visitors and layout B to 400 visitors), there is still a chance that simply comparing the rates could lead us astray. To make robust conclusions that we can build a business on, we need to be able to quantify our confidence in our conclusions.

To do that, we need to think about our data probabilistically.

## 2.2 Incomplete and Imperfect Data

The fundamental problem of data analysis is that real-world data is incomplete and imperfect.

Data is incomplete because its collection is time-consuming, expensive, or even impossible. In the example we just

looked at, we ran the A/B test for a week. A business can't wait a year to find out the results of such a test. Sometimes, even if we can afford the time, it's impossible to store all the data. For example, we cannot store every packet a network handles. And sometimes data is missing because it's impossible to measure: we can't observe whether somebody likes a movie they haven't seen, or the value of a house that hasn't been put on the market.



FIGURE 2.2 Working in the real world means dealing with imperfect data

Even if it can be collected and stored, data is often imperfect. Physical sensors can be miscalibrated or broken; users sometimes behave anomalously (perhaps the kids are using the family Netflix account); and data can be corrupted or go missing.

There are two main approaches to determining the implications of incomplete, imperfect real-world data: *frequentism* and *Bayesianism*. We discuss the frequentist approach in [Appendix: Frequentism](#), but this report is about the Bayesian approach.

## 2.3 Bayesian Basics

To learn from data the Bayesian way, you need three things: a generative model that describes a process you can observe, prior beliefs about the details of that process, and



some real-world data that is its outcome. Bayesian inference takes these three things as input and yields updated (or *posterior*) beliefs. Those posterior beliefs about how the world works can then be used to deploy a model, or, if you collect more data, they can themselves be used as input for another round of inference. Let's discuss these inputs and outputs in more detail.

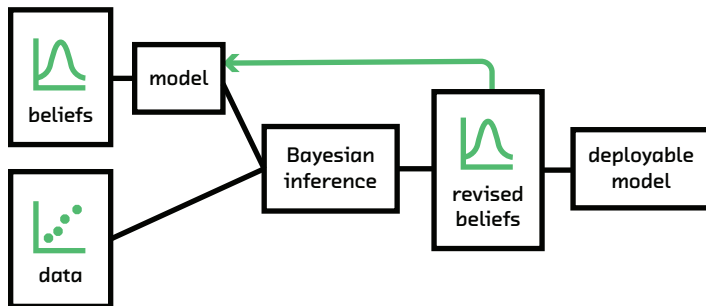


FIGURE 2.3 The Bayesian data analysis workflow

A *generative model* is a process that simulates how we believe data is generated in the real world. For example, if we are conducting an A/B test, our generative model might be: each visitor who is shown layout A has a particular conversion probability, and each visitor who is shown layout B has a different conversion probability. We don't need to know these probabilities in advance; they are the unknown *parameters* of our model and are precisely what we're trying to determine. If we know them, then we will be able to say whether layout A or layout B is better. For more complicated problems, the generative model might have many more parameters or a much

more hierarchical structure.

*Prior beliefs* are certainties or suspicions about the values of the parameters of the generative model. For example, in the A/B testing scenario, perhaps layout A has been live for a few months and therefore we have some domain-specific knowledge about it: we think its conversion rate is about 3.5%. We're not so sure about this that we'd rule out it being 2% or 5% with certainty, but we think those are both pretty unlikely. But if layout B is completely new, we might genuinely have no idea what its conversion rate is, other than that it must be at least 0% and at most 100%. Our prior beliefs about the conversion rates for layouts A and B might then look something like [FIGURE 2.4](#).

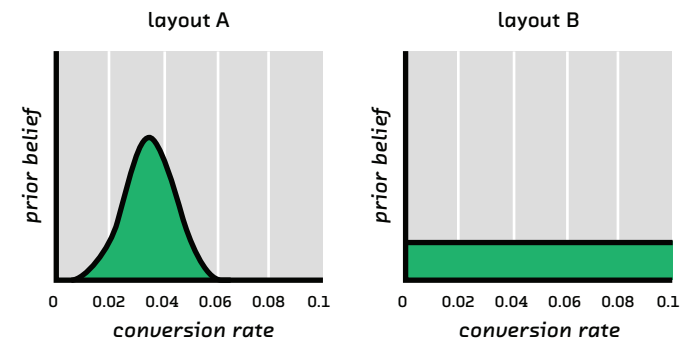


FIGURE 2.4 Prior beliefs

Finally, we need some *data*. In the context of A/B testing, that means recording which layout each new visitor saw, and whether or not they signed up.

## 2.4 Bayesian Inference

Common sense tells us that a hypothesis is more likely to be true if:

1. Experience or knowledge means we already suspect it is true.
2. The new data we collect make sense if we assume that it is true.

Bayes's rule is a simple mathematical expression that formalizes and quantifies this idea. With it, the three inputs mentioned above (the generative model, prior beliefs, and the data) can be used to learn from observations. Mathematically, the rule can be written as shown in [FIGURE 2.5](#).

$$\underbrace{P(H|D)}_{\text{probability the hypothesis is true given the data we have}} \propto \underbrace{P(H)}_{\text{probability the hypothesis was true before we had data}} \underbrace{P(D|H)}_{\text{probability of observing the data assuming the hypothesis is true}}$$

FIGURE 2.5 Bayes's rule

On the righthand side,  $P(H)$ , the probability a hypothesis is true, is the mathematical version of statement 1 above — i.e., the extent to which we already believe a hypothesis.

Also on the righthand side,  $P(D|H)$ , the probability of observing the data if the hypothesis is true, is the mathematical version of statement 2 — i.e., a measure of how much sense

the data makes if we assume the hypothesis is true.

Bayes's rule tells us that simply by multiplying these two things together we get a measure of  $P(H|D)$ , the probability that the hypothesis is in fact true.

Bayes's rule is a very natural way to think about the implications of imperfect, incomplete data. And crucially, it connects the two things on the righthand side of the expression, which we can calculate, to the thing on the left, which is what we usually need to know.

A hypothesis is a particular choice of parameters.  $P(H)$  in Bayes's rule is therefore our prior belief about how credible that choice of parameters is, before we make any observations. We can calculate  $P(D|H)$ , the *likelihood* of the data given the hypothesis, by simulating the process described by the generative model. With  $P(H)$  and  $P(D|H)$  we are able to infer  $P(H|D)$  — that is, we can measure the probabilistic implications of our data.

Let's go back to A/B testing. In that case, the possibility that the conversion rate for layout A is 2.9% is an example of a particular hypothesis. Suppose our prior belief of that particular hypothesis,  $P(H)$ , is 0.1 ([FIGURE 2.4](#)).

Suppose also the data shows that 40/1,000 people who were shown layout A registered. We now need to determine  $P(D|H)$ , the probability of that outcome, assuming the conversion rate is 2.9%. In this simple case it's possible to write down a mathematical formula for the answer, but that isn't always possible, and as we'll see later, probabilistic programming means that it is never necessary. We can simply simulate the process using a computer. Regardless, it turns out that the probability of getting 40/1,000 registrations if the conversion

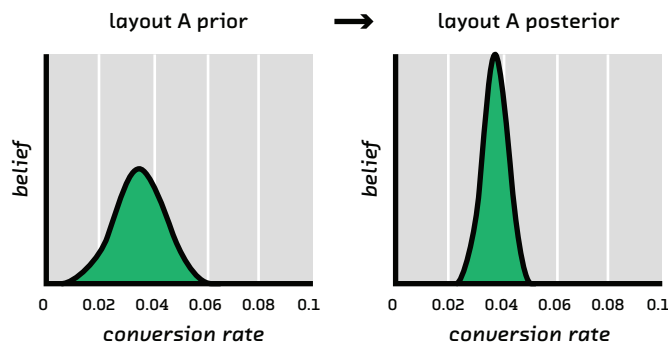


FIGURE 2.6 Prior + data yield a posterior for layout A

rate is 2.9% is about 1 in 100. Which is to say:  $P(D|H) = 0.01$ .

We now multiply our prior belief that the conversion rate is 2.9% by  $P(D|H)$  to get a measure of  $P(H|D)$ . This is what we want: it tells us how likely the hypothesis is to be true, given our prior belief and the data. The actual value of  $P(H|D)$  is not as important as its value *relative to all the other hypotheses*.<sup>1</sup> To figure that out we have to perform the same calculation (determine the likelihood of getting the observed data, and multiply that by our prior belief) for every other possible value of the conversion rate. This gives us the relative probability of every possible value of the conversion rate of layout A. This is the *posterior distribution*, the most important product of Bayesian inference (see [FIGURE 2.6](#)).

<sup>1</sup> In fact, to get  $P(H|D)$ , rather than just a number proportional to  $P(H|D)$ , we would need to normalize Bayes's rule. This usually involves integration and can be difficult, if not impossible. Fortunately, it is unnecessary to solve our A/B testing problem.

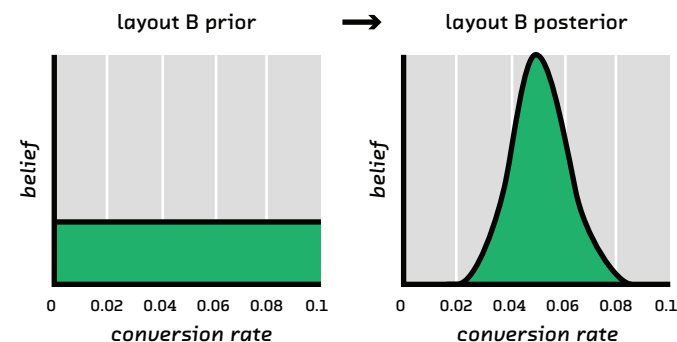


FIGURE 2.7 Prior + data → posterior for Layout B

To compare layouts A and B, we need data for layout B too. Suppose 20/400 people shown layout B registered. By combining this data with our *uninformative* prior belief about the conversion rate (it could be anywhere between 0% and 100%), we can calculate the posterior distribution for the conversion rate of layout B using the same method we used for layout A ([FIGURE 2.7](#)).

## 2.5 Bayesian Posteriors

Posterior distributions are what we use to answer concrete questions about the implications of our imperfect, incomplete data. For example, to determine how likely it is that layout B is better than layout A, we sample at random from the two posterior distributions, and record whether the criterion "B is larger than A" is true. This is repeated a large number of times. The fraction of times the criterion is true gives us the probability that B is better than A—i.e., the answer to our question. In this case, that's about 86%, which means we're

86% certain layout B is an improvement (but there's a 14% chance it's actually worse — perhaps we should run our A/B test for longer!).

There is nothing special about the question "Is layout B better than layout A?" For example, perhaps B is promising, but we are worried about a worst-case scenario, so we want to know the probability that its conversion rate is less than 1%. Or perhaps we want to report a traditional confidence interval for layout A to our boss. To do that, we might want to determine the interval in which we are 95% certain its conversion rate lies. Or perhaps we want to know the probability that layout B is more than twice as good as layout A. We can use these two posterior distributions to answer these and pretty much any other questions about A and B.

Note that the Bayesian method does not give a simple yes or no answer: it gives a probability. This is a feature, not a bug. Again, real-world data is incomplete and imperfect; there isn't a single right answer, and approaches to data analysis that suggest there is give a false sense of security. But Bayesian inference gives us the next best thing: we can say that, more likely than not, A is better than B. And more importantly, we can say precisely how likely it is that this is true.

Suppose we improve the A/B model by taking into consideration how users arrive at the website or their demographics. This improvement might yield tighter constraints on the conversion rates, but there is still a range of possible values defined by a posterior distribution.

Our data is neither perfect nor complete, which means our inferences will never be exact. But we can, if necessary for reporting, reduce our posterior to a single number (such as the

mean of the posterior distribution) or a confidence interval. For example, we can say that the most likely value of the conversion probability for layout A is 3.9%.

## 2.6 In a Nutshell

Let's take stock. We started with a generative model, a set of beliefs about credible values of the parameters of that model, and some data. By the process of Bayesian inference, those beliefs have been updated and our new beliefs are captured by the posterior distribution. Those new beliefs can be used to answer arbitrary, concrete questions about the parameters of the model, as we've seen. Or they can be used as prior beliefs for further rounds of inference, if we are able to collect more data.

This process of updating prior beliefs in the light of data is, in a very real sense, a redistribution of credibility. In our

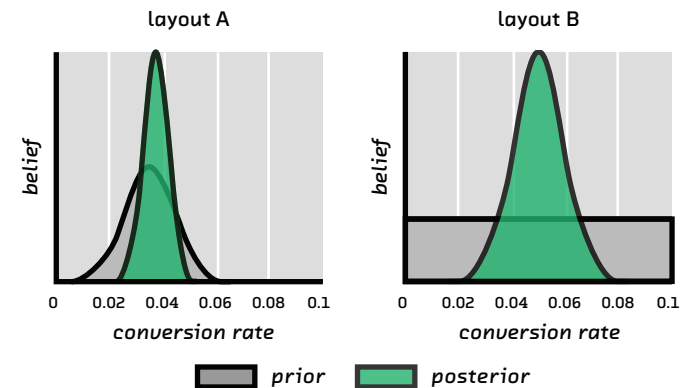


FIGURE 2.8 The posteriors for layouts A and B overlaid on their priors

A/B test example we had relatively strong beliefs about credible values for one parameter, but no prior beliefs at all for the other. Under the Bayesian approach, if our prior beliefs are strong (as with layout A), the data has a weaker redistributive effect than if the prior beliefs are weak (as with layout B). This is why the prior and posterior distributions for layout A look quite similar, but the prior and posterior distributions for layout B look very different ([FIGURE 2.8](#)).

## 2.7 Uses and Advantages

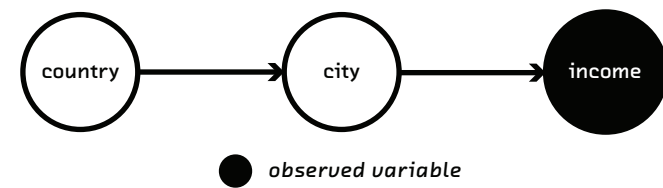
Having defined Bayesian inference, we'll now list its advantages and in so doing illustrate a range of concrete contexts in which it is particularly effective. This is a long list, because Bayesian inference is a powerful and generic approach. But all the advantages below derive from a central idea: Bayes's rule. The power of Bayes's rule stems from the fact that it relates a quantity we can calculate (the likelihood that we would have observed the measured data if the hypothesis were true) to one we can use to answer arbitrary questions (the posterior probability that a hypothesis is true given the data).

### 2.7.1 User Segmentation and the Generative Model

The structure of the generative model is almost arbitrary. The only requirement is that it allows us to calculate the likelihood of our data (i.e., the probability of getting the observed data, given a particular choice of parameters). The likelihood need not be analytic—something we can write down as an equation—as this would severely constrain our options. As long as we can simulate the behavior of the model, we can calculate the probability of a set of data.

This freedom allows us to learn about complex systems with interdependencies, which are known as **hierarchical models**. In such a model, we observe the behaviour of individual events, but we incorporate the belief those events can be grouped together in a hierarchy.

This is exactly the approach we use in one of our prototypes ([4.3 Real Estate](#)). In that model, apartments are in neighborhoods and neighborhoods are in boroughs. Not all apartments in a neighborhood are alike, but they are on average similar to each other. And neighborhoods are on average more similar to others in the same borough than elsewhere. So, the average West Village apartment is different than the average East Village apartment, and very different from the average Brooklyn apartment. This is a hierarchical model. By modeling this way, we can learn about the West Village not only from the West Village, but also from the East Village and Brooklyn. As we'll see in [4.3 Real Estate](#), this can be very useful when data at a particular level of the hierarchy is sparse, e.g. there are no sales in a particular neighborhood in a particular time period.



**FIGURE 2.9** Hierarchical models allow you to group observations together and learn from their similarities

A similar geographical hierarchy (e.g., country, city, etc.) can be applied to user segmentation. We could add in more levels if they are justified by our data or product (e.g., demographic groups). We discuss how to make this decision in [2.7.6 Time Series and Model Comparison](#). The point though is that we can use the Bayesian approach to model an arbitrary system. We don't need to look through a zoo of statistical tests and special cases to find one that looks kind of like our problem.

### 2.7.2 Healthcare and Prior Beliefs

Priors allow us to insert institutional knowledge or logical constraints. We witnessed this in the A/B testing scenario: in the case of layout A we had some institutional knowledge, while in the case of layout B we had only the weak constraints offered by logical certainties (the conversion rate must be between 0% and 100%).

We can see the power of this ability to draw on prior beliefs in the realm of healthcare. For example, most people do not have cancer. We therefore have a strong prior belief that a randomly chosen person does not have cancer. In the Bayesian approach, this strong prior means that a single suspicious blood test wouldn't set alarm bells ringing. The prior reduces the chance of false positives, which in this case would result in needless worry and expense, and perhaps even risky diagnostic procedures.

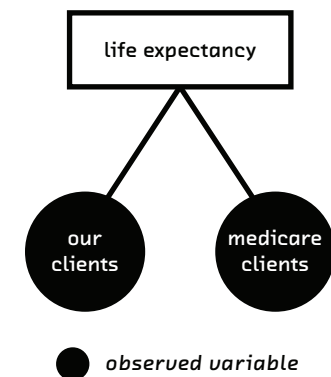
If you really do have no prior beliefs then you can use a "flat" or uninformative prior, as we used for layout B in the A/B

testing example.<sup>2</sup> A flat prior in cancer screening would imply that a random person has a 50% chance of having cancer. More often than not, however, we can do better.

### 2.7.3 Insurance and Multiple Datasets

The relatively simple relationship between data and model means that it is easy to infer the collective implications of several independent datasets. The datasets need not be measurements of the same thing taken at different times. They can be truly distinct and heterogeneous.

For example, consider risk assessment for life insurance. Perhaps our goal is to determine the current life expectancy of a particular demographic group. Naturally we know the ages at death of all our customers who were in that group. But perhaps we're worried that people who take out life insurance are a biased group, so we'd like to use some other data. Bayesian inference allows us to combine our proprietary data with another dataset that



**FIGURE 2.10** You can learn about the thing you are interested in from two or more datasets

---

<sup>2</sup> Under a flat prior, the Bayesian approach reduces to the classical maximum likelihood problem: which choice of parameters is most likely to have generated this data?

also depends on life expectancy. For example, one option would be to use publicly available information about Medicare payments, which also depend on this factor. By pooling these two very different sets of information, we can make a better (more precise, less biased) measurement of the quantity we're interested in.<sup>3</sup>

## 2.7.4 Finance, Risk, and the Posterior

The posterior distribution is rich with information. The most likely value can be used as a "best guess" (e.g., the most likely value of the conversion rate for layout A is 3.9%), but the full posterior distribution is an implicit measure of how probable every possibility is, both in an absolute sense and compared to every other possibility.

This allows us to quantify risk and, in so doing, control exposure to financial downsides while maximizing the profitability of a strategy. The ability to say how likely it is that a possibility is right or wrong is a powerful one in all areas of business, but perhaps the most vivid illustrations are in finance, where rare but catastrophic outcomes are important.

One of the prototypes we built for this report discusses this in the context of consumer lending (see [4.2 Loans](#)), but let's touch briefly on an investment banking example here.

---

<sup>3</sup> A famous example of parameter estimation based on multiple heterogeneous datasets is the measurement of the fundamental properties of the universe by cosmologists [see, e.g., <https://arxiv.org/abs/1502.01589>]. They combine data from many different types of astronomical observations to better constrain the age of the universe, and the amount of dark matter and dark energy it contains.

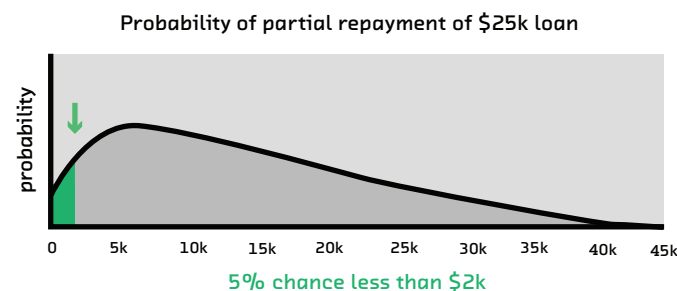


FIGURE 2.11 Bayesian inference allows us to quantify the probability of a catastrophic outcome, such as being repaid only a small fraction of a loan

The Sharpe ratio is a metric used to compare trading strategies. The Sharpe ratio of a given strategy is the ratio of the mean of return to the variance. A high Sharpe ratio implies high return and low variance, both of which are good.

Our "best guesses" of the Sharpe ratios of two strategies might imply that strategy X is better than strategy Y. But by comparing the posterior distributions of these two quantities (much as we did in the earlier A/B testing example), we can deduce the probability that this is true and therefore quantify the risk that we are wrong—which could be a very costly possibility.<sup>4</sup>

## 2.7.5 Small Datasets, Truncated Experiments, and Online Learning

As discussed in [2.6 In a Nutshell](#), for a given set of prior

---

<sup>4</sup> See Thomas Wiecki's talk "Probabilistic Programming in Quantitative Finance," available at [https://youtu.be/MeKucat\\_gw8](https://youtu.be/MeKucat_gw8).



beliefs, plentiful data has a stronger effect on our new beliefs (i.e., the posterior) than a small amount of data.

Bayesian inference allows us to feed in data one observation at a time, updating our beliefs iteratively. In the A/B testing example, we could have done our analysis by reapplying Bayes's rule after observing each visitor, constructing a new posterior distribution that would act as the prior for the next visitor. The final conclusions at the end of the experiment would have been the same, but the online approach would have allowed us to build a live dashboard of the progress of our experiment while it was running, and provided us with a measure of our current confidence that one layout is better than the other. We could even have decided to stop the experiment early if things were going better or worse than some predetermined threshold.

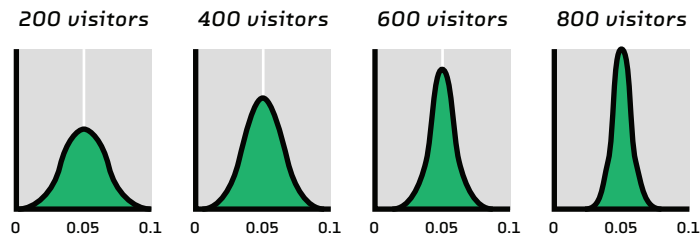


FIGURE 2.12 Our beliefs get stronger as we run the A/B test for longer

This ability to feed in data one observation at a time is particularly valuable when data is scarce. It allows you to smoothly transition from a data-poor world where you operate on hunches and heuristics encoded in the prior to a data-

rich world where your beliefs become dominated by what you learn from real-world observations. This is a great way to bootstrap machine learning in a new product or startup about which you don't yet have much data.

### 2.7.6 Time Series and Model Comparison

In the applications discussed so far, we've used Bayesian inference to determine the parameters of a model whose basic structure is fixed. But we can also use Bayesian methods to compare models. A more complicated model can always do a better job of fitting data than a simple model, but Occam's razor and common sense tell us that we should prefer a simpler model that does a reasonably good job. This preference is quantified and formalized in Bayesian model comparison.

A natural example comes from change point analysis in a time series.<sup>5</sup>

We'd like to know when and how big the change point was (that's parameter estimation), but we'd also like to be confi-

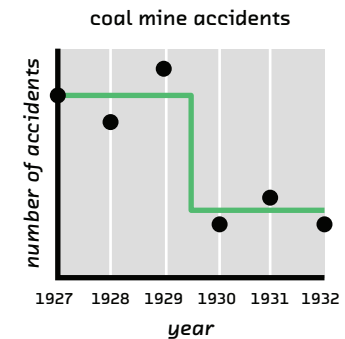


FIGURE 2.13 A change point identified in the rate of accidents in coal mines. For more examples using this data set see <https://pymc-devs.github.io/pymc/tutorial.html>

<sup>5</sup> A change point is an abrupt jump in the value being recorded.



dent one occurred at all (that's model comparison). The Bayesian approach allows us to quantify the evidence for a model that contains a change point compared to one that does not (or compared to a model that contains several change points).

Change point analysis crops up in many temporal data analysis problems. You can use a Bayesian model comparison approach to answer questions like whether there is evidence for a jump in sales, or a drop in hardware failures, or a change in website traffic reported by a social media monitoring application. And, as we discuss in [2.7.9 Coherence and Rigor](#), that same Bayesian approach can be used for other things as well.

### 2.7.7 Recommenders and Active Learning

Bayesian inference is also a powerful tool with which to build active learning systems, or human-in-the-loop machine learning systems. More data generally makes the predictions of a machine learning system more confident, but it can be time consuming or costly to acquire. In an active learning system, the algorithm itself makes decisions about whether to incur those costs before making a final decision. A Bayesian system can do this because the posterior tells it not just the current best guess, but also how likely it is that we are wrong and precisely which data would make us more confident.

An example of where this could be useful is in recommendation engines. A Bayesian active learning system could ask each user the required minimum number of questions during sign-up that would ensure that product recommendations reached some confidence threshold. This approach provides a framework for machines to collaborate with humans, making machines more efficient and human work more valuable.

### 2.7.8 Regulated Environments and Interpretability

In regulated industries such as healthcare and finance, it is important to have interpretable models, or models where it is possible to understand why they generated the results they did. There may be a legal or ethical duty to explain the basis for a decision. This can be difficult or impossible if the decisions are based on black-box machine learning models such as neural networks.<sup>6</sup>

The generative models that are the foundation of the Bayesian approach are interpretable by construction. The same attributes that allow an active learning system to determine which piece of missing data would make us more confident about a decision also allow us to say which pieces of information are responsible for a decision.

This is in stark contrast to deep learning, where explanations for a decision may require complex ad hoc analysis, if they are even possible.<sup>7</sup>

### 2.7.9 Coherence and Rigor

The final advantage of the Bayesian approach is its

---

<sup>6</sup> See our previous reports *Deep Learning* (FF03) and *Summarization* (FF04) for more on neural networks.

<sup>7</sup> Nonparametric Bayesian approaches (such as Gaussian processes, which we don't address in detail in this report) combine some of the flexibility and power of neural networks with some of the interpretability of the regular Bayesian approach to machine learning. Note that "nonparametric" approaches should really be called "massively parametric"; the number of parameters grows with the size of the dataset, and is thus potentially very large.

conceptual coherence and rigor. This is not an academic luxury. Coherence means that a model can be used to answer multiple questions, including those you didn't know you had. Rigor means that you can count on the answers being correct, or you can quantify the probability that they are not. Together these attributes simplify development and enable rapid experimentation.

We've seen examples of unanticipated questions already. As mentioned in [2.7.5 Small Datasets, Truncated Experiments, and Online Learning](#), we can use a Bayesian model not only to decide if layout A is better than layout B, but also to decide when to stop the A/B test. In [4.2 Loans](#) we'll see a model that was designed to predict the outcome of individual loans, which was then used to build a working prototype that allowed comparison of multiple loans. The time series models mentioned previously in the context of model comparison and change points ([2.7.6 Time Series and Model Comparison](#)) could also be used to detect anomalies, should they arise.<sup>8</sup>

The rigor of the Bayesian approach derives from the fact that confidence (or its absence) is a first-class output, not an afterthought. The ability to say how likely it is you are *wrong* is extremely powerful! In the same way a complete test suite gives you the confidence to change your code in traditional software development, probabilistic confidence estimates give you a similar freedom when doing machine learning.

---

<sup>8</sup> See our blog post "Probabilistic Programming for Anomaly Detection" (<http://blog.fastforwardlabs.com/post/143792498983/probabilistic-programming-for-anomaly-detection>) for more on this important use case.

The ability to quantify confidence is a form of rapid feedback that can be measured at all times during development and deployment.

## 2.8 Conclusion

This chapter explained the Bayesian approach to modeling incomplete, imperfect data, and enumerated a long list of real-world advantages. But we skipped over a lot of the computational difficulties. For example, in [2.5 Bayesian Posteriors](#) we answered concrete questions by simply "sampling" from posterior distributions. But it turns out that sampling isn't so simple! More generally, Bayesian inference can be both difficult to implement and computationally expensive.

The next chapter introduces the subject of this report, probabilistic programming. New probabilistic programming languages remove the barriers to entry to the Bayesian approach, such that what is in a sense the gold standard of machine learning is now a much more practical proposition.

## CHAPTER 3

# Probabilistic Programming

The previous chapter painted a very rosy picture of Bayesian data analysis. In that ideal world, we could declare to our computer what we think is true, show it examples of the real world, and use Bayesian inference to update our beliefs. The reality, however, is that real-world Bayesian data analysis suffers from practical problems related to both computational efficiency and developer usability. In some cases these problems are show stoppers. Probabilistic programming is an attempt to democratize the Bayesian approach by solving these problems, making the approach simpler and computationally tractable in many more situations.

### 3.1 A Naïve Sampling Algorithm: Rejection Sampling

We'll start by looking at a problem that crops up throughout Bayesian analysis: sampling from a known probability distribution. Sampling is the process of generating observations that conform to that distribution. For example, the outcome of a roll of a six-sided die has a known probability distribution. If we roll the die many times, we are sampling from that distribution. Sampling is something we need to do often (and efficiently!), both when applying Bayes's rule and when asking questions of the posterior distribution afterward.

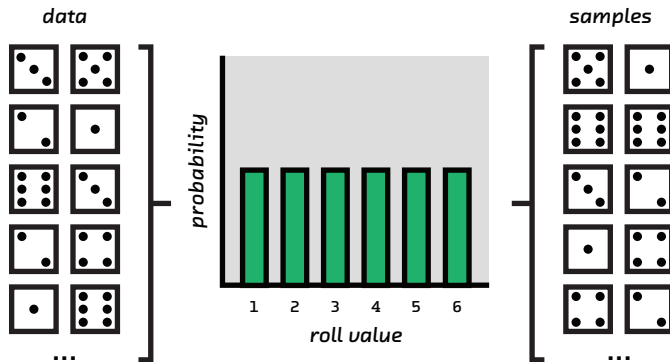


FIGURE 3.1 Samples generated by your model should resemble real world data

Efficiently simulating the outcomes of rolls of a die is easy. That's because sampling from a uniform probability distribution with a computer is computationally inexpensive.<sup>1</sup> But with a slightly more complicated probability distribution, it quickly becomes expensive.<sup>2</sup> Imagine we want to generate a sample of baby names. We have a list of baby names and the probabilities of a person having that name (this data is in fact published by the Social Security Administration<sup>3</sup>). We want to sample from that known distribution. The simplest way of

<sup>1</sup> For this example, you can generate a random number in the interval 0 to 1 (which is computationally cheap), multiply that number by 6, and then round up to the nearest integer.

<sup>2</sup> For a visualization of various sampling algorithms, see <http://alexeyradul.name/ideas/2013/introduction-to-automatic-differentiation/>.

<sup>3</sup> See <https://www.ssa.gov/oact/babynames/>.

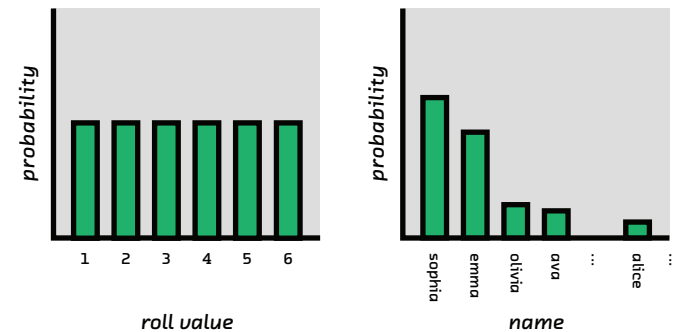


FIGURE 3.2 Sampling from uniform distributions is easy, but more complex distributions, such as baby names, can quickly become computationally expensive

generating observations that conform to this distribution is rejection sampling:

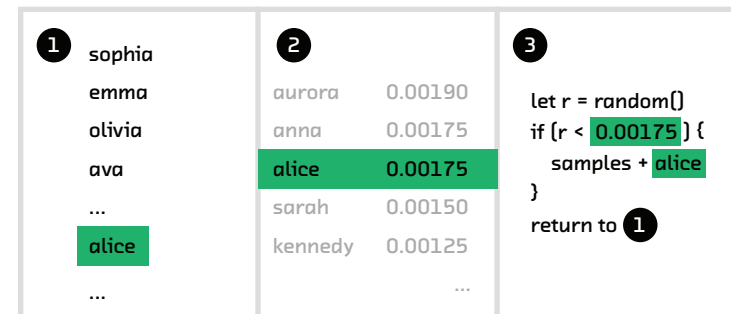


FIGURE 3.3 An example of rejection sampling

1. Select a trial name at random (i.e., temporarily assume all names are equally probable — this is like simulating rolls

of a die, so it's computationally cheap). Perhaps we get the name "Alice".

2. Look up its probability in the known probability distribution. "Alice" has a probability of 0.175% for the year 2015, according to the Social Security Administration.
3. Generate a new a random number between 0 and 1 (again, this is cheap). If that number is less than 0.00175, then "Alice" is accepted into our simulated set of observations. Otherwise, we reject it and go back to step 1.

Repeating this procedure many times will generate a sample of names that conforms to the probability distribution given by the Social Security Administration.

The problem here is that 0.175% is a small probability. Even the most popular names have probabilities of less than 1%. This means that the vast majority of our attempts to generate a single sample will end in failure. Our *acceptance rate* is low, so it takes a long time to generate a simulated set of baby names.

It gets worse. What if each trial took longer to conduct? For example, imagine we want to create a sample of planetary systems around stars.<sup>4</sup> We know what the distribution of stars looks like, but we need to perform an expensive simulation to derive a set of planets for each star. If each simulation took 1 minute to run and we needed 1,000 samples to work with, an acceptance rate of 1% would mean

---

<sup>4</sup> A real problem faced by astrophysicists such as Dan Foreman-Mackey, the developer of *emcee*, a Python package that implements Markov chain Monte Carlo sampling (<http://dan.iel.fm/emcee/>).

the calculation would take 10 weeks.<sup>5</sup> This makes it clear that we need sampling strategies that have better acceptance rates!

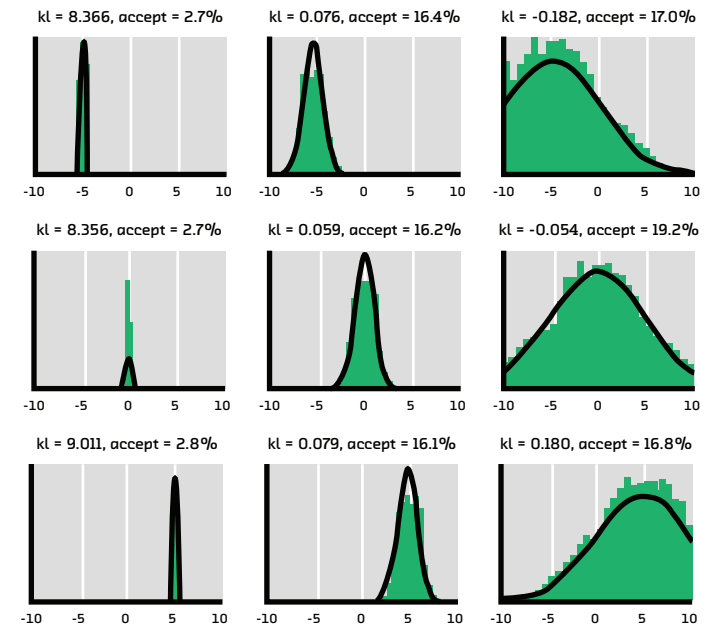


FIGURE 3.4 Rejection Sampling on a Gaussian Function where "accept" is how many samples were accepted and a small "kl" means a better representation of the distribution

---

<sup>5</sup> On the other hand, if we had a perfect acceptance rate it would take less than a day. It is important to note, though, that a 100% acceptance rate is also bad, since it means we are not properly exploring the space.

### What is a likelihood?

When dealing with Bayesian inference, the main components are the data, the priors, and the likelihood. In the case of a generative model where we combine distributions in a way that we think describes how our data is being created, the likelihood can be computed through an analysis of the underlying distributions. It tells us how reasonable it is for that generative model to take on any given set of parameters. Thanks to probabilistic programming and the advances we will talk about in [3.3.1 Automatic Differentiation](#), the creation of this likelihood is now automatic, allowing the developer to focus on architecting the model.

### 3.2 A Better Sampling Algorithm: Metropolis-Hastings

The Metropolis-Hastings algorithm, invented in 1953, was the first sampling algorithm to use Markov chains to improve the acceptance rate of samples. The algorithm rests on a simple idea: probability distributions are usually smooth. That means if we have found a sample with high probability, then there are likely to be other high-probability samples nearby.

This idea transforms the way we choose trial samples. In rejection sampling, we simply pick a new random sample from the domain of our distribution. In Metropolis-Hastings we instead take a small, random step away from our current sample. This is generally done by evaluating a Gaussian distribution centered on our current sample, which on average keeps us in around the same location we were in before but will, at lower and lower probability, move us further away in

order to explore the space (see below for a sample implementation of this method).<sup>6</sup>

```
#!/usr/bin/env python3
import numpy as np
import itertools as it

def sampler_mh(dist, x0=0, burnin=1000, alpha=0.5,
               verbose=False):
    x = x0
    samples_accept = 0
    for i in it.count(1):
        candidate = np.random.normal(loc=x, scale=alpha)
        candidate_prob = min([1.0, dist(candidate) /
                             dist(x)])
        accept = np.random.rand()
        if accept < candidate_prob:
            samples_accept += 1
            x = candidate
        if i > burnin:
            yield x, i, samples_accept
```

The other important way in which Metropolis-Hastings sampling differs from rejection sampling is how we decide whether or not to accept the trial sample. We make this decision by comparing the ratio of the probability of the trial sample to that of the previous sample. If this ratio is larger than a random number between 0 and 1, then we accept the trial. If it

---

<sup>6</sup> The extent that we explore is tuned by the alpha parameter. A small value means we try to stay in the same area we started in, while a larger value moves us around the space more rapidly.

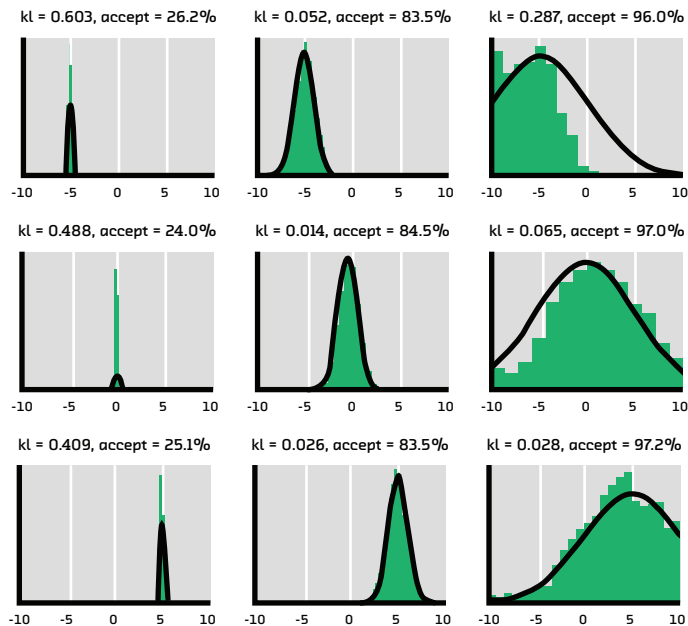


FIGURE 3.5 Metropolis Hastings on a Gaussian Function where "accept" is how many samples were accepted and a small "kl" means a better representation of the distribution

is smaller, we reject it and return to our previous sample.

This criterion means that wherever we start, we spend more time in areas of high probability and less time in areas of low probability. In fact, it's possible to show that we spend time in each location proportional to its probability, which means that the algorithm is doing exactly what we want: generating samples that are drawn from the probability distribution, and doing so with a high acceptance rate.

Metropolis-Hastings is a huge improvement over rejection sampling, but it still has problems. Firstly, because the

position of the next sample depends on the current one, there is a correlation between the samples. That is to say, the current sample doesn't depend only on the probability distribution but also on the previous state of the sampler. This can be solved with a process called *thinning*, where we only use every  $N$  accepted samples. However, this reduces the effective acceptance rate.

Secondly, we must discard the early samples the algorithm returns, until it reaches areas of reasonably high probability. This technique is called *burn-in*. As a rule of thumb, a good choice is to discard the first 50% of the samples. This necessary step further halves your acceptance rate!

Despite these problems, Metropolis-Hastings is still frequently used. Its simplicity and robustness make it usable in the most difficult situations, even when more modern approaches fail.

### 3.3 A Contemporary Sampling Algorithm: Hamiltonian Monte Carlo

The next big innovation in sampling came in 1987, with the advent of Hamiltonian Monte Carlo (HMC).<sup>7</sup> In this method, we take advantage of the derivatives of our likelihood function to move toward areas of high probability without depending on lucky steps, as in Metropolis-Hastings. In addition, the internal workings of HMC allow it to avoid samples being

<sup>7</sup> [http://www.stats.ox.ac.uk/~sip/work/report\\_hmc.pdf](http://www.stats.ox.ac.uk/~sip/work/report_hmc.pdf), <http://alexey.radul.name/ideas/2013/introduction-to-automatic-differentiation/>

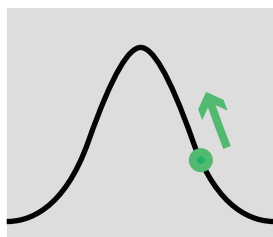


FIGURE 5.6 The Hamilton Monte Carlo method uses the derivative of the likelihood function to move towards high probability areas

correlated, as is the case with Markov chain-based methods. This means we don't need to thin the samples, so the acceptance rate remains high.

HMC also has a few drawbacks, though. First, it requires us to write down the derivative of our likelihood function. This can be almost impossible when the system being modeled

is very complex.<sup>8</sup> Second, the algorithm requires tuning parameters that can be extremely hard to set by hand.

HMC changed from a cool algorithm that was practical only in special cases to a generic and powerful approach thanks to two important innovations: automatic differentiation and NUTS.

### 3.3.1 Automatic Differentiation

The problem of derivatives was solved by *automatic differentiation*,<sup>9</sup> a technique that allows the computer to take exact

<sup>8</sup> And when the distributions are categorical (as with the baby names example), it is impossible to write down derivatives. For this reason, even advanced Bayesian inference algorithms still use Metropolis-Hastings to sample from categorical distributions.

<sup>9</sup> <http://alexey.radul.name/ideas/2013/introduction-to-automatic-differentiation/>

derivatives of arbitrary functions. This method works by introducing a new type of number (called a dual number) and running the function with it instead. Doing this yields not only the function evaluation, but also the derivative.

$$f(x) = y$$

$$f(x+z*e) = y + f'(z)e$$

FIGURE 5.7 By introducing a new abstract number,  $e$ , we are able to evaluate a function and its derivative at the same time

For example, if we have a function  $f(x)$ , we know we can evaluate it at any value simply by substituting in for  $x$ . However, if we introduce a new variable that has the property  $e^2 = 0$ , we can evaluate  $f(x + y*e)$  to get both the function evaluated at  $x$  and the first derivative evaluated at  $y$ . This method works for any function that you can normally evaluate with real numbers, and can be done very efficiently with computers.

Automatic differentiation is revolutionizing computational numerics in science, engineering, and machine learning. Researchers used to have to spend a lot of time finding tractable approximations to the derivatives of complicated functions. Automatic differentiation makes that hard (sometimes impossible) work unnecessary. It has made many new algorithmic methods practical for a much larger audience.



### 3.3.2 NUTS

No-U-Turn Sampling (NUTS)<sup>10</sup> is a twist on Hamiltonian Monte Carlo, first published in 2011. It features one nice property, and one absolutely essential improvement that is in some ways the final element that makes probabilistic programming a reality.

The nice property is that it is more efficient than plain HMC. It picks better trajectories that move the sampler toward high-probability regions faster, which increases the acceptance rate.

The essential improvement is its auto-tuning step. This means that users of the algorithm no longer need to spend time doing the difficult, error-prone work of tuning the parameters of HMC. A preliminary calculation sets those parameters automatically.

### 3.4 A Sampling Algorithm for the Future: ADVI

NUTS and automatic differentiation had a huge effect on what was possible. But researchers haven't stopped there. Before we look at how these developments are put together to create probabilistic programming, we'll touch on the current state of the art: *automatic differentiation variational inference* (ADVI).

Variational inference (VI) samples from a distribution by building a simple approximation of the distribution. That approximation is so simple that it can be sampled from directly, entirely circumventing the need for Metropolis-Hastings, HMC, or NUTS.

---

<sup>10</sup> See <https://arxiv.org/abs/1111.4246>.

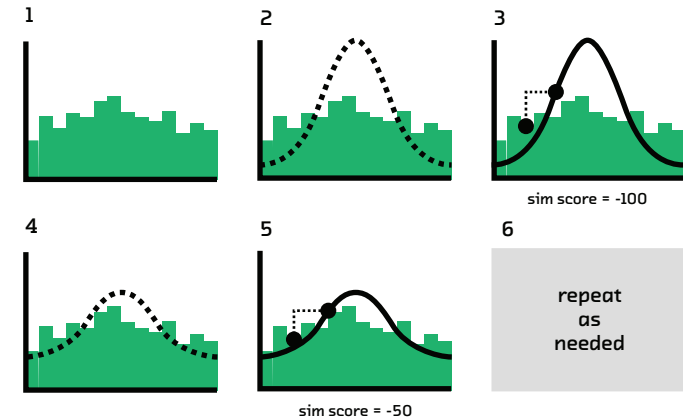


FIGURE 3.8 The steps of ADVI

To do this, we start with simple distributions that we understand well (e.g., Gaussian distributions) and perturb them until they match the real distribution from which we want to sample. The bulk of the work is done by *stochastic gradient descent* (SGD), which is the same algorithm used to train neural networks. Because of its widespread use, SGD is well understood and optimized, so VI is fast. Furthermore, VI is iterative and fundamentally realtime—data can be discarded, which allows it to be changing or streaming.<sup>11</sup> Sampling methods, on the other hand, require multiple passes over the entire dataset in order to converge to a final model, making it impossible to make a realtime version.

The conversion of a model from a sampling approach to VI used to be difficult and require complex math. For example, consider Latent Dirichlet Allocation (LDA), the popular topic

---

<sup>11</sup> See our report Probabilistic Methods for Realtime Streams [FF02].

modeling algorithm published in 2003.<sup>12</sup> The first implementation of this method used a sampling approach, which meant that fitting the model was slow and it could not be used for large or streaming datasets like those common in news and social media. It took a further seven years to get from the initial implementation to the now-ubiquitous VI implementation.<sup>13</sup>

ADVI, published in 2016,<sup>14</sup> is an algorithm that creates the VI approximations automatically. It uses automatic differentiation (see [3.3.1 Automatic Differentiation](#)) to learn about complicated functions, which it uses to create initial approximations. It then uses SGD to refine those approximations. This exciting development has the potential to make advancements like those seen in LDA, which took experts seven years to do by hand, as easy as flipping a switch!

### 3.5 Putting It All Together: Probabilistic Programming

Probabilistic programming is a programming paradigm that makes Bayesian data analysis much easier. Where once it was impractical or even impossible, it now requires concise code and relatively quick calculations. How?

NUTS and ADVI are the concrete algorithmic innovations that are ushering in the probabilistic programming era. Because the end user doesn't need to provide derivatives or tune parameters, they make the inference step easy. And because

they're built on Hamiltonian Monte Carlo or variational inference, they are fast. The two leading probabilistic programming environments, Stan and PyMC3, use NUTS and ADVI to make inference a one-liner.

But fast and robust algorithms are not all that's required to make Bayesian inference a practical proposition. Probabilistic programming languages also make life simpler by providing a concise syntax in which to define generative models using a library of built-in probability distributions. Probabilistic concepts such as generative models and priors are therefore primitive objects defined in the core language. That means that specifying the model is a *declarative* problem for the engineers. They declare what is known to be true. They do not need to explain the detailed steps needed to derive conclusions from those truths in an imperative fashion.

For example, we can declare that the probability that a website visitor will convert is based on a coin flip. Furthermore, we can say that the probability of the outcome of the coin flip isn't 50/50 but rather dependent on the layout of the website the visitor is shown, and the visitor's demographics. We can define and fit this model in Stan with the following simple code and the relevant data:<sup>15</sup>

---

<sup>12</sup> See our report Summarization (FF04).

<sup>13</sup> See <http://papers.nips.cc/paper/3902-online-learning-for-latentdirichlet-allocation>.

<sup>14</sup> See <http://arxiv.org/abs/1603.00788>.

---

<sup>15</sup> We'll see more examples of Stan and other languages in [4 Prototypes](#). Stan is one of many options. Our point here is to show that it is concise and declarative.

```

data {
  int<lower=0> N;
  int<lower=0> N_features;
  matrix[N, N_features] X;
  int<lower=0, upper=1> convert[N];
}
parameters {
  vector[N_features] p_coef;
}
model {
  convert ~ bernoulli(logit(X * p_coef));
}

```

Modern probabilistic programming languages allow the user to break out into a flexible general-purpose programming language when the built-in distributions won't do.<sup>16</sup> Along with the development of inference algorithms like NUTS and ADVI, this change turned the complex, restrictive Bayesian inference packages of the past into the simple, flexible probabilistic programming languages of today.

Engineers hoping to use Bayesian approaches used to need a statistics textbook to specify the model and an expert understanding of a huge range of computational tricks to implement, tune, and tweak the temperamental algorithms. These low-level concerns distracted from the analysis

problem itself—but probabilistic programming languages allow us to stop writing the statistician's equivalent of assembly code!

---

<sup>16</sup> The requirement to use built-in distributions was one of the things that made the very first attempts at probabilistic programming languages (e.g., BUGS) of limited use. Solving algorithms such as NUTS are agnostic to the distribution being used, which means that you can define your model however you like.

## CHAPTER 4

# Prototypes

### 4.1 When to Use Probabilistic Programming

Probabilistic programming may be the right approach if you answer yes to any of the following questions:

- Do you want to make use of institutional knowledge (suspicions, beliefs, logical certainties) about the quantities you want to measure and predict, rather than learn solely from the data?
- Do you have several different datasets that you want to learn from?
- Do you need to quantify the probability of all possibilities, not just determine which is most likely?
- Do you want to do online learning (i.e., continually update your model as data arrives)?
- Do you want to do active learning (i.e., gather more information until your predictions reach some threshold of confidence)?
- Do you want to use data to decide if a more complicated model is justified?
- Do you need to explain your decisions to customers or regulators?
- Do you want to use a single model to answer several questions?

- Do you have sparse data with a shared structure?

These correspond to the advantages we listed in [2.7 Uses and Advantages](#).

Guided by these questions, we built two prototypes that demonstrate some of the unique strengths of the probabilistic programming approach. We present the first, for modeling loans, in the next section. The second, for modeling real estate prices, is described in [4.3 Real Estate](#).

## 4.2 Loans

Consumer lending is a risky business for lenders, and it can be a frustrating and inscrutable experience for borrowers. Probabilistic programming can help with both these problems.

From the point of view of the lender, it is crucial to be able to quantify the probability of all possible outcomes. A model that can only tell you the most likely outcome is of little use in assessing risk. Two loans can both be likely to be repaid in full, but a loan that has a 99% chance of being repaid in full is very different from one that has a 51% chance.

Yet even if we can quantify this possibility of full repayment, we haven't quantified the possibility of all possible outcomes. That's because a borrower who pays back almost all of a loan is clearly preferable to one who pays back very little. If we can only say how likely nonpayment is, and treat all nonpayments alike, we fail to capture this.

For example, one loan might have a 99% chance of being repaid in full, but a 1% chance of having not a penny paid back. Another loan might have a 90% chance of being repaid in full, a 9% chance of almost all of the money being paid back, and

again a 1% chance of not a penny being paid back. We need a model that can tell the difference between these two loans. This means we need to assign probabilities to all possible outcomes, including (and especially!) the rare, catastrophic ones. And for that, we need probabilistic programming.

The approach has benefits for the borrower too. The complexity of a model that yields probabilistic estimates rather than simple yes/no answers does create communication and user experience challenges, as we'll see. But generally speaking, probabilistic models are interpretable (see [2.7.8 Regulated Environments and Interpretability](#)). This means that there is usually an answer to the question, "Why has this decision been made?" That answer can be given in a form that makes sense to a borrower (or user or customer); e.g., "The amount of this loan is high, so we're worried about the possibility of you repaying only a very small fraction." Using other approaches to machine learning (notably neural networks), such a question sometimes cannot be answered even in principle, much less in a form that can be shown to a user.

Transparency of the kind made possible by probabilistic programming increases user trust, but it may also become a legal requirement for models that make algorithmic decisions.<sup>1</sup>

And borrowers are not the only people to whom you might need to explain the behavior of a credit model. In many jurisdictions, lenders have a statutory duty to ensure and demonstrate that credit decisions are fair. Often this means that you must show that a model will not disproportionately impact

---

<sup>1</sup> See e.g. <http://fusion.net/story/321178/european-union-right-to-algorithmic-explanation/>.

protected groups of people. An uninterpretable model makes this difficult.

#### 4.2.1 Loan Data

Our goal is to provide a tool that helps loan officers decide whether to offer a loan. To do this, we would like to show them the relative probability of all possible outcomes. This is therefore a supervised machine learning problem. Supervised learning requires training data.

We got our training data from Lending Club. Lending Club is a startup peer-to-peer lending platform; an online marketplace where borrowers and lenders are matched. In principle, Lending Club itself does not make loans or borrow money.<sup>2</sup>

With the apparent goal of allowing lenders to better understand the characteristics of their borrowers, Lending Club publishes the semi-anonymized attributes of some 300,000 completed (i.e., paid in full or defaulted) loans dating back to 2007. The data includes borrower attributes that are likely to correlate with overall creditworthiness (income, debt, etc.), the details of the specific loan (amount, interest rate, term, etc.), and, crucially for our purposes, the outcome of the loan (paid in full or, if not paid in full, the amount repaid before default). This data is not without limitations and flaws and is not a good basis on which to build a generally applicable model of consumer debt (see [Health Warning](#) below), but for the purposes of demonstrating the power of probabilistic programming, it's exactly what we need to train our model.

---

<sup>2</sup> But see <http://www.bloomberg.com/news/features/2016-08-18/how-lending-club-s-biggest-fanboy-uncovered-shady-loans>.

Each record in the dataset corresponds to an issued loan. It records over 100 features of the borrower and the loan. We built a model that focuses on seven of these features that we found to be most predictive of repayment in our initial ad hoc exploration of the data:

- Annual income (in dollars)
- Ratio of monthly consumer debt payments (excluding mortgage) to monthly income (fraction)
- Credit card utilization (fraction of available limit)
- Interest rate of loan (percent)
- Amount of loan (in dollars)
- Term of loan (3 years or 5 years)
- Purpose of loan (chosen from 14 categories, including "debt consolidation," "medical," and "vacation")

Note that the borrower probably knows the values of these attributes (or could find them out) before making an application. Using only features for which this is true serves our goal of making the loan decision more transparent for the borrower. That is why we did not use the use Lending Club letter "grade," which is present in the training data but is a proprietary metric similar to FICO score.<sup>3</sup> In the interests of focusing our prototype on probabilistic programming, we also did not use the free text field in which the borrower is invited to write about the purpose of the loan.

---

<sup>3</sup> The FICO score is a metric used in the United States to quantify an individual's credit risk.

#### 4.2.2 Loan Model

Our loan model is actually comprised of two independent models.<sup>4</sup> One model, which we call `default`, predicts whether the loan will be repaid in full or defaulted on. This is a binary outcome. The other model, `fraction`, predicts the fraction of the loan that the borrower will repay in the event that they don't repay it in full.

The generative model for `default` is a linear Bernoulli model. A Bernoulli trial is an event with two possible outcomes, such as a coin toss. If the coin is fair, each outcome occurs with probability 0.5, so:

```
P(Heads) = 0.5
P(Tails) = 1 - P(Heads) = 0.5
```

More generally, for a Bernoulli trial with two outcomes:

```
P(Outcome A) = x
P(Outcome B) = 1 - P(Outcome A) = 1 - x
```

This would be fine for our `default` model if `x` were the same for every loan. But it is not: each loan has a different repayment probability that depends on the attributes of the borrower and the loan. We capture that in the model by allowing the probability of repayment to depend linearly on the

---

<sup>4</sup> We could have coupled these two models, either in the prior or the structure of the model. But we had enough data that this was not necessary, and to do so would have obscured the focus of this prototype, which is on quantifying risk in a clear way.

attributes:<sup>5</sup>

```
P(repayment in full) = sigmoid(f(loan))
```

Here, `f` is a linear combination of the attributes of the loan listed in [4.2.1 Loan Data](#). The `sigmoid` function is a transformation that forces its input to be in the range 0 to 1, which it must be in this case to make sense as a probability.

Our goal is to infer plausible ranges of values for the coefficients that each feature should be multiplied by to form the linear combination `f`. With those we can predict the probabilities of all the possible outcomes for a new loan.

The second generative model, `fraction`, predicts the fraction of the loan that the borrower will repay in the event that they don't repay it in full. This model must capture the logical certainty that the repayment fraction will be at least 0 but no more than 1. Here we use the flexibility of probabilistic programming to define the generative process to be one governed by a beta distribution. The beta distribution is a continuous distribution that, like the normal distribution, has a mean and variance. But crucially, it assigns zero probability to outcomes less than 0 or greater than 1, which is exactly what we want to predict the repayment fraction in the event of

---

<sup>5</sup> We are assuming here that repayment probability depends linearly on the attributes of the loan. If we suspected otherwise we could add nonlinear features to the model. For example, loan amount and borrower income are already features, but we could also include their ratio. In short, probabilistic programming requires feature engineering, which requires domain knowledge.

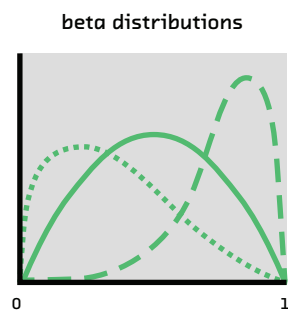


FIGURE 4.1 Example beta distributions

If a coefficient in a model is positive, then the bigger the corresponding feature, the more likely the good outcome (payment in full, or payment of a large fraction of the principal before default).

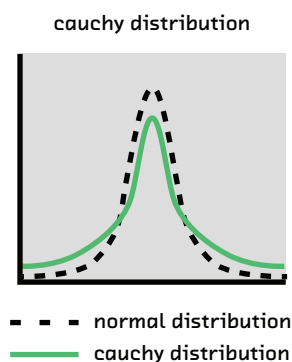


FIGURE 4.2 Comparison of the Cauchy and normal distributions

default (see FIGURE 4.1).

As with the Bernoulli model, we set things up so that the behavior of the beta distribution depends linearly on the attributes of the borrower and loan. And again, our goal is to infer credible values for the coefficients that govern this relationship.

If a coefficient in a model is positive, then the bigger the corresponding feature, the more likely the good outcome (payment in full, or payment of a large fraction of the principal before default).

For example, if the coefficient for annual income is positive, then we have evidence that people who earn more money are more likely to repay in full. Of course, we suspect this is true, and if we had very little data we could insert this knowledge in the prior distributions we place on these coefficients. But in this case we have the luxury of a very large amount of data, so we'll assume all

the coefficients are around 0 (i.e., none of the features matter), but we won't make that assumption very strong (i.e., we will admit the possibility that they are quite different from 0), and we'll allow the data to dominate rather than the priors. The standard way of doing this is with a Cauchy prior. The Cauchy distribution is a continuous distribution like the normal distribution (see FIGURE 4.2), but it turns out to have some convenient properties for priors on linear regression coefficients.<sup>6</sup>

### Probability distributions

Probability distributions are the building blocks of probabilistic programming. They are used to define the generative model, and to place priors on coefficients. Here are a few examples with common use cases.

Generative models:

- Bernoulli/binomial — Events with two outcomes
- Categorical/multinomial — Events with several outcomes
- Poisson — Discrete events (e.g., website visitors, hardware failures)
- Normal — Observations with noise (e.g., sensor data)

<sup>6</sup> See Gelman et al. [2008], <http://www.stat.columbia.edu/~gelman/research/published/priors11.pdf>. Note that in order for this strategy to work, continuous data must be *normalized* — i.e., transformed to have a mean of 0 and variance of 1.



Priors:

- Beta — Probabilities in Bernoulli trials
- Dirichlet — Probabilities in categorical trials
- Lognormal — Prior on unknown parameters that must be positive (e.g., physical size)
- Cauchy — Prior on parameters of linear regression

#### 4.2.3 Using Stan

We have all the ingredients for inference: the generative model, data, and prior. We chose to use the probabilistic programming language Stan to do the inference. Stan is an extremely popular language, particularly in academia. In Stan the user first specifies the generative model in a concise, declarative language. This is the complete listing for the `default` model described previously:

```
data {  
  int<lower=0> N;  
  int<lower=0> N_features;  
  matrix[N, N_features] X;  
  int<lower=0, upper=1> repaid[N];  
}  
parameters {  
  vector[N_features] p_coef;  
}  
model {  
  vector[N] p;  
  p_coef ~ cauchy(0, 2.5);  
  p = logit(X * p_coef);  
  repaid ~ bernoulli(p);  
}
```

The code contains three blocks. The `data` block describes the data that the compiled program should expect. In this case `N` is the number of loans in the training data and `N_features` is the number of features.<sup>7</sup> `X` is the feature matrix (which has `N` rows and `N_features` columns). Finally, `repaid` is an array of length `N` containing 1s or 0s that encodes whether a loan was repaid or not.

The `parameter` block describes the parameters of the model; i.e., the quantities whose posterior distributions we want to learn. In our case, that's a vector (or list) `p_coef` of `N_features` numbers. These are the coefficients in the linear relationship between the attributes of the loan and the repayment probability.

Finally, the `model` block defines priors on the parameters (a broad Cauchy distribution, as mentioned earlier) and the generative model that relates the features `X`, the unknown parameters `p_coef`, and the prediction target `repaid`. Stan defines a `logit` function to generate the probabilities and a highly efficient `bernoulli` distribution that simulates Bernoulli trials. It has a huge library of such functions.

The `fraction` model, which relates the attributes of borrower and loan to the partial repayment fraction, is a little more complicated when expressed as Stan code. This is because the beta distribution it uses has two arguments, while the Bernoulli distribution in the `default` repayment model has only one. It is qualitatively very similar, however, so we omit it here for the sake of brevity.

---

<sup>7</sup> `N_features` turns out to be 20, since of the 7 features listed previously, 2 are categorical.

From this point on, Stan handles the gory details of inference. It converts the declarative model code into very fast C++ code and compiles the C++ into an executable binary. When executed, this binary uses one of a suite of inference strategies to efficiently sample from the posterior distribution of the parameters. The most mature of these strategies is NUTS (see [3.3.2 NUTS](#)), which is what we used, but there are other options (see e.g. [3.4 A Sampling Algorithm for the Future: ADVI](#)). Stan code can be compiled and run by a dedicated command-line driver or by PyStan and RStan, which aim to make it easier to work with from Python and R.

We wrote simple Python code to prepare the data (which should be in the form of a Python dictionary with keywords that correspond to the entries in the `data` block of the Stan model). We then used PyStan to compile and run the inference step. When inference is complete, PyStan passes an object back to the Python environment that contains samples from the posterior distribution.

We examined these posterior samples to ensure inference converged successfully (as well as performing the usual machine learning checks on accuracy, precision, and recall). Once satisfied, we pickled the posterior samples to disk to allow us to deploy the model to make predictions on previously unseen loans.

#### 4.2.4 Loan Model Deployment

Having trained the model and stored the posterior distribution implied by the data and priors, we were in a position to simulate the outcome of new loans. Stan is focused on inference and does not provide a way to predict the outcome of a

single loan outside that context. This means we had to reimplement the generative model in another language. Note that the reimplementation does *not* need to do inference, which is by far the hardest step. Given a set of features and a draw from the posterior distribution of the parameters of the model, it simply needs to simulate the random process defined by the generative model. For our `default` repayment model, the Python code for the generative model looks something like this:

```
import numpy as np

def sigmoid(u):
    return 1 / (1 + np.exp(-u))

p = sigmoid(np.dot(X.T, p_coef))
repaid = (np.random.binomial(1, p) == 1)
```

Here, `X` is an array of the features of a particular loan and `p_coef` is a draw from the posterior distribution of linear coefficients that we used Stan to find.

To simulate the outcome of a loan (i.e., a particular value of  $X$ ), this code is evaluated for many values of `p_coef` (i.e., many draws from the pickled posterior distribution). The posterior distribution encodes our confidence about what values of `p_coef` are plausible in the light of our data and prior beliefs. By using many draws from this posterior, and simulating the Bernoulli trial many times, we can propagate this confidence into the posterior distribution for repayment. That is, we can say not just "This loan will be repaid" but, e.g., "This loan has an 81% probability of being repaid."

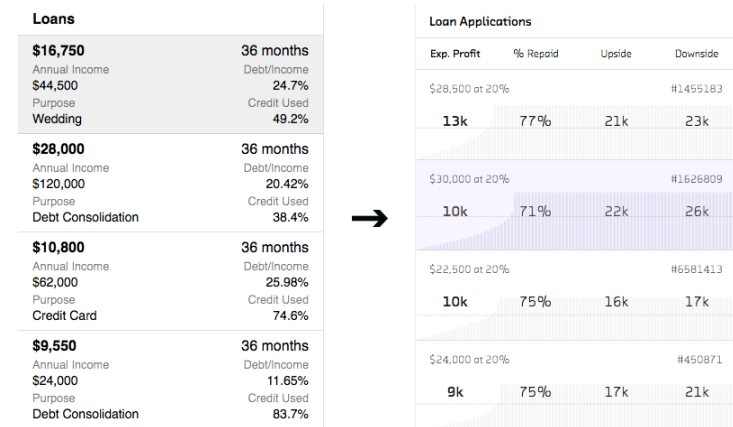
For the prototype, we wrapped this code and a correspond-

ing function for the `fraction` model in a web API using Flask.

#### 4.2.5 Product: Loan Officer Simulator

Armed with a model that predicted loan repayment probabilities, we set about creating an interface and conceptual frame for those predictions that would best demonstrate the strengths of probabilistic programming. In order to help users really grasp the usefulness of these predictions, we decided to add a simulation layer to the prototype, casting the user in the role of a loan officer tasked with evaluating and negotiating the interest rates for a group of loans. By placing users in a game where their virtual money is at stake, we let them experience for themselves just how useful probabilistic predictions can be.

We started our prototype design process by laying out several loans in a left sidebar. Clicking on one would display a more detailed account of the loan and, eventually, allow the user to take action. At first we experimented with displaying information about the applicant's finances, since these factors, along with loan characteristics such as the amount and interest rate, are what feed into the predictive model. As we explored that iteration in the context of a product whose purpose is to help the user make better loans, however, we realized that displaying that information so prominently is actually a distraction. Displaying the applicant info encourages the user to make their own guesses about how those facts impact their chance of repayment, when really we want to guide them to the model's predictions. The model is driven by the quarter million loan outcomes we trained the model on, so it has much more data to help it infer outcomes from applicant



The diagram illustrates a transition from a list of loans to a detailed view of loan outcomes. On the left, a table titled 'Loans' lists four loan entries. An arrow points from this table to a table titled 'Loan Applications' on the right, which shows the expected profit, percentage repaid, upside, and downside for each loan application.

Loans	
<b>\$16,750</b> Annual Income \$44,500 Purpose Wedding	<b>36 months</b> Debt/Income 24.7% Credit Used 49.2%
<b>\$28,000</b> Annual Income \$120,000 Purpose Debt Consolidation	<b>36 months</b> Debt/Income 20.42% Credit Used 38.4%
<b>\$10,800</b> Annual Income \$62,000 Purpose Credit Card	<b>36 months</b> Debt/Income 25.98% Credit Used 74.6%
<b>\$9,550</b> Annual Income \$24,000 Purpose Debt Consolidation	<b>36 months</b> Debt/Income 11.65% Credit Used 83.7%

Loan Applications				
Exp. Profit	% Repaid	Upside	Downside	
\$28,500 at 20%				
13k	77%	21k	23k	#1455183
\$30,000 at 20%				
10k	71%	22k	26k	#1626809
\$22,500 at 20%				
10k	75%	16k	17k	#6581413
\$24,000 at 20%				
9k	75%	17k	21k	#450871

**FIGURE 4.3** An earlier and later iteration of the list of loans sidebar. We learned that displaying information about the outcomes was more useful than information about the applicant

information than the user does.

Having decided to display information about the predicted outcomes, we then determined which metrics and graphs best communicated the shape of those outcomes. Expected repayment amount, which is the long run average repayment calculated over many (in this case, 4,000) loan repayment simulations, is the primary metric we chose. If you had to reduce the range of generated probabilities to one number, this is the number you would want to use. To make it easier to compare across different loan amounts we adjusted it by subtracting the original loan amount. That calculation gave us the average profit.<sup>8</sup>

<sup>8</sup> To make the comparisons between loans and different interest rates easy to comprehend, we set each loan to a term of 3 years.

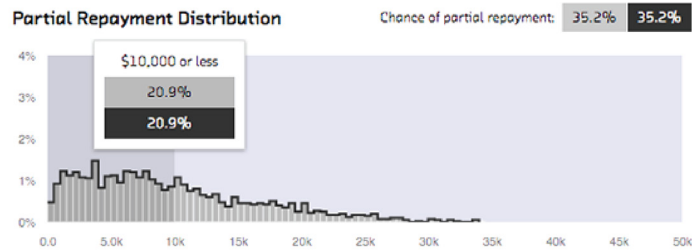


FIGURE 4.4 The distribution graph provides a detailed look at the probabilities of partial repayment outcomes

Expected profit is the best reduction of the probabilities to one number, but that flattening still results in the loss of a lot of information. We use three other metrics to provide further context. Percentage repaid is the chance that the loan will be fully repaid, a big help when assessing the risk of a particular loan. The upside is the potential profit if the loan is fully repaid, this shows how much you stand to make if everything goes right. Downside shows the amount you stand to lose in a catastrophic outcome. You could lose this much or more in 5% of the outcomes. Being able to quantify the risk of a catastrophic outcome is a big benefit of using a probabilistic approach.

Besides metrics, we also had the opportunity to use graphs to visualize the generated probabilities. We started with the emblematic graph of probabilistic programming,<sup>9</sup> the probability distribution graph. While the graph was useful for demonstrating partial repayment outcomes, those probabilities were generally swamped by the more likely full-re-

<sup>9</sup> It is our cover for this report, after all

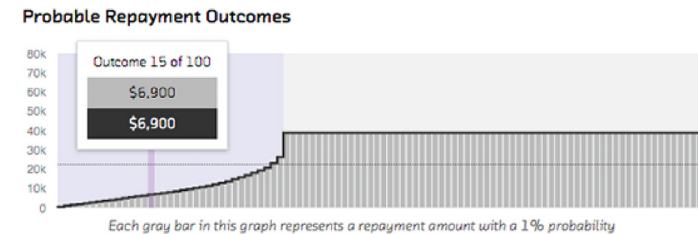


FIGURE 4.5 The probable repayment outcomes graph puts the partial repayment probabilities in the context of the chances of full repayment

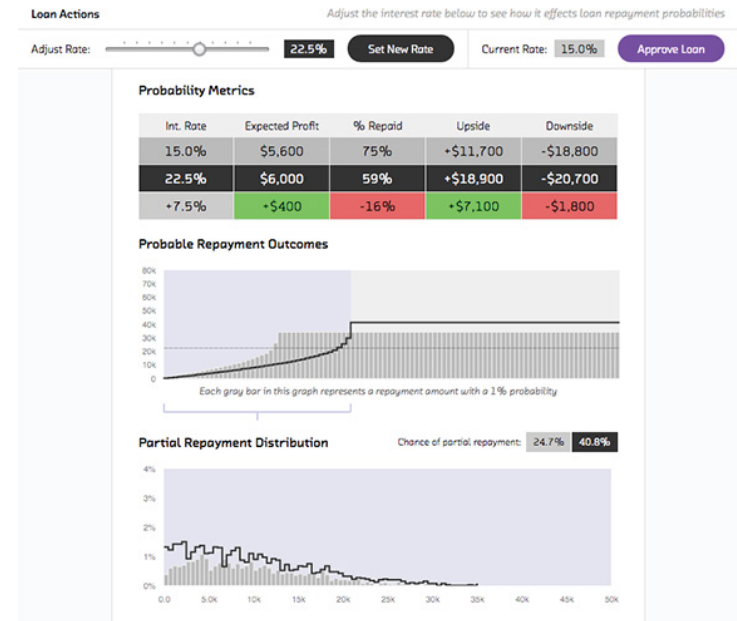
payment outcome. We therefore made the decision to focus the distribution graph on partial-repayment outcomes only. These outcomes are still an important part of loan evaluation, and a key advantage of the probabilistic approach. When competing against other organizations offering loans any predictive advantage, even for low-probability events, is a big deal.

Knowing the partial repayment distribution graph would make up an important but limitted part of the detailed loan view, we tried to come up with something else that could put those partial repayments in the context of the overall likely outcome. After some experimentation we came up with a somewhat unconventional graph visualization of possible repayment outcomes. Titled "Probable Repayment Outcomes", it placed the repayment outcomes of 4,000 simulations of the loan (grouped into 100 bars, each representing a 1% probability) side-by-side. These outcomes were then sorted from least to greatest amount, creating a visualization of the *shape* of the loan. The percent likelihood the loan would be fully repaid is

embedded in the graph as the long shelf of full repayment outcomes on the right. On the left side of the graph you can see the partial repayment probabilities, their relative steepness gives you an idea of how quickly the payment amounts fall off from the full repayment level. This is the partial repayment information that is shown in greater detail in the probability distribution graph. The dotted horizontal line marks the original loan amount.

The big benefit of this graph is how it shows context, as you can see the relative likelihood of the partial repayments in the context of the more likely full repayment outcome. Since the graph scale is kept constant across loans, it also gives you an idea of the relative stakes at play in different loans. There are trade-offs, because the graph is unconventional the graph may not be immediately recognizable and more prone to misinterpretation. To try and minimize those risks, we tried to add guideposts in the axis labels and hover tips as much as possible.

With our metrics and graphs in place, we started working on the interactive layer of the prototype. The sidebar allowed you to compare different loans, but we knew we also wanted the user to be able to see how changing one variable effected the probable outcomes. Technically, the model would allow us to change any of the factors that fed into the model, from applicant income level to the interest rate of the loan. We quickly found that limiting the variables that could be adjusted led to a better understanding of how the model reacted to changes. We settled on interest rate as the manipulable input because it made the most sense conceptually. You can imagine a lending organization negotiating with an applicant to



**FIGURE 4.6** When you adjust the interest rate the metrics and graphs show you the effect of your changes on the probable outcomes

set the most profitable rate.

To communicate the effects of an interest rate change, we displayed the adjusted probabilities next to the current ones across the metrics and graphs.<sup>10</sup> We used color coding and experimented with several arrangements to try and clearly show the effect of different interest rate choices.

<sup>10</sup> This was technically made much easier by our use of React and Redux, which encourage you to keep all of the application state in one place.

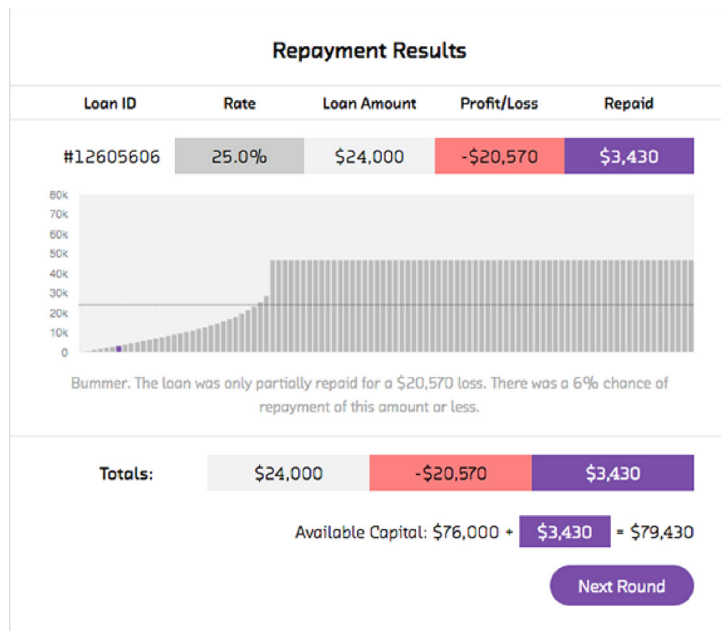


FIGURE 7 The simulation element of the prototype encourages the user to engage with the model's predictions

Once a user has decided on an interest rate, they are then able to approve the loan, which takes the loan amount from their starting capital. After they have approved several loans they are able to make the loans, at which point we simulate the loan outcomes by randomly selecting from one of those displayed in the Probable Repayment Outcomes graph. Their relative profit or loss from that round is then carried over into the next round, where the user can repeat the process and see how much they can make. This simulation layer is relatively light, but by causing the user to feel like they have something at stake in their decisions, we found that it caused them to

engage with predictions in a more active way than if we had only showed them visualizations without the extra layer of interactivity.

#### 4.2.5.1 Loan Officer Simulator Front-End Tech

The web interface of Loan Officer Simulator was built using the React.js library.<sup>11</sup> Along with React, Redux.js<sup>12</sup> was immensely helpful in keeping our data in-sync across the sidebar and detail view, and generally keeping app state manageable as we switched and rewrote components through different prototype iterations. The graphs are rendered using D3.js,<sup>13</sup> getting React and D3 to work together nicely is an interesting challenge. We were grateful to people, such as Freddy Rangel<sup>14</sup> who had written up their experiences in integrating the two. The CSS library Basscss<sup>15</sup> iterate quickly on the design.

#### Health Warning

Loan Officer Simulator is a demo of probabilistic programming, not investment advice! Lending Club borrowers are unrepresentative of many other kinds of borrowers and the company has been implicated in some shady data practices, placing our training data in some doubt.

Even if the data was perfect, the model still wouldn't

<sup>11</sup> <https://facebook.github.io/react/>

<sup>12</sup> <http://redux.js.org/>

<sup>13</sup> <https://d3js.org/>

<sup>14</sup> <https://github.com/freddyrangel/playing-with-react-and-d3>

<sup>15</sup> <http://basscss.com/>

reflect an important aspect of consumer lending: the borrower has the right to decline a loan if the interest rate is set too high. We'd need data about potential borrowers who declined loan offers to model this process properly.

### 4.3 Real Estate

The real estate prototype for this report was focused on simplicity, while still being able to capture some of the nuances of the data. In general, real estate in New York City is a stable system. Demand for apartments is quite high, and development is happening at a staggering rate. As a result, the number of available residences in a particular area can change quite quickly and tightly correlates to the rising of prices.

This, however, is offset by the amount of speculation that is happening in areas that are further out on the subway lines. Developers frequently buy houses in these underdeveloped areas with the intention of building them up in 5-10 years, when prices start rising in the neighborhood. Furthermore, due to the increasing incidence of devastating hurricanes, areas highly impacted by flooding that haven't previously seen much of this speculation are being bypassed.

All in all, this creates interesting trends where sale prices of apartments and buildings in a given area are tightly correlated to the previous trends in development of that area. Places that have seen a low but steady amount of development will soon see small increases in costs, which will attract more developers, which will only serve to ramp up the price increases. Conversely, whereas areas where development has stopped as a result of natural disaster also see a steady climb

in development (albeit mostly reconstruction), the amount of feedback from this construction is small and it doesn't create a feedback cycle.

What we would like is a process that can generate likely block-level median sale prices for areas that the model has never seen before. For example, we'd like to generate possible median prices for every block in New York City for 2017, as a prediction of what will happen in the future. In this case, for each block we will generate hundreds (if not thousands) of these possible data points and then show statistics about the results to give a sense of the range of possible outcomes. As described in [4.3.3 Product: Probabilistic Real Estate](#), we can just show the user the average result for each block, but an important result of being able to do Bayesian inference is that we are able to see the full distribution of results.

The ability to generate probable data points means the model can answer many questions. For example, "How many blocks will generate houses with a median sale price less than \$400,000 in 2017?" Or, more interestingly, "How many blocks will generate mean house prices less than the set price at least 75% of the time in 2017?" This latter version of the question factors in the uncertainties in the model—generated house prices will vary with our confidence in the result, so using the confidence in the questions we ask fully utilises all of the predictive power.

From a modeling perspective, this is a perfect problem for a probabilistic approach. First, the data is hierarchical, which means we can bootstrap understanding for an area without much data with data from areas around it (we discuss this more in [4.3.2 Real Estate Model](#)). Furthermore, a probabilis-



tic approach gives us an idea of the confidence of our various predictions and thus the volatility of an area. Since there are many features and feedback mechanisms that can be used to account for price fluctuations, having a sense of when our model is applicable and when it isn't is an important feature.

#### 4.3.1 Real Estate Data

New York City has quite a lot of data surrounding housing in the five boroughs. PLUTO,<sup>16</sup> for example, is a dataset that contains tax and unit information for every lot in the city, including the geometry of the lot. In addition, there is a special dataset that includes every real estate sale in the city,<sup>17</sup> including metadata regarding the zoning state of the lot at the time of sale.

These two datasets can be joined together to give a full picture of what is happening with house prices in New York City. We have data on both the number of available units and which ones were sold and for how much.

To make this useful, we create hexagonal bins that encompass about four square blocks and take the median house sale price per year of each bin. In addition, we note the number of available residential units in the bin and what neighborhood and borough the bin is in.

---

<sup>16</sup> See <http://www1.nyc.gov/site/planning/data-maps/open-data.page#pluto>.

<sup>17</sup> See <https://www1.nyc.gov/site/finance/taxes/property-annualized-sales-update.page>.

#### 4.3.2 Real Estate Model

In order to capture these dynamics in the model as simply as possible, we chose to use as features only the year, the block (plus neighborhood and borough), and the number of available residential units on that block. The intuition is that this should be enough to capture both the availability of housing in an area and the block/neighborhood/borough dynamics of demand. The feature vector for this ends up being:

$$X = [1, \text{year}, \text{year}^2, \text{units}, \text{units} \cdot \text{year}, \text{units} \cdot \text{year}^2]$$

Just as we did in the loan prototype (see [4.2 Loans](#)), we create a linear function based on these features and learn coefficients for our features. However, since we have hierarchical data (every data point is at the block level but also has neighborhood and borough information), there are many more parameters than data points. That is to say, each hexagonal bin has its six coefficients as well as the six coefficients for the neighborhood it's in, six more for the borough it's in, and finally six more for a city-wide value.

These four sets of coefficients are put together into a larger linear function to predict what the median house price for a block is. Interestingly, since we have this natural hierarchy in the model we can ask questions not only at the block level but also at the neighborhood, borough, or city level.

A major benefit of this hierarchy is how it deals with the problem of sparse data. In any given year, many blocks have zero sales and no information can be inferred about them. However, we can use spatial proximity to learn about a block that had no sales by looking at neighboring blocks that may



have had sales. Even if an entire neighborhood had no sales in a year, there would be at least one sale in the borough. This is similar to what happens in the real world when evaluating the price of a house — it is compared to similar properties that are close to the house in question.

An interesting result of this hierarchy is that we can see what the dominant factors for a house's price are. For example, we may see for a particular block that the coefficient for the *neighborhood* is particularly high but that of the *block* is fairly low. This tells us that the neighborhood as a whole is popular and has high prices, but there is nothing particularly special about that block. Furthermore, we may identify a block that has a much lower value for the "available units" coefficient than the borough it's located in, indicating that the block has already been saturated with too many new apartments.

Another difference in how the coefficients are treated in this model versus the Lending Club model is that we assume that our coefficients are correlated with each other. It's easy to see this in the case of construction, where we have a year term and a squared year term. However, in addition the number of available units in a block is correlated with year.

Unfortunately, highly correlated (log)normal distributions are notoriously hard to sample from. A solution to that problem is using the Cholesky decomposition to first uncorrelate the data and then sample from the uncorrelated log-normal distribution. This optimization isn't necessary, but it makes the sampling run a lot faster than it would otherwise. It also is a useful comparison that relates probabilistic programming to normal programming — while it is easy to get something working, sometimes it takes extra care to have it work quickly.

The median that is calculated from the linear function is then fed into a Student's t-distribution in order to account for the long-tail nature of the data and for noise in the predictions. The benefit of this is that the model can change the "degrees of freedom" parameter of the t-distribution in order to learn just how long-tailed the distribution of house sale prices actually is.

This model was heavily influenced by the model used in Reaktor's *Kannattaa Kokauppa* project on Finnish housing prices.<sup>18</sup> While many of their assumptions and priors about housing markets had to be changed to deal with the nuances of housing in New York City, their work served as a good basis for our work and also shows how model structure can be used for multiple types of problems — as long as the fundamental generative process is similar, the model can be reused. We therefore don't include the Stan code in this report. To see an example of the process of turning a model description into a deployable probabilistic program, see [4.2 Loans](#).

#### 4.3.3 Product: Probabilistic Real Estate

Just as the model for this prototype was heavily influenced by the model by Reaktor, our initial visualization of the NYC data was based on their visualization of Finnish housing prices.<sup>19</sup> For our first iteration, we focused on displaying the median prices across New York City at the lot level. The probabilistic model made this level of

---

<sup>18</sup> See <http://ropengov.github.io/r/2015/06/11/apartment-prices/> and <http://kannattaakokauppa.fi/#/en/>.

<sup>19</sup> <http://kannattaakokauppa.fi/#/en/>

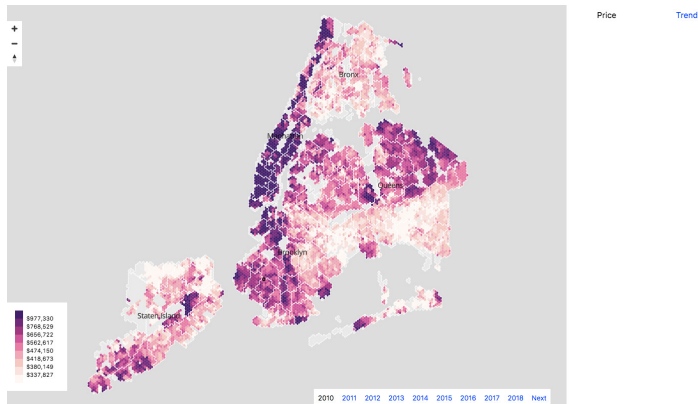


FIGURE 4.8 The earliest prototype showed median prices across New York City

specificity technically possible, but the visualization proved too computationally intensive to render on the web.<sup>20</sup> We then settled on grouping those lots into hexes. Hexes both made rendering easier and resulted in a more continuous visualization, eliminating the distracting gaps between lots.

Once we got a hex-based median price map prototyped, we started thinking hard about how to make something that more clearly demonstrated the strengths of probabilistic programming. There comes a time in every prototyping process where we take a step back from early iterations and take a fresh look at the possibilities of the data we have. In this case that meant focusing on the samples generated by the prob-

<sup>20</sup> Finding the right javascript library for rendering the map was a big part of the early front-end prototyping. See [4.3.2.1 Probabilistic Real Estate Front-End Tech](#).

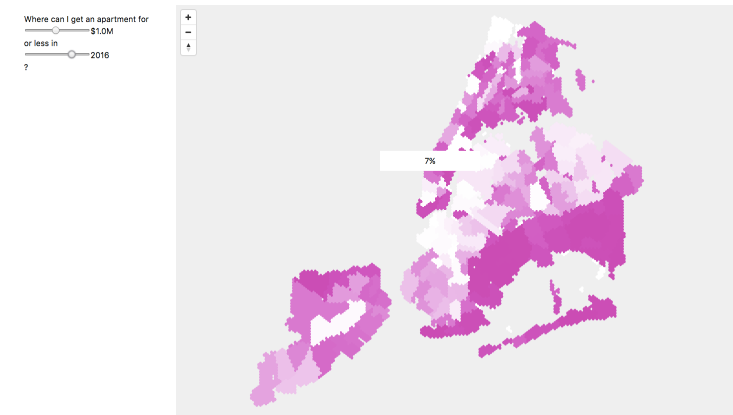


FIGURE 4.9 We shifted to a question interface to better show the strengths of probabilistic programming

abilistic model. The price map we had prototyped was using those samples to calculate median prices, reducing the distribution to just one number. We began focusing on how we could make better use of those samples.

We found the answer, and the form of the next iteration, by focusing on the types of questions that are best suited to be answered by probabilities. Instead of displaying median prices we would allow the user to choose a price level. The map would then be a display of the percentage of generated samples that were less than the user-specified price – in other words, the percentage of properties in that area that you could afford at the chosen price. We also switched to a neighborhood, rather than hex, map view. This was once again motivated by computational constraints (making dynamic calculations across the samples was more expensive than always displaying the median). The neighborhood view also made

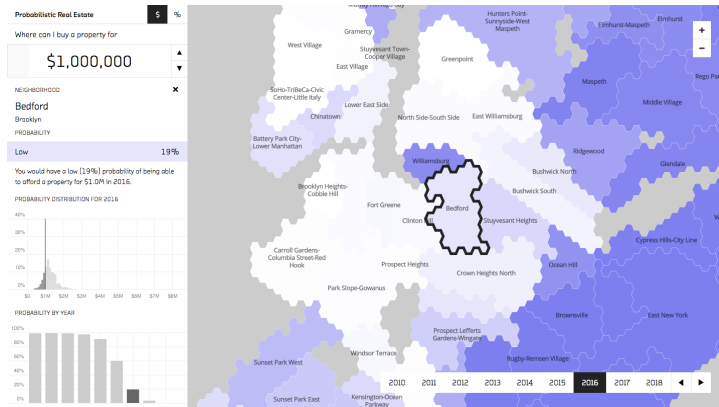


FIGURE 4.10 The neighborhood detail view features a probability distribution graph highlighting how the probability is calculated

the map display more readily understandable for users, since neighborhoods are a more familiar city unit than hex tiles. (Though we would later bring the hex display back into the map for the neighborhood detail view.)

As we continued to iterate on the probabilistic price view, we started to develop the graphs for the neighborhood detail view. The first and key graph was the probability distribution, which shows the number of samples for different price points. The graph is a pleasingly intuitive illustration of how the probability is calculated using the price. The vertical line represents the selected price, and all of the samples which fall to the left of the line represent the samples below the selected price. That the graph and the central motivating question of the application lined up so well was a sign we were on the right conceptual track.

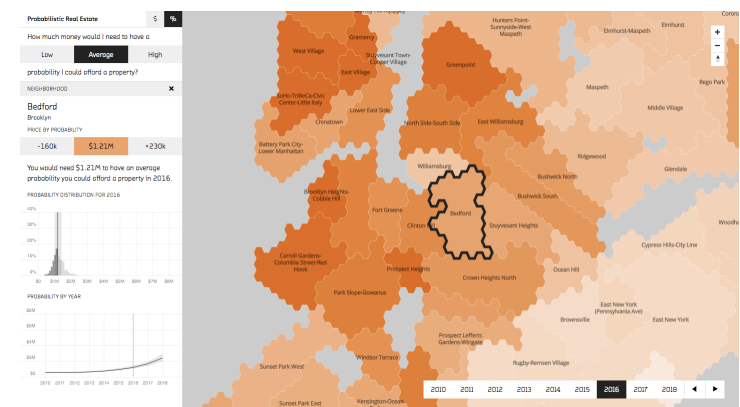


FIGURE 4.11 The probability mode of the app lets a user select a certainty level and displays a price for each neighborhood that corresponds to that certainty

Looking at the probability distribution graph in more detail suggested another useful feature for the application. Price mode allowed a user to select a price and see the corresponding probability, we next built a mode that reversed that relationship and allowed the user to select a probability level and see the corresponding price. Initially we allowed users to select probability as a percent value, but that felt too granular for this mode, so we simplified it into low (25%), average (50%) and high (75%) probabilities. Set to average probability, the map is functionally the same as the original median price map we made, the difference being that now it is framed in a way that emphasizes the strengths of a probabilistic approach.

In the final product, then, we had built from a median price map into an app which answered targeted user questions making full use of the sample generation at which prob-

abilistic programming excels. Rather than having to reduce everything to a one-number average, we encouraged the user to explore the data from a variety of angles. The ability to qualify the probability of all possibilities allowed us to create a more relevant and robust visualization of real estate prices in NYC.

#### 4.3.2.1 Probabilistic Real Estate Front-End Tech

Like the [4.2.5.1 Loan Officer Simulator Front-End Tech](#), the interface for Probabilistic Real Estate interface was coded using React.js, Redux.js, D3.js, and Basscss.<sup>21</sup> The map itself was rendered using MapboxGL.js<sup>22</sup> which, because of its use of WebGL, we found to be more performant for our needs than other map-rendering options such as Leaflet.js and D3.js.

### 4.4 PyMC3

As we discussed in [4.2.4 Loan Model Deployment](#), deployment of Stan code can be difficult. Stan is not a general-purpose language, which limits how we can interact with it from, e.g., a web app.

One way around this would have been to use another probabilistic programming language that interfaces directly with our language of choice. PyMC3 (see [5.1.2 PyMC3](#)) is another such language that works on top of Python. This means that the resulting model is much easier to use for things other than inference and offline analysis of the results which makes

---

<sup>21</sup> <https://facebook.github.io/react/>, <http://redux.js.org/>, <https://d3js.org/>, <http://basscss.com/>

<sup>22</sup> <https://www.mapbox.com/mapbox-gl-js/api/>

exposing the model to an API much simpler. For example, we could rewrite the default loan model in PyMC3 as follows:

```
import pymc3 as pm
from pymc3.distributions.continuous import Cauchy
from pymc3.distributions.discrete import Bernoulli
from scipy.special import logit

X, repaid_actual = process_dataset()

model = pm.Model()
with model:
    # Create Model
    p_coef = Cauchy(0, 2.5)
    repaid = Bernoulli(logit(X * p_coef),
                      observed=logit(repaid_actual))

    # Do Inference
    start = pm.find_MAP()
    step = pm.NUTS()
    trace = pm.sample(num_samples, step, start,
                     progressbar=True)
```

We now have a model object that we can use to answer later queries. In addition, PyMC3 supports all of the modern solving strategies that Stan does (most importantly NUTS and ADVI; see [3 Probabilistic Programming](#)).

During our early benchmarks, PyMC3 was still in very active development, and we found that it ran considerably slower than Stan. The PyMC3 development group now benchmarks prerelease versions of PyMC3 on par with Stan. We discuss PyMC3 more in detail in [5.1.2 PyMC3](#).

## CHAPTER 5

# Tools and Products

In this chapter, we review the landscape of probabilistic programming, including the available tools and the people making them, and the people applying those tools in real applications.

### 5.1 Probabilistic Programming Tools

#### 5.1.1 Stan

Stan<sup>1</sup> is the most mature, widely adopted, and well established probabilistic programming language. It is supported reliably<sup>2</sup> and is the default choice for academics in the sciences interested in these languages. Stan is an academic project and thus tends to be the first language to support cutting-edge innovations in the field. We explained Stan's usage in detail in [4.2.3 Using Stan](#). It is a concise, powerful language for model specification and inference.

Stan is a statically typed and compiled language. Once compiled, you can sample from a model repeatedly with different data. Any changes to the model require recompilation.

You can interact with the model using a command-line

---

<sup>1</sup> <http://mc-stan.org/>

<sup>2</sup> <http://mc-stan.org/support/>

driver or interfaces to R or Python. Due to the interests of Stan's development team, the command-line version tends to get updates first, followed by the R interface, followed by the Python interface. This doesn't matter for general probabilistic programming use, but it can be a problem if you are trying to use Stan's latest features. For example, while we were building the prototypes that accompany this report ADVI support had not yet been merged into the main Python branch.

Because Stan is not built on top of a general-purpose programming language, it has additional problems that severely complicate deployment of products based on its models. These are discussed in detail in [4.2.4 Loan Model Deployment](#).

### 5.1.2 PyMC3

PyMC3<sup>3</sup> shares many of Stan's features and performance characteristics. Crucially, however, it is a Python package, which means you interact with it directly from within a general-purpose programming language.

At the time of writing, the stable version of PyMC is version 2.<sup>4</sup> Version 3 should be released before the end of 2016; notable improvements over version 2 include a redesigned Python interface that takes advantage of context managers and support for the ADVI algorithm. PyMC3 also implements traditional Metropolis-Hastings, which allows it to be used

---

<sup>3</sup> <https://pymc-devs.github.io/pymc3/>

<sup>4</sup> For a useful shoot-off between PyMC version 2, Stan, and emcee, see "Frequentism and Bayesianism IV: How to Be a Bayesian in Python" (<http://jakevdp.github.io/blog/2014/06/14/frequentism-and-bayesianism-4-bayesian-in-python/>).

for models with discrete parameters. PyMC3 has a lively community around it and is under very active development. This can be fantastic for getting help or requirements of learning about new features, but it also means that things can change quite quickly.

Because PyMC3 is a Python module (rather than a language like Stan), it is an incredibly flexible and versatile tool. We touch on one of the benefits of this in [4.4 PyMC3](#): it simplifies deployment enormously. Another crucial benefit is that distributions that don't exist in the framework can be created easily. For example, we can create a vectorized Latent Dirichlet Allocation model that can be solved quickly and with either Metropolis-Hastings or ADVI as follows:

```
import numpy as np
import pymc3 as pm
from pymc3.distributions.dist_math import bound, factln
from theano import tensor as tt

ALPHA = 1.
BETA = 1.

class ManyMultinomials(pm.Discrete):
    def __init__(self, n, p, *args, **kwargs):
        super(ManyMultinomials, self).__init__(*args,
                                                **kwargs)
        self.n = n
        self.p = p

    def logp(self, x):
        n = self.n
        p = self.p
```

```

    return bound(
        factln(n).sum() + tt.sum(x * tt.log(p) -
            factln(x)),
        tt.all(x >= 0),
        tt.all(x <= n[:, np.newaxis]),
        tt.all(tt.eq(tt.sum(x, axis=1), n)),
        n >= 0
    )

with pm.Model() as model:
    topic_word = pm.Dirichlet('topic_word', ALPHA *
        np.ones(N_WORDS), shape=(N_TOPICS, N_WORDS))
    doc_topic = pm.Dirichlet('doc_topic', BETA *
        np.ones(N_TOPICS), shape=(N_DOCUMENTS, N_TOPICS))
    doc_word = pm.Deterministic('doc_word', doc_topic.
        dot(topic_word))
    words = ManyMultinomials('words', word_counts,
        doc_word, observed=data)

with model:
    advi_fit = pm.advi()

```

PyMC3 uses Theano to implement automatic differentiation for NUTS and ADVI. This means that to build new distributions you sometimes need to interact directly with Theano objects, as in the LDA implementation just shown. Nevertheless, this example demonstrates the true power of probabilistic programming. In a simple and declarative way we are able to build an extremely rich, hierarchical model.

### 5.1.3 Edward

Edward<sup>5</sup> is a new library designed to perform inference on generative models specified in any of its supported frameworks (Stan, PyMC3, TensorFlow, and vanilla Python at the time of writing). Though it aims to be agnostic to the modeling language, Edward itself is built on top of TensorFlow. This has the advantage of enabling integration of probabilistic programming paradigms with other machine learning algorithms, e.g. through Bayesian optimization of algorithm hyperparameters, all of which can be expressed in TensorFlow and automatically parallelized across multiple devices (CPU and/or GPU).

Edward's principal developer is Dustin Tran, a graduate student at Columbia. It comes with a strong academic heritage and shares some contributors with Stan, so we expect it to come into its own in due time. At present, though, it is intended to be an experimental test bed for complex models, large data, and model fine-tuning.

### 5.1.4 Anglican

Anglican<sup>6</sup> is a probabilistic programming language built into Clojure, which gives it a few nice features. As with PyMC3, you don't need to compile the model, which means it is comparatively very easy to improve the model iteratively. Once specified, the model can be combined with observed data to generate a posterior distribution that can be sampled from at will. The nice thing about this, relative to Stan, is that the

---

<sup>5</sup> <https://github.com/blei-lab/edward/>

<sup>6</sup> <http://www.robots.ox.ac.uk/~fwood/anglican/>

number of samples does not have to be determined beforehand. It is simple to resume sampling if, for example, it seems that the samples have not completed the burn-in period.

Anglican fits very neatly into Clojure, which makes it possible to interact with and feed data into Anglican models easily. In fact, Anglican is defined mostly in terms of new Clojure macros and functions. Because everything in Anglican is native to Clojure, it integrates with Clojure much more smoothly than even PyMC integrates with Python. However, the Clojure data science environment is not nearly as well developed as the Python and R ecosystems.

Also worth noting is that the developers of Anglican come from a computer science rather than a statistics background. While this may account for some of the elegance of Anglican as a programming language, it also likely accounts for the fact that Anglican is lagging behind some competitors in terms of the methods of inference that are built into the language. Most notably, it is missing support for both ADVI and NUTS.

### 5.1.5 emcee

`emcee`<sup>7</sup> does not aim to be a fully featured probabilistic programming language. Instead, it's simply a very nice, lightweight implementation of the MCMC algorithm proposed by Goodman & Weare (2010).<sup>8</sup> The module's author tweaked the algorithm to make it parallelizable.<sup>9</sup>

---

<sup>7</sup> <http://dan.iel.fm/emcee/current>

<sup>8</sup> <http://msp.org/camcos/2010/5-1/camcos-v5-n1-p04-s.pdf>

<sup>9</sup> See <http://jakevdp.github.io/blog/2014/06/14/frequentism-and-bayesianism-4-bayesian-in-python/>.

### 5.1.6 BayesDB

BayesDB<sup>10</sup> is an active project from the Probabilistic Computing Project at MIT that augments SQL-like databases with the ability to do automatic inference. For example, data can be inserted that has missing values and those missing values can be inferred using other rows. In addition, new data can be simulated using existing data in order to explore hypothetical situations and to get a better sense of the processes generating the data.

At its core, BayesDB attempts to bring the capabilities of Bayesian inference to a wider audience by eliminating the need for any modeling. This is done with a novel algorithm that is able to determine subtle structure in datasets in order to build up large numbers of possible models that all get used together.

While the approach is interesting, the final product is slow and not well documented. When we attempted to use the software, we ran into bugs and very high resource utilization for small datasets, leading us to question the usefulness of the current implementation of this solution.

That being said, this is a project to keep an eye on since any future success the developers have at improving their system could be very important for bringing Bayesian inference to a non-expert audience.

### 5.1.7 BUGS and JAGS

BUGS and JAGS are early probabilistic programming languages. BUGS was written in Pascal. JAGS was written in C++.

---

<sup>10</sup> <http://probcomp.csail.mit.edu/bayesdb/>



Neither are still under active development. They both use Gibbs sampling, which is slower than NUTS or ADVI for large problems but does work with discrete parameters. Their language is more constrained and even more declarative than Stan's, which means that both remain popular teaching tools.<sup>11</sup>

## 5.2 Probabilistic Programming in the Wild

Here we discuss some real-world applications of probabilistic programming and the groups that are using it.

### 5.2.1 Stan Group Inc.

Stan Group Inc. is a new consultancy that supports companies and individuals intending to use, or using, the Stan programming language as they move toward Bayesian methods for their analytical work. Founded in early 2016, the company offers Stan and Bayesian training courses, a subscription for Stan statistical support, and custom model development and deployment.

Two of the four founders of Stan Group are Stan developers, and all four have backgrounds in either statistics or data science. On the scientific advisory side, Andrew Gelman, a professor of statistics and political science and director of the Applied Statistics Center at Columbia University, has published numerous books on Bayesian methods. Bob Carpenter designed the Stan programming language and is part of the core Stan developer team, alongside Michael Betancourt.

---

<sup>11</sup> See e.g. *STATS131* (<https://www.stat.auckland.ac.nz/~brewer/stats331.pdf>) by Brendon Brewer, our favorite elementary introduction to Bayesian data analysis.

One of the Stan Group's projects involves helping a major pharmaceutical company determine drug dosages using Bayesian methods implemented in Stan. In particular, they model how drugs are absorbed into the body (i.e., pharmacokinetics). The selection of a drug dosage regimen is based on estimates of the patient's pharmacokinetic parameters, which govern how quickly the body can absorb the drug (weight, age, sex, etc.). Using the Bayesian approach, they estimate a patient's pharmacokinetic parameters from measured drug levels using the a priori pharmacokinetic parameters of the population model as a starting point. Then they adjust the starting estimates based on the patient's measured drug levels. Combining knowledge about the individual with knowledge about the population leads to more precise estimates than using either dataset by itself, which results in a cheaper, safer, and more effective regimen.

Stan Group has also worked on pricing models and customer segmentation in collaboration with a publishing house, and they work with hedge funds and other players in the financial space, a heavily regulated industry where the interpretability of Bayesian models is a critical feature.

### 5.2.2 Charles River Analytics

Charles River Analytics (CRA) is a consultancy working on research projects primarily with the Department of Defense and DARPA. Their products include computer vision systems and expert and decision support tools. They provide outsourced software engineering and development services including custom solutions and data models, autonomous robotic platforms, and multiagent systems for information

management and command and control applications.

CRA's principal scientist, Avi Pfeffer, has been working for over 20 years on Figaro,<sup>12</sup> an open-source probabilistic programming language implemented in Scala. Figaro has helped build hydrological terrain models for army logistics. Given data on rainfall, water runoff, and terrain conditions, it allowed domain experts with limited knowledge of Bayesian methods to quickly build an application for optimal placement of fuel tanks, fuel pipelines, and water tanks. One of the goals for future development of the Figaro language is to further reduce the required input of experts in probabilistic programming and/or Bayesian methods. The developers hope to enable domain experts to encode their knowledge into the model so that Figaro can automatically infer the model specifics, handling the complexities of the math under the hood.

According to Pfeffer, the team at CRA invented an algorithm used in radar tracking that would not have occurred to them without probabilistic programming. They were working on the problem of tracklet merging, part of target tracking, where given sensor data (e.g., radar images) you want to identify objects' trajectories (e.g., aircraft). In a first step, you define the tracklets (i.e., the locations of the objects in still images). In a second step, you reconstruct the objects' trajectories by stitching together the tracklets. For each tracklet, there are several likely successor tracklets modeled by a distribution over candidates. For each successor, there are several likely predecessor tracklets, again modeled by a distribution over predecessors. The constraint, of course, is that the

predecessor needs to be the original tracklet. In formal terms, that constraint requires chaining together two random variables, a fundamental operation in Figaro but difficult outside the realm of probabilistic programming.

### 5.2.3 Stitch Fix

Stitch Fix is a fashion retailer providing a subscription-based personalized styling service. During signup, customers provide information about their personal style preferences alongside basic information such as height and weight. Stitch Fix sends customers a personalized box containing items of clothing based on their stated preferences. Customers keep the items they like and return the rest.

At Stitch Fix, algorithms and human stylists work hand-in-hand (if algorithms had hands, that is): algorithms make clothing recommendations to human stylists, who fine-tune the final content of the Stitch Fix boxes prior to shipment. The team at Stitch Fix uses Bayesian methods and probabilistic programming to help improve recommendations.

Bayesian methods allow learning from small datasets, and Stitch Fix does not have much data per individual customer. (Music streaming services, for example, have much more data to work with because people tend to listen to different songs much more frequently than they buy new articles of clothing.) Furthermore, these methods allow Stitch Fix to integrate the information it does have about individuals with population trends to estimate style preferences. Models are further improved by utilizing hierarchical information. For example, linen pants are a type of pant. If Stitch Fix learns a customer dislikes all pants, there is no need to recommend linen pants.

---

<sup>12</sup> See <https://www.cra.com/work/case-studies/figaro>.

Bayesian models and probabilistic programming languages allow easy encoding of such hierarchical relationships. Finally, the confidence/uncertainty estimates baked in via the posterior allow Stitch Fix to determine where human assistance is needed. Human stylists are expensive, and Bayesian methods allow Stitch Fix to allocate stylists' expertise where it's needed the most.

Despite its benefits, Bayesian methods are not used as widely at Stitch Fix as one might think. The team is excited about Bayesian methods, but like us, they found Bayesian models difficult to productionalize (see our discussion of this issue in [4.2.4 Loan Model Deployment](#) and [4.4 PyMC3](#)). For now, the team has devised clever work arounds, they use Bayesian models to estimate coefficients they subsequently use in production, and they agree with our assessment that developers of probabilistic programming languages need to remove obstacles for productionalizing Bayesian models to further adoption (see our suggestions in [4.4 PyMC3](#)).

#### 5.2.4 REsurety

REsurety is a risk management company that specializes in limiting the risks of wind power generation. The upfront costs of building a wind farm are substantial, and wind power generators face two primary risks: fluctuating energy prices and the variability of wind. REsurety offers a guaranteed income stream (i.e., a surety) to companies that build wind farms, limiting the risk of sinking money into building a wind farm only to have the wind not blow profitably or having energy prices drop to unprofitable levels for power generators.

For REsurety to price its guarantees, it needs to evaluate

the wind and energy price risks. These risks, both wind risk and price risk, are modeled using Bayesian inference. REsurety models wind risk in great detail, predicting wind every hour for years into the future. Likewise, they model prices at a very granular level.

REsurety uses the R language for all of its models, but has noted that in hindsight, Python seems to be a more attractive option.

#### 5.2.5 Kickoff.ai

Kickoff.ai is a software package that attempts to predict soccer match outcomes. It was built as a research project by a team of machine learning academics in Switzerland. The software uses Bayesian inference at both the player and team levels to predict winners, and because it is at its core Bayesian, it also provides confidence intervals, which can be turned into betting odds.

Kickoff.ai is a bit like fantasy soccer in that it treats a team as a collection of players rather than a monolithic entity. The use of a team model that comprises individual player models gives Kickoff.ai a great deal more source data to work with because its inputs are not limited to matches in which an identical team was playing. Instead, it can use any game in which a player appeared and model the player, then use the models for each player to predict the outcome of a future match that includes that player.

In the 2016 UEFA Euro Cup, Kickoff.ai predicted 65% of the match outcomes correctly, beating some bookmakers' predictions. When Kickoff.ai's confidence intervals were used to make odds, it did almost as well as professional bookmakers.

## CHAPTER 6

### Ethics

A common obstacle organizations face when building data science programs is that leadership and nontechnical personnel distrust the accuracy of the organization's data and, by extension, the outputs of analysis.<sup>1</sup> Discomfort with black-box models, like the neural networks we studied in our last two reports,<sup>2</sup> only exacerbates these fears: leadership in regulated industries (like life sciences and financial services) are required to explain why models generated results that impact the consumers of their services, and leadership in non-regulated industries often prefer to understand the factors that lead them to take critical business decisions supported by data.

As discussed in 2 Bayesian Data Analysis, Bayesian data analysis can be a good option to win leadership's trust in data science because it is interpretable by construction. Interpret-

---

<sup>1</sup> In a recent survey conducted by KPMG (<https://assets.kpmg.com/content/dam/kpmg/pdf/2016/07/2016-ceo-survey.pdf>), 1 out of 10 CEOs indicated they "actively distrust their organization's use of data and analytics," with life sciences and banking CEOs expressing the highest degree of distrust.

<sup>2</sup> See Deep Learning: Image Analysis (FF03) and Summarization (FF04).

ability, however, is a far cry from certainty. The first step for most organizations to succeed with data science—particularly with the rich distributions provided by Bayesian analysis—is to get comfortable with uncertainty. In practice, this means developing processes and cultural habits to guide decisions based on confidence levels, probabilities, and risks.

The ethical dilemmas of probabilistic programming stem from the same issues organizations face whenever they make decisions in the presence of uncertainty. These issues include the influence of priors, the interpretation of posteriors, and, as with all data science, managing data privacy.

### 6.1 Choosing the Prior

Proponents of Bayesian data analysis support including expert information about a domain to inform analysis of current data. After all, that's the power of the technique! But using priors without biasing results can be tricky, and the practice is not accepted by all statisticians.

In a 2011 interview,<sup>3</sup> David Cox (a statistician) and Deborah Mayo (a philosopher of science) discussed whether it's sound to reference prior information when analyzing current data. Cox mentioned a rule set by 20th-century scientist Ronald Fisher that states that it's all right to use prior information in the design of data collection, but not in data analysis. For acceptable priors in designing an experiment, he cites things like "a belief that certain situations are likely to give similar outcomes, or a belief that studying this effect is likely to be

interesting." In essence, we can start an experiment with the prior belief that the data is worth studying to answer some question, but we should then let the data speak for itself.

On the surface, this approach appears to support empirical objectivity. But it can fall short. In a commentary on Cox and Mayo's interview, Andrew Gelman (one of Stan's creators) recounts a famous study where "a sample of a few thousand people was analyzed, and it was found that the most beautiful parents were 8 percentage points more likely to have girls, compared to less attractive parents."<sup>4</sup> The results may have been statistically significant for this small sample, but would have been reduced to noise when projected onto the entire population. Importantly, priors would have shown that the difference in sex ratios in the general population, comparing beautiful to less-beautiful parents, was less than 1 percent. In this situation, therefore, the prior has higher value than the observed data, and could have helped the researcher make the right conclusion about the population at large and evaluate how representative the sample actually was.

In this example, the prior was constructed from past statistical analysis: our expert opinion was formed from the result of previous statistical tests. But things get more delicate if our domain expertise is based mainly on personal opinion. As Cox continues in the 2011 interview: "There are situations where it is very clear that whatever a scientist or statistician might do privately in looking at data, when they present their information to the public or government department or whatever,

---

<sup>3</sup> See [http://www.phil.ut.edu/dmayo/personal\\_website/Cox-Mayo-2011.pdf](http://www.phil.ut.edu/dmayo/personal_website/Cox-Mayo-2011.pdf).

---

<sup>4</sup> See <http://www.stat.columbia.edu/~gelman/research/published/ChanceEthics5.pdf>.

they should absolutely not use prior information, because the prior opinions of some on these prickly issues of public policy can often be highly contentious with different people with strong and very conflicting views." Algorithmic policing tools like those provided by Northpointe bring this point home. In its report on these tools, Propublica uncovered that implicit bias in the algorithms linked race to crime rates, skewing risk scores to penalize black defendants.<sup>5</sup> Long-standing social inequalities, therefore, can create priors we choose to correct for when making our models.

Finally, we need to be careful that overconfidence in the prior does not impact the rigor of research. The US Food & Drug Administration (FDA), for example, has published formal guidance on the use of Bayesian statistics in medical device clinical trials.<sup>6</sup> This document warns about using priors "based mainly on personal opinion derived by the elicitation of experts" and changing prior information mid-trial, which "may imperil the scientific validity of the trial results." As using prior information can lead to shorter trials on smaller sample populations, or produce different decisions about whether a device is safe, trial designers must be prepared to defend their prior choices and even meet with the FDA to approval trial designs using Bayesian techniques.

---

<sup>5</sup> See <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing>.

<sup>6</sup> Published in 2010, the document cites WinBUGS as a probabilistic programming package practitioners can consider to implement a Bayesian clinical trial methodology. See <http://www.fda.gov/RegulatoryInformation/Guidances/ucm071072.htm#2>.

## 6.2 Interpreting and Communicating the Posterior

When applying Bayesian data analysis in your organization, it's important to remember that most people will be uncomfortable reasoning in probabilities. An anecdote featuring President Obama does a nice job of illustrating how distributions can frustrate people. "Throughout Spring 2011," write Jeffrey Friedman and Richard Zeckhauser, "intelligence analysts and officials debated the chances that Osama bin Laden was living in Abbottabad, Pakistan."<sup>7</sup> Opinions on the likelihood bin Laden was there varied widely, between 60-95%; confidence levels in these opinions varied between 30-80%. Obama complained that his advisers offered "probabilities that disguised uncertainty as opposed to actually providing you with more useful information," ultimately concluding the odds that bin Laden was in Abbottabad were 50/50.

Obama is no dummy. The anecdote shows that people have an easier time wrapping their heads around maximum likelihoods than a full distribution of possibilities. As such, when using Bayesian methods, you should think carefully about how to communicate the output of posteriors to help others in your organization make decisions.

For example, recall how with the Lending Club model (see [4.2.1 Loan Data](#)) we used only 7 of the over 100 possible features in building our model to help make the loan decision

---

<sup>7</sup> See [https://www.hks.harvard.edu/fs/rzeckhau/Estimative\\_Probability.pdf](https://www.hks.harvard.edu/fs/rzeckhau/Estimative_Probability.pdf). In a related vein, the CIA wrote a classic 1964 study on the ambiguities of "words of estimative probability" that shows how different people ascribe different quantitative probabilities to qualitative terms like "serious possibility" or "almost certain."

transparent for the borrower. While the absolute accuracy of our models improves with additional features, there are certain problems, like granting loans, admitting candidates to jobs, or assessing the risk of individuals, where it's inappropriate or even illegal to use all the available information. Simpler input can generate outputs that are easier to understand.

To leverage the full predictive power of Bayesian methods, it's likely worthwhile to take the time to educate your colleagues or users on the basic concepts behind the models. Users don't need a doctorate in statistics, but should have a basic understanding of:

- How the posterior was generated from the prior and the likelihood
- What affiliated confidence rates mean, and how they should treat them in making decisions
- What probability and confidence thresholds must be met for the organization to take action or not
- When more data and analysis is required to inform a decision with a sufficient confidence level

It can help for data scientists to work together with non-technical business leaders to set expectations for users. With basic understanding, users can interpret outputs themselves, which reduces confusion and eventual pushback against the data science team.

### 6.3 Managing Data Privacy

When selecting priors, it's useful to piggyback on work done by fellow practitioners in a given domain. If hospitals, healthcare researchers, financial services organizations, or

even Google were to openly publish and exchange model parameters, it would be a big win for the performance of future modeling! Increasingly, however, models are built using sensitive data. And while protected health information (PHI) or other personally identifiable information (PII) cannot be shared legally, sometimes parameters in our models leave traces about the individuals in a validation or training set. As such, we have to take measures to make sure models built on sensitive personal data don't inadvertently compromise privacy.

One increasingly popular technique for protecting privacy in statistical models is called "differential privacy." In essence, "differential privacy is the statistical science of trying to learn as much as possible about a group while learning as little as possible about any individual in it."<sup>8</sup> More formally, given an algorithm A that outputs a value B when run on a dataset C, the goal of differential privacy is to hide the effect of a small change in C on the output of A. If we have two databases that differ in the value of a single individual, we must include some randomness to guarantee that any algorithmic output (e.g., posterior parameters) cannot be traced back to that individual.

Researchers are actively working on building differentially private probabilistic programming (e.g., PrivInfer is a framework for reasoning about differential privacy on exist-

---

<sup>8</sup> See <https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/>. See also <https://blog.cryptographyengineering.com/2016/06/15/what-is-differential-privacy/> for an introductory overview.



ing probabilistic programming languages<sup>9</sup>). In Bayesian inference, differential privacy can be achieved by:

- Adding noise to the input data
- Adding noise to the parameters (this requires bounding the sensitivity of the output parameters when we change a single data sample, relying on specific properties of the model and the prior distribution)
- Taking samples from the posterior distribution instead of releasing the entire posterior distribution (this requires no noise during inference, but is highly specific to the model and prior and is not yet a rigorous means to guarantee privacy)

Something to keep in mind when exploring differential privacy is that "the devil is in the budget": the more information we ask of our database, the more noise has to be injected to minimize data leakage. The size of the budget depends on the sensitivity of our model parameters.

## 6.4 Conclusion

Probabilistic programming provides powerful, practical advantages over other machine learning techniques, given its ability to draw meaningful, interpretable conclusions from small datasets. But this power depends on our ability to appropriately use unbiased prior distributions. Doing this ethically requires thought because our choices impact model results and risk compromising individual privacy.

---

<sup>9</sup> See <https://arxiv.org/abs/1605.00283>.

## CHAPTER 7 Future

Probabilistic programming languages offer their users the ability to accomplish incredibly complex calculations with relative ease. Currently, however, these capabilities have some limitations: integration with other systems (such as deployment systems) and the amount of data that can be processed quickly is limited.

In the near future, we predict that ADVI (see [3.4 A Sampling Algorithm for the Future: ADVI](#)) will become more robust and more accurate. Currently the main limitations in ADVI come from its memory use and accuracy. While it does provide fantastic approximate results very quickly, there is a huge effort in the community to increase accuracy so that there is never a need to use more resource intensive sampling algorithms.

This push for more powerful ADVI will address the limitation of data scalability. This is especially true because ADVI can be run as an online algorithm, allowing models to update in realtime. Future products can take advantage of this by having very complex hierarchical models where the research models and deployed models share the exact same code and are simply reading data from different sources.

Lastly, it's important to note that the most important accomplishment of probabilistic programming is bringing in-



ferential reasoning to the masses. By providing easy mechanisms to do Bayesian inference on problems, data scientists will start approaching problems in more rigorous ways without the mathematical overhead that was normally associated with such methods (particularly by always considering the implications of a model they are defining).

Moving more in this direction, we can imagine simpler interfaces into these methods in the future. For example, graphical interfaces where assumptions about causal relationships can be drawn and the system iterates through possible models in order to assess the assumptions. This naturally leads to a Star-Trek-like computer system where you can speak to your computer in order to inspect the probable implications of data.

## 7.1 Probabilistic Sci-Fi: BayesHead 5000



Customer Assistance Office

Re: Inquiry on Limitations of the BayesHead 5000

Dear Mr. Jim,

We here at Monte Carlo Inc. are reaching out in response to your inquiry dated September 15. We want you to know that we appreciate your business. You are a valued customer and, though confused about several aspects of our products, you strike me as a decent and intelligent consumer. Personally, I can understand how

someone without thorough knowledge of this cutting edge technology could find themselves in the very predicament that prompted your letter.

However, I do not mean to equivocate, Mr. Jim. We will not be reimbursing you for the BayesHead 5000. Nor will we be covering your costs for surgical removal. While there are several technical reasons that we cannot accommodate your request, I believe in this particular instance there may be value in addressing your recent error and contextualizing our role in it. After all, what kind of corporation would we be if we implanted our products in your inner ear canal and then failed to address your concerns in such a time of distress!

Firstly, I sympathize with you and all our clients who have been banned from traditional gambling venues. Although we take great pride at Monte Carlo Inc. in having granted our customers the almost mystical prescience that has led to these regulations, we are pained to learn that many customers, such as yourself, have now turned the events of their own lives —personal disputes with loved ones, even their health—into venues for wagering their hard earned savings. It is perfectly understandable, given the BayesHead's sophistication, that those who have invested not only in the product but the "costly operation" (as you put it), would begin to see such opportunities in their daily lives.

But it is my feeling, Mr. Jim, that it is unwise to place bets on your daughter's, and her friend's, college applications. It is especially unwise in your situation, for a number of reasons, which I will investigate in the remainder of this letter.

To begin, I wonder how certain you are that the other fathers in the pool had either no BayesHead, or a version of the Bayes-

Head older than your own. I'm sure you are aware that we recently released the BayesHead 6000, which has some remarkable new functionality. It has, in fact, transformed the whole landscape of probabilistic programming, rendering your BayesHead as more of a liability, evidenced by the false sense of security it seems to have offered you in your wager. You see, while the BayesHead 5000 will input your assumptions and, using Bayesian statistical analysis and advanced scraping tools, model the probable outcomes of a given situation, the BayesHead 6000 has much more advanced functionality. How could we possibly improve on the BayesHead 5000 you are wondering? With the BayesHead 6000, our product has actually inverted its process. Rather than taking your assumptions as capital t Truth, and manipulating the data accordingly, the BayesHead 6000 treats the available data as capital t Truth. It uses your assumptions as inputs in its algorithm, but then uses the existing data to correct your assumptions, in order to allow you to adjust your model and improve your predictions.

Take, for example, the bets you and Greg placed on your daughters. There are certain quantitative measures that are fairly standard, which don't require any prior assumptions: GPA, SAT score, list of extracurriculars and foreign languages. But then there is the interview, and the personal essay. The BayesHead needs your assumptions to measure how these variables will impact the committee. And here you are at a great disadvantage. Because while Greg, with the BayesHead 6000 (surely you know by now that he has one), was able to have his opinion of his daughter corrected with real world data taken from both her online presence and her own BayesHead 6000 (the Wilsons have the family plan), your BayesHead 5000 does not have the functionality to correct your per-

spective on your daughter's charisma and intellectual curiosity.

Luckily, with an upgrade to the BayesHead 6000, you'll no longer be in danger of overestimating your child's abilities. And, for a limited time, we are offering free installation. Though, do note, as there has been some confusion about this lately, that standard fees apply to removal of the BayesHead 5000. In my opinion (if I may speak from personal experience) it is actually preferable to keep the old technology implanted, as there can be a sort of uncomfortable phantom itch that recurs with removal.

I have to say, Mr. Jim, I do sympathize with your position, and especially with your daughter. While we cannot accept responsibility for your predicament, we understand your frustration that your confidence in our product has resulted in a depletion of your daughter's tuition fund (although we cannot ignore the silver lining that perhaps she will perform as you initially had hoped! now that she has a chance for another round of applications). That said, if it is her desire to attend a safety school this year, I would like to refer you to the student loan division of Monte Carlo Inc. They offer extremely reasonable interest rates and, as I'll let them explain, significant discounts when loans are paired with a BayesHead upgrade.

Best Regards,

Gary

Sales Associate

Monte Carlo Inc.

*Luck: It's all in your head*

## CHAPTER 8

# Conclusion

Probabilistic programming makes Bayesian inference accessible. It allows you to build interpretable models that quantify uncertainty and make use of both data and domain knowledge.

In this report, we've explored the history and foundation of these techniques ([2 Bayesian Data Analysis](#), [3 Probabilistic Programming](#)). We include a broad survey of the open source tools available today and where they are heading, as well as exploring emerging application areas in industries from fashion to sports to finance ([5 Tools and Products](#)).

We present two prototypes that explore the potential of these techniques ([4 Prototypes](#)). Our real estate prototype explores techniques for inferring prices for properties that are not in the data set, including both properties in neighborhoods with sparse data and properties as they may exist in the future, should present trends hold steady. The loan prototype explores the risk profiles for consumer lending, demonstrating the power of these techniques for a portfolio of outcomes.

In the next few years, we expect the algorithms we discuss in this report to become more effective and the tools to employ them to become commonplace and accessible, leading to a significant improvement in how we combine our domain

knowledge and experience with data to learn about the world and make more effective decisions.

## APPENDIX

### Frequentism

In 2 Bayesian Data Analysis we described one way to draw conclusions from incomplete, imperfect data. The frequentist (or classical) approach that dominated statistics during the 20th century (until Bayes' revival) offers a different approach.

Roughly speaking, it begins by defining a "null hypothesis". In the case of the A/B test we saw in 2 Bayesian Data Analysis, that hypothesis might be that layouts A and B have the same conversion fractions. We then calculate how unlikely our observations are, assuming the null hypothesis is true. If that probability, the p-value, is below some threshold (canonically 0.05) then you reject the null hypothesis. In a more complicated experiment you might need to calculate a different statistic than the p-value.<sup>1</sup>

A lot of ink has been shed over the relative merits of this approach and Bayesianism. Our view is that the debate is over: Bayesians won. The frequentist approach is hamstrung by the fact that it attempts to use the probability of the data given the hypothesis  $P(D|H)$  to answer questions about

---

<sup>1</sup> See the talk "Statistics For Hackers" for a tour of the possibilities, <https://speakerdeck.com/jakevdp/statistics-for-hackers>.

probability of the hypothesis given the data  $P(H|D)$ .<sup>2</sup> In some ways, frequentist statistics is a collection of fragile methods to work around this mismatch.

---

<sup>2</sup> See "The problem with p-values" for a more complete discussion of this distinction, <https://aeon.co/essays/it-s-time-for-science-to-abandon-the-term-statistically-significant>.

## About Fast Forward Labs

Fast Forward Labs is a research company that will help you recognize and develop new product and business opportunities through emerging technologies.

The November 2016 Fast Forward Labs report on Probabilistic Programming is brought to you by Grant Custer, Micha Gorelick, Kathryn Hume, Hilary Mason, Ryan Micallef, Friederike Schüür, Nick Vermeer, and Mike Williams.

<http://www.fastforwardlabs.com>