SAMPLE CHAPTER

Practical Probabilistic Programming

Avi Pfeffer

FOREWORD BY Stuart Russell

MANNING

www.itbook.store/books/9781617292330

Practical Probabilistic Programming

by Avi Pfeffer

Chapter 1

Copyright 2016 Manning Publications

www.itbook.store/books/9781617292330

brief contents

PART 1 INTRODUCING PROBABILISTIC PROGRAMMING AND FIGARO 1

1 ■ Probabilistic programming in a nutshell 3
2 ■ A quick Figaro tutorial 27
3 ■ Creating a probabilistic programming application 57

PART 2 WRITING PROBABILISTIC PROGRAMS 91

4 ■ Probabilistic models and probabilistic programs 93
5 ■ Modeling dependencies with Bayesian and Markov networks 129
6 ■ Using Scala and Figaro collections to build up models 172
7 ■ Object-oriented probabilistic modeling 200
8 ■ Modeling dynamic systems 229

www.itbook.store/books/9781617292330

PART 3 INFERENCE 255

9 ■ The three rules of probabilistic inference 257
10 ■ Factored inference algorithms 283
11 ■ Sampling algorithms 321
12 ■ Solving other inference tasks 360
13 ■ Dynamic reasoning and parameter learning 382

www.itbook.store/books/9781617292330

1 Probabilistic programming in a nutshell

This chapter covers

- What is probabilistic programming?
- Why should I care about it? Why should my boss care?
- How does it work?
- Figaro—a system for probabilistic programming
- A comparison between writing a probabilistic application with and without probabilistic programming

In this chapter, you'll learn how to make everyday decisions by using a probabilistic model and an inference algorithm—the two main components of a probabilistic reasoning system. You'll also see how modern probabilistic programming languages make creating such reasoning systems far easier than a general-purpose language such as Java or Python would. This chapter also introduces *Figaro*, the probabilistic programming language based on Scala that's used throughout the book.

www.itbook.store/books/9781617292330

1.1 What is probabilistic programming?

Probabilistic programming is a way to create systems that help us make decisions in the face of uncertainty. Lots of everyday decisions involve judgment in determining relevant factors that we don't directly observe. Historically, one way to help make decisions under uncertainty has been to use a probabilistic reasoning system. *Probabilistic reasoning* combines our knowledge of a situation with the laws of probability to determine those unobserved factors that are critical to the decision. Until recently, probabilistic reasoning systems have been limited in scope, and have been hard to apply to many real-world situations. Probabilistic programming is a new approach that makes probabilistic reasoning systems easier to build and more widely applicable.

To understand probabilistic programming, you'll start by looking at decision making under uncertainty and the judgment calls involved. Then you'll see how probabilistic reasoning can help you make these decisions. You'll look at three specific kinds of reasoning that probabilistic reasoning systems can do. Then you'll be able to understand probabilistic programming and how it can be used to build probabilistic reasoning systems through the power of programming languages.

1.1.1 How do we make judgment calls?

In the real world, the questions we care about rarely have clear yes-or-no answers. If you're launching a new product, for example, you want to know whether it will sell well. You might think it will be successful, because you believe it's well designed and your market research indicates a need for it, but you can't be sure. Maybe your competitor will come out with an even better product, or maybe it has a fatal flaw that will turn off the market, or maybe the economy will take a sudden turn for the worse. If you require being 100% sure, you won't be able to make the decision of whether to launch the product (see figure 1.1).

The language of probability can help make decisions like these. When launching a product, you can use prior experience with similar products to estimate the probability that the product will be successful. You can then use this probability to help decide whether to go ahead and launch the product. You might care not only about whether the product will be successful, but also about how much revenue it will bring, or alternatively, how much you'll lose if it fails. You can use the probabilities of different outcomes to make better-informed decisions.

Okay, so probabilistic thinking can help you make hard decisions and judgment calls. But how do you do that? The general principal is expressed in the Fact note.

FACT A judgment call is based on *knowledge* + *logic*.

You have some knowledge of the problem you're interested in. For example, you know a lot about your product, and you might have done some market research to find out what customers want. You also might have some intelligence about your competitors and access to economic predictions. Meanwhile, logic helps you get answers to your questions by using your knowledge.

www.itbook.store/books/9781617292330

Figure 1.1 Last year everyone loved my product, but what will happen next year?

You need a way of specifying the knowledge, and you need logic for getting answers to your questions by using the knowledge. Probabilistic programming is all about providing ways to specify the knowledge and logic to answer questions. Before I describe what a probabilistic programming system is, I'll describe the more general concept of a probabilistic reasoning system, which provides the basic means to specify knowledge and provide logic.

1.1.2 *Probabilistic reasoning systems help make decisions*

Probabilistic reasoning is an approach that uses a model of your domain to make decisions under uncertainty. Let's take an example from the world of soccer. Suppose the statistics show that 9% of corner kicks result in a goal. You're tasked with predicting the outcome of a particular corner kick. The attacking team's center forward is 6' 4'' and known for her heading ability. The defending team's regular goalkeeper was just carted off on a stretcher and has been replaced by a substitute playing her first game.

www.itbook.store/books/9781617292330

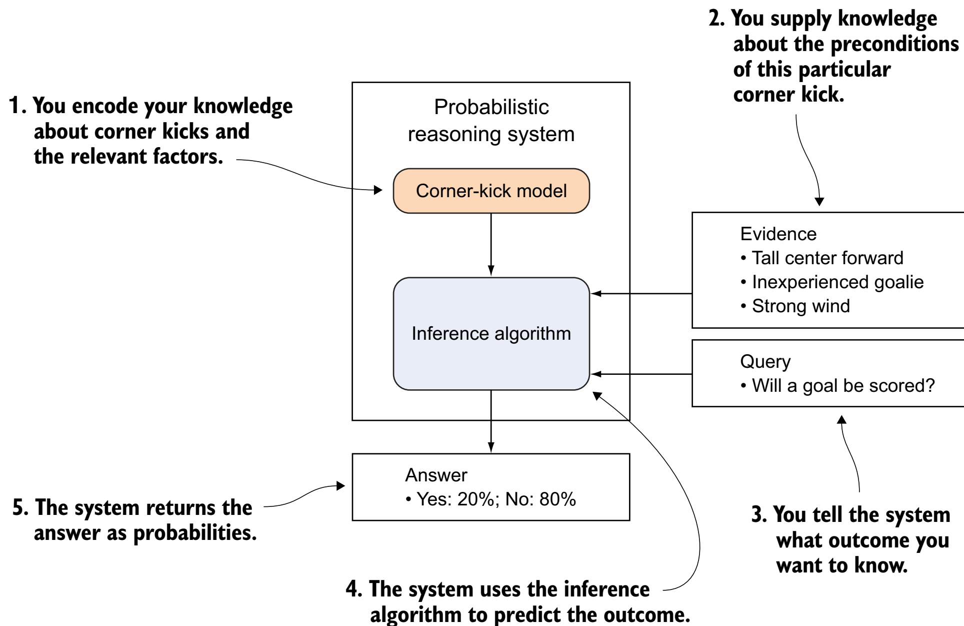

Figure 1.2 How a probabilistic reasoning system predicts the outcome of a corner kick

Besides that, there's a howling wind that makes it difficult to control long kicks. So how do you figure out the probability?

Figure 1.2 shows how to use a probabilistic reasoning system to find the answer. You encode your knowledge about corner kicks and all the relevant factors in a corner-kick model. You then supply evidence about this particular corner kick, namely, that the center forward is tall, the goalie is inexperienced, and the wind is strong. You tell the system that you want to know whether a goal will be scored. The inference algorithm returns the answer that a goal will be scored with 20% probability.

KEY DEFINITIONS

General knowledge—What you know to hold true of your domain in general terms, without considering the details of a particular situation

Probabilistic model—An encoding of general knowledge about a domain in quantitative, probabilistic terms

Evidence—Specific information you have about a particular situation

Query—A property of the situation you want to know

Inference—The process of using a probabilistic model to answer a query, given evidence

www.itbook.store/books/9781617292330

In probabilistic reasoning, you create a *model* that captures all the relevant general knowledge of your domain in quantitative, probabilistic terms. In our example, the model might be a description of a corner-kick situation and all the relevant aspects of players and conditions that affect the outcome. Then, for a particular situation, you apply the model to any specific information you have to draw conclusions. This specific information is called the *evidence*. In this example, the evidence is that the center forward is tall, the goalie is inexperienced, and the wind is strong. The conclusions you draw can help you make decisions—for example, whether you should get a different goalie for the next game. The conclusions themselves are framed probabilistically, like the probability of different skill levels of the goalie.

The relationship between the model, the information you provide, and the answers to queries is well defined mathematically by the laws of probability. The process of using the model to answer queries based on the evidence is called *probabilistic inference*, or simply *inference*. Fortunately, computer algorithms have been developed that do the math for you and make all the necessary calculations automatically. These algorithms are called *inference algorithms*.

Figure 1.3 summarizes what you've learned.

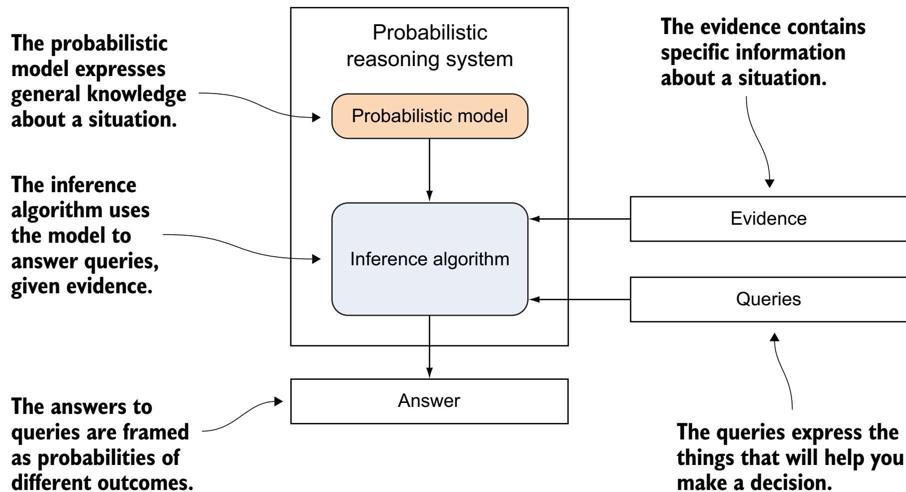

Figure 1.3 The basic components of a probabilistic reasoning system

In a nutshell, what we've just discussed are the constituents of a probabilistic reasoning system and how you interact with one. But what can you do with such a system? How does it help you make decisions? The next section describes three kinds of reasoning that can be performed by a probabilistic reasoning system.

www.itbook.store/books/9781617292330

1.1.3 Probabilistic reasoning systems can reason in three ways

Probabilistic reasoning systems are flexible. They can answer queries about any aspect of your situation, given evidence about any other aspect. In practice, probabilistic reasoning systems perform three kinds of reasoning:

- *Predict future events.* You've already seen this in figure 1.2, where you predict whether a goal will be scored based on the current situation. Your evidence will typically consist of information about the current situation, such as the height of the center forward, the experience of the goalie, and the strength of the wind.
- *Infer the cause of events.* Fast-forward 10 seconds. The tall center forward just scored a goal with a header, squirting under the body of the goalie. What do you think of this rookie goalkeeper, given this evidence? Can you conclude that she's poorly skilled? Figure 1.4 shows how to use a probabilistic reasoning system to answer this question. The model is the same corner-kick model you used before to predict whether a goal would be scored. (This is a useful property of probabilistic reasoning: the same model that can be used to predict a future result can be used after the fact to infer what caused that result.) The evidence here is the same as before, together with the fact that a goal was scored. The query is the skill level of the goalie, and the answer provides the probability of various skill levels.

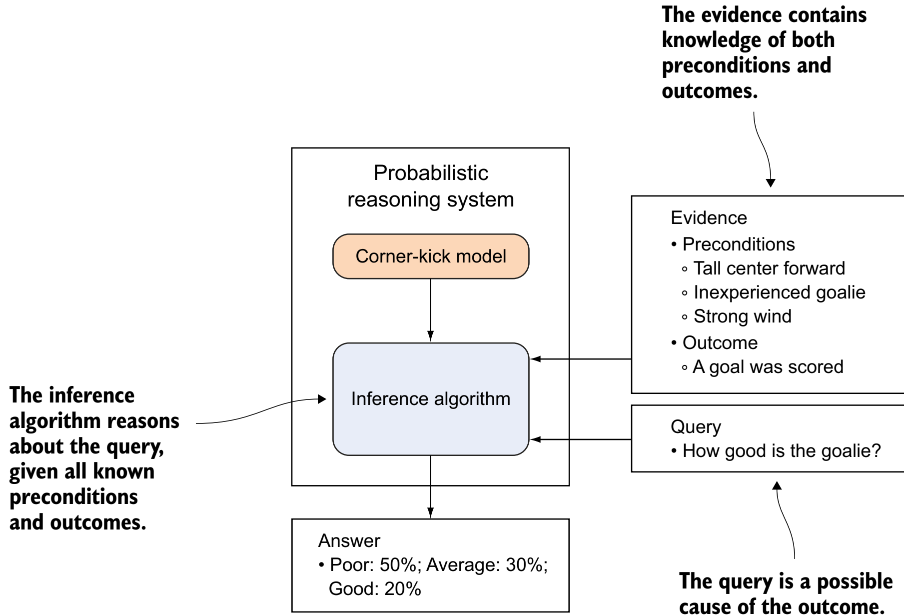

Figure 1.4 By altering the query and evidence, the system can now infer why a goal was scored.

www.itbook.store/books/9781617292330

If you think about it, the first reasoning pattern describes reasoning forward in time, predicting future events based on what you know about the current situation, whereas the second reasoning pattern describes reasoning backward in time, inferring past conditions based on current outcomes. When you build probabilistic models, typically the models themselves follow a natural temporal sequence. A player takes the corner kick, then the wind operates on the ball as it's coming in, then the center forward leaps up to try to head the ball, and then the goalie tries to make a save. But the reasoning can go both forward and backward. This is a key feature of probabilistic reasoning, which I'll repeat throughout the book: the direction of reasoning doesn't necessarily follow the direction of the model.

- *Learn from past events to better predict future events.* Now fast-forward another 10 minutes. The same team has won another corner kick. Everything is similar to before in this new situation—tall center forward, inexperienced goalie—but now the wind has died down. Using probabilistic reasoning, you can use what happened in the previous kick to help you predict what will happen on the next kick. Figure 1.5 shows how to do this. The evidence includes all evidence from last time (making a note that it was from last time), as well as the new information about the current situation. In answering the query about whether a goal

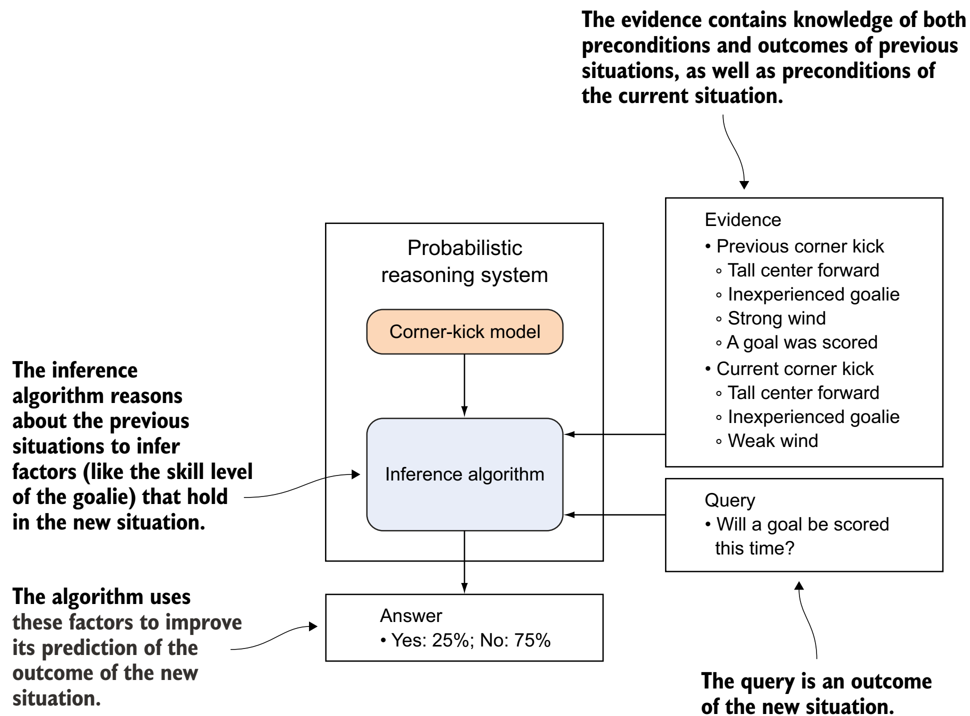

Figure 1.5 By taking into account evidence from the outcome of the last corner kick, the probabilistic reasoning system can produce a better prediction of the next corner kick.

www.itbook.store/books/9781617292330

will be scored this time, the inference algorithm first infers properties of the situation that led to a goal being scored the first time, such as the skill levels of the center forward and goalie. It then uses these updated properties to make a prediction about the new situation.

All of these types of queries can help you make decisions, on many levels:

- You can decide whether to substitute a defender for an attacker based on the probability that a goal will be scored with or without the extra defender.
- You can decide how much to offer the goalie in her next contract negotiation based on your assessment of her skill.
- You can decide whether to use the same goalie in the next game by using what you've learned about the goalie to help predict the outcome of the next game.

LEARNING A BETTER MODEL

The preceding three reasoning patterns provide ways to reason about specific situations, given evidence. Another thing you can do with a probabilistic reasoning system is learn from the past to improve your general knowledge. In the third reasoning pattern, you saw how to learn from a particular past experience to better predict a specific future situation. Another way to learn from the past is to improve the model itself. Especially if you have a lot of past experiences to draw on, such as a lot of corner kicks, you might want to learn a new model representing your general knowledge of what typically happens in a corner kick. As figure 1.6 shows, this is achieved by a learning algorithm. Somewhat different from an inference algorithm, the goal of a learning algorithm is to produce a new model, not to answer queries. The learning algorithm begins with the original model and updates it based on the experience to produce the new model. The new model can then be used to answer queries in the future. Presumably, the answers produced when using the new model will be better informed than when using the original model.

Probabilistic reasoning systems and accurate predictions

Like any machine learning system, a probabilistic reasoning system will be more accurate the more data you give it. The quality of the predictions depends on two things: the degree to which the original model accurately reflects real-world situations, and the amount of data you provide. In general, the more data you provide, the less important the original model is. The reason for this is that the new model is a balance between the original model and the information contained in the data. If you have very little data, the original model dominates, so it had better be accurate. If you have lots of data, the data will dominate and the new model will tend to forget the original model, which doesn't matter as much. For example, if you're learning from an entire soccer season, you should be able to learn the factors that contribute to a corner kick quite accurately. If you have only one game, you'll need to start out with a good idea of the factors to be able to make accurate predictions about that game. Probabilistic reasoning systems will make good use of the given model and available data to make as accurate a prediction as possible.

www.itbook.store/books/9781617292330

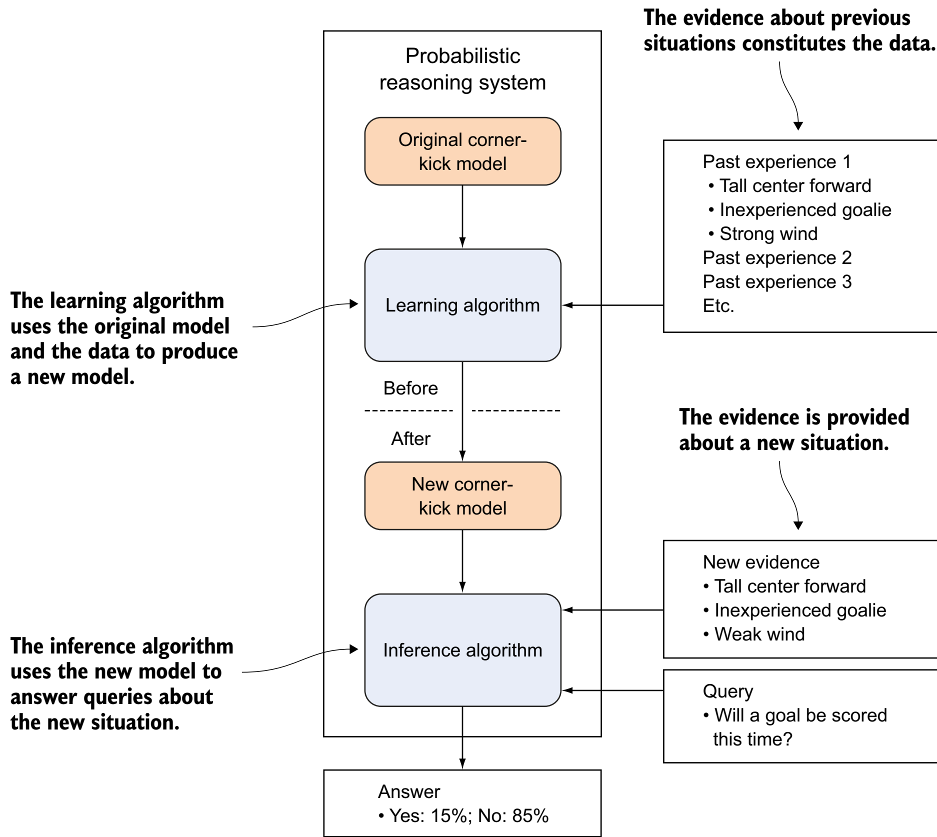

Figure 1.6 You can use a learning algorithm to learn a new model based on a set of experiences. This new model can then be used for future inferences.

Now you know what probabilistic reasoning is. What then, is probabilistic programming?

1.1.4 Probabilistic programming systems: probabilistic reasoning systems expressed in a programming language

Every probabilistic reasoning system uses a *representation language* to express its probabilistic models. There are a lot of representation languages out there. You may have heard of some of them, such as Bayesian networks (also known as belief networks) and hidden Markov models. The representation language controls what models can be handled by the system and what they look like. The set of models that can be represented by a language is called the *expressive power* of the language. For practical applications, you'd like to have as large an expressive power as possible.

A *probabilistic programming* system is, very simply, a probabilistic reasoning system in which the representation language is a programming language. When I say *programming language*, I mean that it has all the features you typically expect in a programming

www.itbook.store/books/9781617292330

language, such as variables, a rich variety of data types, control flow, functions, and so on. As you'll come to see, probabilistic programming languages can express an extremely wide variety of probabilistic models and go far beyond most traditional probabilistic reasoning frameworks. Probabilistic programming languages have tremendous expressive power.

Figure 1.7 illustrates the relationship between probabilistic programming systems and probabilistic reasoning systems in general. The figure can be compared with figure 1.3 to highlight the differences between the two systems. The main change is that models are expressed as programs in a programming language rather than as a mathematical construct like a Bayesian network. As a result of this change, evidence, queries, and answers all apply to variables in the program. Evidence might specify particular values for program variables, queries ask for the values of program variables, and answers are probabilities of different values of the query variables. In addition, a probabilistic programming system typically comes with a suite of inference algorithms. These algorithms apply to programs written in the language.

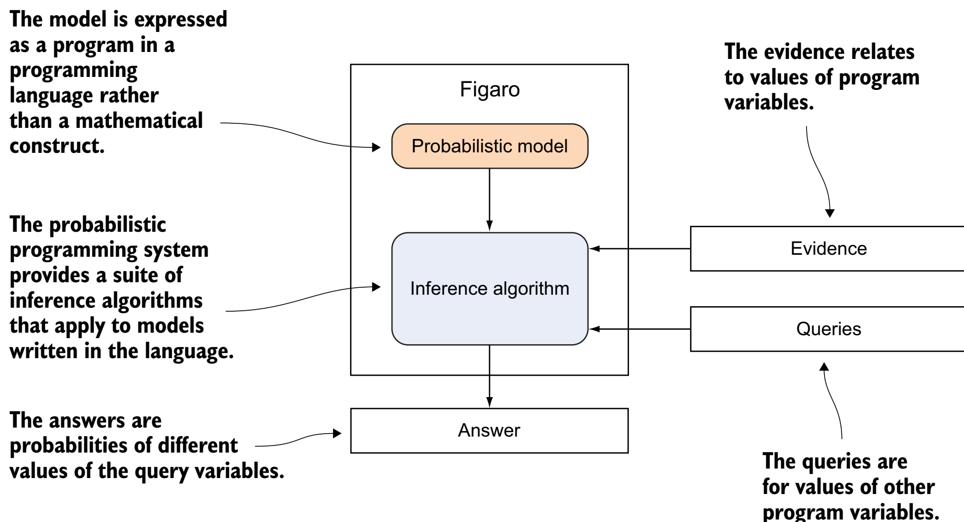

Figure 1.7 A probabilistic programming system is a probabilistic reasoning system that uses a programming language to represent probabilistic models.

Although many kinds of probabilistic programming systems exist (see appendix B for a survey), this book focuses on functional, Turing-complete systems. *Functional* means that they're based on functional programming, but don't let that scare you—you don't need to know concepts such as lambda functions to use functional probabilistic programming systems. All this means is that functional programming provides the theoretical foundation behind these languages that lets them represent probabilistic

www.itbook.store/books/9781617292330

models. Meanwhile, *Turing-complete* is jargon for a programming language that can encode any computation that can be done on a digital computer. If something can be done on a digital computer, it can be done with any Turing-complete language. Most of the programming languages you're familiar with, such as C, Java, and Python, are Turing-complete. Because probabilistic programming languages are built on Turing-complete programming languages, they're extremely flexible in the types of models that can be built.

KEY DEFINITIONS

Representation language—A language for encoding your knowledge about a domain in a model

Expressive power—The ability of a representation language to encode various kinds of knowledge in its models

Turing-complete—A language that can express any computation that can be performed on a digital computer

Probabilistic programming language—A probabilistic representation language that uses a Turing-complete programming language to represent knowledge

Appendix B surveys some probabilistic programming systems besides Figaro, the system used in this book. Most of these systems use Turing-complete languages. Some, including BUGS and Dimple, don't, but they're nevertheless useful for their intended applications. This book focuses on the capabilities of Turing-complete probabilistic programming languages.

Representing probabilistic models as programs

But how can a programming language be a probabilistic modeling language? How can you represent probabilistic models as programs? I'll hint at the answer to this question here but save a deeper discussion for later in the book, when you have a better idea of what a probabilistic program looks like.

A core idea in programming languages is *execution*. You execute a program to generate output. A probabilistic program is similar, except that instead of a single execution path, it can have many execution paths, each generating a different output. The determination of which execution path is followed is specified by random choices throughout the program. Each random choice has a number of possible outcomes, and the program encodes the probability of each outcome. Therefore, a probabilistic program can be thought of as a program you randomly execute to generate an output.

Figure 1.8 illustrates this concept. In the figure, a probabilistic programming system contains a corner-kick program. This program describes the random process of generating the outcome of a corner kick. The program takes some inputs; in our example, these are the height of the center forward, the experience of the goalie, and the strength of the wind. Given the inputs, the program is randomly executed to generate outputs. Each random execution results in a particular output being generated.

www.itbook.store/books/9781617292330

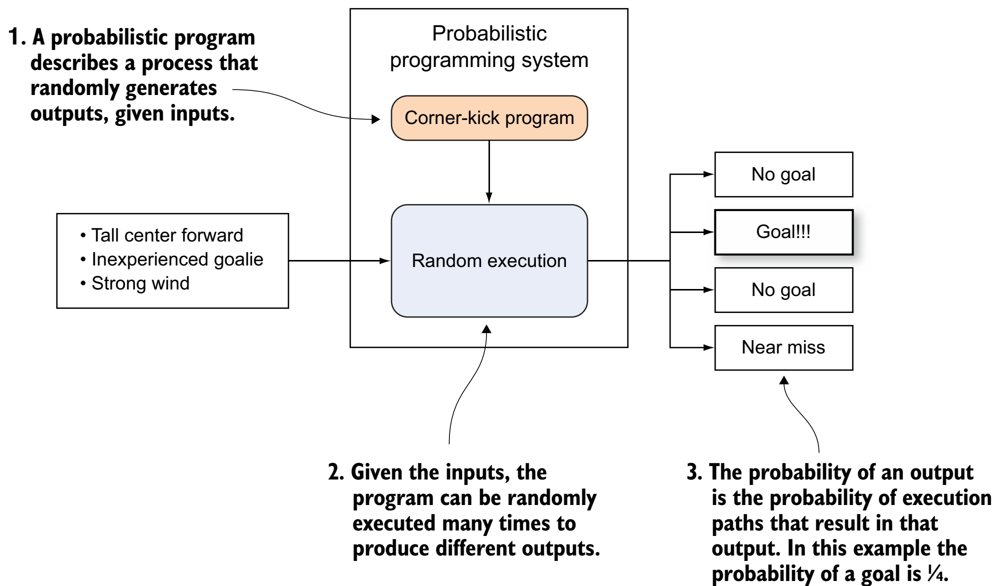

Figure 1.8 A probabilistic program defines a process of randomly generating outputs, given inputs.

Because every random choice has multiple possible outcomes, many possible execution paths exist, resulting in different outputs. Any given output, such as a goal, can be generated by multiple execution paths.

Let's see how this program defines a probabilistic model. Any particular execution path results from a sequence of random choices having specific outcomes. Each random choice has a probability of occurring. If you multiply all these probabilities together, you get the probability of the execution path. So the program defines the probability of every execution path. If you imagine running the program many times, the fraction of times any given execution path will be generated is equal to its probability. The probability of an output is the fraction of times the program is run that result in that output. In figure 1.8, a goal is generated by 1/4 of the runs, so the probability of a goal is 1/4.

> **NOTE** You might be wondering why the block in figure 1.8 is labeled Random Execution rather than Inference Algorithm, as it has been in other figures. Figure 1.8 shows what a probabilistic program means, as defining a random execution process, rather than how you use a probabilistic programming system, which is by using an inference algorithm to answer queries, given evidence. So although the structure of the figures is similar, they convey different concepts. As a matter of fact, random execution forms the basis for some inference algorithms as well, but many algorithms aren't based on simple random execution.

www.itbook.store/books/9781617292330

MAKING DECISIONS WITH PROBABILISTIC PROGRAMMING

It's easy to see how you can use a probabilistic program to predict the future. Just execute the program randomly many times, using what you know about the present as inputs, and observe how many times each output is produced. In the corner-kick example of figure 1.8, you executed the program many times, given the inputs of tall center forward, inexperienced goalie, and strong wind. Because 1/4 of those runs resulted in a goal, you can say that the probability of a goal, given these inputs, is 25%.

The magic of probabilistic programming, however, is that it can also be used for all the kinds of probabilistic reasoning described in section 1.3.1. It can be used not only to predict the future, but also to infer facts that led to particular outcomes; you can "unwind" the program to discover the root causes of the outcomes. You can also apply a program in one situation, learn from the outcome, and then use what you've learned to make better predictions in the future. You can use probabilistic programming to help make all the decisions that can be informed by probabilistic thinking.

How does this work? Probabilistic programming became practical when people realized that inference algorithms that work on simpler representation languages like Bayesian networks can be extended to work on programs. Part 3 of this book presents a variety of inference algorithms that make this possible. Fortunately, probabilistic programming systems come with a range of built-in inference algorithms that apply automatically to your programs. All you have to do is provide your knowledge of your domain in the form of a probabilistic program and specify the evidence, and the system takes care of the inference and learning.

In this book, you'll learn probabilistic reasoning through probabilistic programming. You'll learn, first of all, what a probabilistic model is and how it can be used to draw conclusions. You'll also learn some basic manipulations that are performed to draw those conclusions from a model made up of simple components. You'll learn a variety of modeling techniques and how to implement them by using probabilistic programming. You'll also gain an understanding of how the probabilistic inference algorithms work, so you can design and use your models effectively. By the end of this book, you'll be able to use probabilistic programming confidently to draw useful conclusions that inform your decisions in the face of uncertainty.

1.2 Why probabilistic programming?

Probabilistic reasoning is one of the foundational technologies of machine learning. It's used by companies such as Google, Amazon, and Microsoft to make sense of the data available to them. Probabilistic reasoning has been used for applications as diverse as predicting stock prices, recommending movies, diagnosing computers, and detecting cyber intrusions. Many of these applications use techniques you'll learn in this book.

www.itbook.store/books/9781617292330

From the previous section, two points stand out:

- Probabilistic reasoning can be used to predict the future, infer the past, and learn from the past to better predict the future.
- Probabilistic programming is probabilistic reasoning using a Turing-complete programming language for representation.

Put these two together and you have a slogan expressed in the Fact note.

FACT Probabilistic reasoning + Turing-complete = probabilistic programming

The motivation for probabilistic programming is that it takes two concepts that are powerful in their own right and puts them together. The result is an easier and more flexible way to use computers to help make decisions under uncertainty.

1.2.1 Better probabilistic reasoning

Most existing probabilistic representation languages are limited in the richness of the systems they can represent. Some relatively simple languages such as Bayesian networks assume a fixed set of variables and aren't flexible enough to model domains in which the variables themselves can change. More-advanced languages with more flexibility have been developed in recent years. Some (for example, BUGS) also provide programming-language features including iteration and arrays, without being Turing-complete. The success of languages such as BUGS shows a need for richer, more structured representations. But moving to full-fledged, Turing-complete languages opens a world of possibilities for probabilistic reasoning. It's now possible to model long-running processes with many interacting entities and events.

Let's consider the soccer example again, but now imagine that you're in the business of sports analytics and want to recommend personnel decisions for a team. You could use accumulated statistics to make your decisions, but statistics don't capture the context in which they were accumulated. You can achieve a more fine-grained, context-aware analysis by modeling the soccer season in detail. This requires modeling many dependent events and interacting players and teams. It would be hard to imagine building this model without the data structures and control flow provided by a full programming language.

Now let's think about the product launch example again, and look at making decisions for your business in an integrated way. The product launch isn't an isolated incident, but follows phases of market analysis, research, and development, all of which have uncertainty in their outcome. The results of the product launch depend on all these phases, as well as an analysis of what else is available in the market. A full analysis will also look at how your competitors will respond to your product, as well as any new products they might bring. This problem is hard, because you have to conjecture about competing products. You may even have competitors you don't know about yet. In this example, products are data structures produced by complex processes. Again, having a full programming language available to create the model would be helpful.

www.itbook.store/books/9781617292330

One of the nice things about probabilistic programming, however, is that if you want to use a simpler probabilistic reasoning framework, you can. Probabilistic programming systems can represent a wide range of existing frameworks, as well as systems that can't be represented in other frameworks. This book teaches many of these frameworks using probabilistic programming. So in learning probabilistic programming, you'll also master many of the probabilistic reasoning frameworks commonly used today.

1.2.2 Better simulation languages

Turing-complete probabilistic modeling languages already exist. They're commonly called *simulation languages*. We know that it's possible to build simulations of complex processes such as soccer seasons by using programming languages. In this context, I use the term *simulation language* to describe a language that can represent the execution of complex processes with randomness. Just like probabilistic programs, these simulations are randomly executed to produce different outputs. Simulations are as widely used as probabilistic reasoning, in applications from military planning to component design to public health to sports predictions. Indeed, the widespread use of sophisticated simulations demonstrates the need for rich probabilistic modeling languages.

But a probabilistic program is much more than a simulation. With a simulation, you can do only one of the things you can do with a probabilistic program: predict the future. You can't use it to infer the root causes of the outcomes that are observed. And, although you can update a simulation with known current information as you go along, it's hard to include unknown information that must be inferred. As a result, the ability to learn from past experience to improve future predictions and analyses is limited. You can't use simulations for machine learning.

A probabilistic program is like a simulation that you can analyze, not just run. The key insight in developing probabilistic programming is that many of the inference algorithms that can be used for simpler modeling frameworks can also be used on simulations. Hence, you have the ability to create a probabilistic model by writing a simulation and performing inferences on it.

One final word. Probabilistic reasoning systems have been around for a while, with software such as Hugin, Netica, and BayesiaLab providing Bayesian network systems. But the more expressive representation languages of probabilistic programming are so new that we're just beginning to discover their powerful applications. I can't honestly tell you that probabilistic programming has already been used in a large number of fielded applications. But some significant applications exist. Microsoft has been able to determine the true skill level of players of online games by using probabilistic programming. Stuart Russell at the University of California at Berkeley has written a program to help enforce the United Nations Comprehensive Nuclear-Test-Ban Treaty by identifying seismic events that could indicate a nuclear explosion. Josh Tenenbaum at the Massachusetts Institute of Technology (MIT) and Noah Goodman at Stanford University have created probabilistic programs to model human cognition with considerable explanatory success. At Charles River Analytics, we've used probabilistic

www.itbook.store/books/9781617292330

programming to infer components of malware instances and determine their evolution. But I believe these applications are only scratching the surface. Probabilistic programming systems are reaching the point where they can be used by larger numbers of people to make decisions in their own domains. By reading this book, you have a chance to get in on this new technology on the ground floor.

1.3 Introducing Figaro: a probabilistic programming language

In this book, you'll use a probabilistic programming system called Figaro. (I named Figaro after the character from Mozart's opera "The Marriage of Figaro." I love Mozart and played Dr. Bartolo in a Boston production of the opera.) The main goal of the book is to teach the principles of probabilistic programming, and the techniques you learn in this book should carry over to other probabilistic programming systems. Some of the available systems are listed with a brief description in appendix B. A secondary goal, however, is to give you hands-on experience with creating practical probabilistic programs, and provide you with tools you can use right away. For that reason, a lot of the examples are made concrete in Figaro code.

Figaro, which is open source and maintained on GitHub, has been under development since 2009. It's implemented as a Scala library. Figure 1.9 shows how Figaro uses

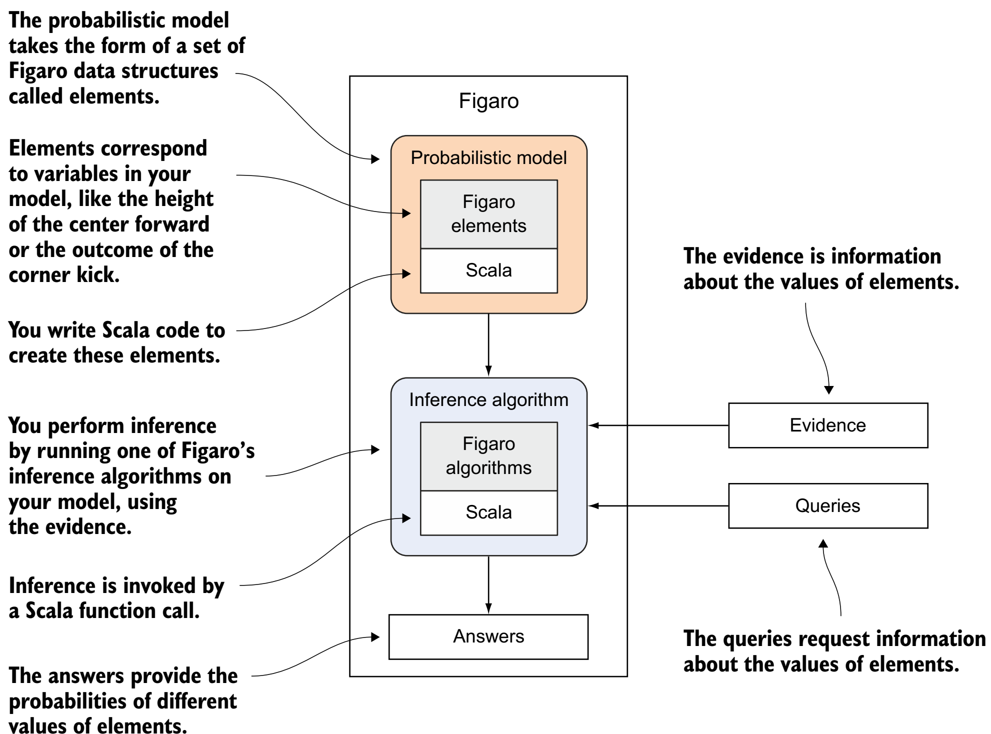

Figure 1.9 How Figaro uses Scala to provide a probabilistic programming system

www.itbook.store/books/9781617292330

Scala to implement a probabilistic programming system. The figure elaborates on figure 1.7, which describes the main components of a probabilistic programming system. Let's start with the probabilistic model. In Figaro, the model consists of any number of data structures known as *elements*. Each element represents a variable that can take on any number of values in your situation. These data structures are implemented in Scala, and you write a Scala program to create a model using these data structures. You can supply evidence by providing information about the values of elements, and you can specify which elements you want to know about in your query. For the inference algorithm, you choose one of Figaro's built-in inference algorithms and apply it to your model, to answer your query, given the evidence. The inference algorithms are implemented in Scala, and invoking an inference algorithm is simply a Scala function call. The results of inference are probabilities of various values of your query elements.

Figaro's embedding in Scala provides some major advantages. Some of these come from embedding in a general-purpose host language, compared to a standalone probabilistic language. Others come specifically because of the favorable properties of Scala. Here's why it's good to embed a probabilistic programming language in a general-purpose host language:

- The evidence can be derived using a program in the host language. For example, you might have a program that reads a data file, processes the values in some way, and provides that as evidence for the Figaro model. It's much harder to do this in a standalone language.
- Similarly, you can use the answers provided by Figaro in a program. For example, if you have a program used by a soccer manager, the program can take the probability of a goal being scored to recommend to the manager what to do.
- You can embed general-purpose code inside the probabilistic program. For example, suppose you have a physics model that simulates the trajectory of a headed ball through the air. You can incorporate this model inside a Figaro element.
- You can use general programming techniques to build your Figaro model. For example, you might have a map containing Figaro elements corresponding to all the players in your squad and choose the appropriate elements for a situation based on the players involved in that situation.

Here are some reasons that Scala is a particularly good choice of language for embedding a probabilistic programming system in:

- Because Scala is a functional programming language, Figaro gets to benefit from functional programming too. Functional programming has been instrumental in probabilistic programming, and many models can be written naturally in a functional manner, as I'll show in part 2.
- Scala is object-oriented; one of the beauties of Scala is that it is both functional and object-oriented. Figaro is also object-oriented. As I'll describe in part 2, object-orientation is a useful way to express several design patterns in probabilistic programming.

www.itbook.store/books/9781617292330

Finally, some of Figaro's advantages go beyond its embedding in Scala. These include the following:

- Figaro can represent an extremely wide range of probabilistic models. The values of Figaro elements can be any type, including Booleans, integers, doubles, arrays, trees, graphs, and so on. The relationships between these elements can be defined by any function.
- Figaro provides a rich framework for specifying evidence by using its conditions and constraints.
- Figaro features a good variety of inference algorithms.
- Figaro can represent and reason about dynamic models of situations that vary over time.
- Figaro can include explicit decisions in its models and supports inferring optimal decisions.

> **Using Scala**
>
> Because Figaro is a Scala library, you'll need a working knowledge of Scala to use Figaro. This is a book on probabilistic programming, so I don't teach Scala in this book. Many great resources for learning Scala are available, such as Twitter's Scala School (http://twitter.github.io/scala_school). But in case you aren't yet confident with Scala, I explain the Scala features used in the code as I go along. You'll be able to follow the book even if you don't know Scala yet.
>
> You don't need to be a Scala wizard to benefit from probabilistic programming and Figaro, and I avoid using some of the more advanced and obscure features of Scala in this book. On the other hand, improving your Scala skills can help you become a better Figaro programmer. You might even find that your Scala skills improve as a result of reading this book.

For several reasons, Figaro is a favorable language for learning probabilistic programming:

- Being implemented as a Scala library, Figaro can be used in Java and Scala programs, making it easy to integrate into applications.
- Also related to being implemented as a library, rather than its own separate language, Figaro provides the full functionality of the host programming language to build your models. Scala is an advanced, modern programming language with many useful features for organizing programs, and you automatically benefit from those features when using Figaro.
- Figaro is fully featured in terms of the range of algorithms it provides.

This book emphasizes practical techniques and practical examples. Wherever possible, I explain the general modeling principle, as well as describe how to implement it in Figaro. This will stand you in good stead no matter what probabilistic programming system you end up using. Not all systems will be capable of easily implementing all the

www.itbook.store/books/9781617292330

techniques in this book. For example, few object-oriented probabilistic programming systems currently exist. But with the right foundation, you can find a way to express what you need in your chosen language.

1.3.1 Figaro vs. Java: building a simple probabilistic programming system

To illustrate the benefits of probabilistic programming and Figaro, I'll show a simple probabilistic application written two ways. First, I'll show you how to write it in Java, with which you might be familiar. Then, I'll show you what it looks like in Scala using Figaro. Although Scala has some advantages over Java, that's not the main difference I'll point out here. The key idea is that *Figaro provides capabilities for representing probabilistic models and performing inference with them that aren't available without probabilistic programming.*

Our little application will also serve as a Hello World example for Figaro. Imagine someone who gets up in the morning, checks if the weather is sunny, and utters a greeting that depends on the weather. This happens two days in a row. Also, the weather on the second day is dependent on the first day: the second day is more likely to be sunny if the first day is sunny. These English language statements can be quantified numerically by the numbers in table 1.1.

Table 1.1 Quantifying the probabilities in the Hello World example

Today's weather		
Sunny	0.2	
Not sunny	0.8	
Today's greeting		
If today's weather is sunny	"Hello, world!"	0.6
	"Howdy, universe!"	0.4
If today's weather isn't sunny	"Hello, world!"	0.2
	"Oh no, not again"	0.8
Tomorrow's weather		
If today's weather is sunny	Sunny	0.8
	Not sunny	0.2
If today's weather isn't sunny	Sunny	0.05
	Not sunny	0.95
Tomorrow's greeting		
If tomorrow's weather is sunny	"Hello, world!"	0.6
	"Howdy, universe!"	0.4
If tomorrow's weather isn't sunny	"Hello, world!"	0.2
	"Oh no, not again"	0.8

www.itbook.store/books/9781617292330

The forthcoming chapters explain exactly how to interpret these numbers. For now, it's enough to have an intuitive idea that today's weather will be sunny with probability 0.2, meaning that it's 20% likely that the weather will be sunny today. Likewise, if tomorrow's weather is sunny, tomorrow's greeting will be "Hello, world!" with probability 0.6, meaning that it's 60% likely that the greeting will be "Hello, world!" and it's 40% likely that the greeting will be "Howdy, universe!"

Let's set for ourselves three reasoning tasks to perform with this model. You saw in section 1.1.3 that the three types of reasoning you can do with a probabilistic model are to *predict* the future, *infer* past events that led to your observations, and *learn* from past events to better predict the future. You'll do all of these with our simple model. The specific tasks are as follows:

1. Predict the greeting today.
2. Given an observation that today's greeting is "Hello, world!" infer whether today is sunny.
3. Learn from an observation that today's greeting is "Hello, world!" to predict tomorrow's greeting.

Here's how to do these tasks in Java.

Listing 1.1 Hello World in Java

```
class HelloWorldJava {
  static String greeting1 = "Hello, world!";
  static String greeting2 = "Howdy, universe!";
  static String greeting3 = "Oh no, not again";

  static Double pSunnyToday = 0.2;
  static Double pNotSunnyToday = 0.8;
  static Double pSunnyTomorrowIfSunnyToday = 0.8;
  static Double pNotSunnyTomorrowIfSunnyToday = 0.2;
  static Double pSunnyTomorrowIfNotSunnyToday = 0.05;
  static Double pNotSunnyTomorrowIfNotSunnyToday = 0.95;
  static Double pGreeting1TodayIfSunnyToday = 0.6;
  static Double pGreeting2TodayIfSunnyToday = 0.4;
  static Double pGreeting1TodayIfNotSunnyToday = 0.2;
  static Double pGreeting3IfNotSunnyToday = 0.8;
  static Double pGreeting1TomorrowIfSunnyTomorrow = 0.5;
  static Double pGreeting2TomorrowIfSunnyTomorrow = 0.5;
  static Double pGreeting1TomorrowIfNotSunnyTomorrow = 0.1;
  static Double pGreeting3TomorrowIfNotSunnyTomorrow = 0.95;

  static void predict() {
     Double pGreeting1Today =
            pSunnyToday * pGreeting1TodayIfSunnyToday +
            pNotSunnyToday * pGreeting1TodayIfNotSunnyToday;
     System.out.println("Today's greeting is " + greeting1 +
        "with probability " + pGreeting1Today + ".");
  }
```

Define the greetings

Specify the numerical parameters of the model

Predict today's greeting using the rules of probabilistic inference

www.itbook.store/books/9781617292330

```
  static void infer() {
     Double pSunnyTodayAndGreeting1Today =
            pSunnyToday * pGreeting1TodayIfSunnyToday;
     Double pNotSunnyTodayAndGreeting1Today =
            pNotSunnyToday * pGreeting1TodayIfNotSunnyToday;
     Double pSunnyTodayGivenGreeting1Today =
            pSunnyTodayAndGreeting1Today /
            (pSunnyTodayAndGreeting1Today +
             pNotSunnyTodayAndGreeting1Today);
     System.out.println("If today's greeting is " + greeting1 +
         ", today's weather is sunny with probability " +
         pSunnyTodayGivenGreeting1Today + ".");
  }

  static void learnAndPredict() {
     Double pSunnyTodayAndGreeting1Today =
            pSunnyToday * pGreeting1TodayIfSunnyToday;
     Double pNotSunnyTodayAndGreeting1Today =
            pNotSunnyToday * pGreeting1TodayIfNotSunnyToday;
     Double pSunnyTodayGivenGreeting1Today =
            pSunnyTodayAndGreeting1Today /
            (pSunnyTodayAndGreeting1Today +
                 pNotSunnyTodayAndGreeting1Today);
     Double pNotSunnyTodayGivenGreeting1Today =
            1 - pSunnyTodayGivenGreeting1Today;
     Double pSunnyTomorrowGivenGreeting1Today =
            pSunnyTodayGivenGreeting1Today *
                 pSunnyTomorrowIfSunnyToday +
            pNotSunnyTodayGivenGreeting1Today *
                 pSunnyTomorrowIfNotSunnyToday;
     Double pNotSunnyTomorrowGivenGreeting1Today =
            1 - pSunnyTomorrowGivenGreeting1Today;
     Double pGreeting1TomorrowGivenGreeting1Today =
            pSunnyTomorrowGivenGreeting1Today *
                 pGreeting1TomorrowIfSunnyTomorrow +
            pNotSunnyTomorrowGivenGreeting1Today *
                 pGreeting1TomorrowIfNotSunnyTomorrow;
     System.out.println("If today's greeting is " + greeting1 +
         ", tomorrow's greeting will be " + greeting1 +
         " with probability " +
         pGreeting1TomorrowGivenGreeting1Today);
  }

  public static void main(String[] args) {
    predict();
    infer();
    learnAndPredict();
  }
}
```

Infer today's weather given the observation that today's greeting is "Hello, world!" using the rules of probabilistic inference

Learn from observing that today's greeting is "Hello, world!" to predict tomorrow's greeting using the rules of probabilistic inference

Main method that performs all the tasks

I won't describe how the calculations are performed using the rules of inference here. The code uses three rules of inference: the chain rule, the total probability rule, and

www.itbook.store/books/9781617292330

Bayes' rule. All these rules are explained in detail in chapter 9. For now, let's point out two major problems with this code:

- *There's no way to define a structure to the model.*
  The definition of the model is contained in a list of variable names with double values. When I described the model at the beginning of the section and showed the numbers in table 1.1, the model had a lot of structure and was relatively understandable, if only at an intuitive level. This list of variable definitions has no structure. The meaning of the variables is buried inside the variable names, which is always a bad idea. As a result, it's hard to write down the model in this way, and it's quite an error-prone process. It's also hard to read and understand the code afterward and maintain it. If you need to modify the model (for example, the greeting also depends on whether you slept well), you'll probably need to rewrite large portions of the model.
- *Encoding the rules of inference yourself is difficult and error-prone.*
  The second major problem is with the code that uses the rules of probabilistic inference to answer the queries. You have to have intimate knowledge of the rules of inference to write this code. Even if you have this knowledge, writing this code correctly is difficult. Testing whether you have the right answer is also difficult. And this is an extremely simple example. For a complex application, it would be impractical to create reasoning code in this way.

Now let's look at the Scala/Figaro code.

Listing 1.2 Hello World in Figaro

```
import com.cra.figaro.language.{Flip, Select}
import com.cra.figaro.library.compound.If
import com.cra.figaro.algorithm.factored.VariableElimination

object HelloWorld {
  val sunnyToday = Flip(0.2)
  val greetingToday = If(sunnyToday,
       Select(0.6 -> "Hello, world!", 0.4 -> "Howdy, universe!"),
       Select(0.2 -> "Hello, world!", 0.8 -> "Oh no, not again"))
  val sunnyTomorrow = If(sunnyToday, Flip(0.8), Flip(0.05))
  val greetingTomorrow = If(sunnyTomorrow,
       Select(0.6 -> "Hello, world!", 0.4 -> "Howdy, universe!"),
       Select(0.2 -> "Hello, world!", 0.8 -> "Oh no, not again"))

  def predict() {
    val result = VariableElimination.probability(greetingToday,
                   "Hello, world!")
    println("Today's greeting is \"Hello, world!\" " +
            "with probability " + result + ".")
  }
```

Import Figaro constructs

Define the model

Predict today's greeting using an inference algorithm

www.itbook.store/books/9781617292330

```
  def infer() {
    greetingToday.observe("Hello, world!")
    val result = VariableElimination.probability(sunnyToday, true)
    println("If today's greeting is \"Hello, world!\", today's " +
            "weather is sunny with probability " + result + ".")
  }

  def learnAndPredict() {
    greetingToday.observe("Hello, world!")
    val result = VariableElimination.probability(greetingTomorrow,
                   "Hello, world!")
    println("If today's greeting is \"Hello, world!\", " +
            "tomorrow's greeting will be \"Hello, world!\" " +
            "with probability " + result + ".")
  }

  def main(args: Array[String]) {
    predict()
    infer()
    learnAndPredict()
  }
}
```

Use an inference algorithm to infer today's weather, given the observation that today's greeting is "Hello, world!"

Learn from observing that today's greeting is "Hello, world!" to predict tomorrow's greeting using an inference algorithm

Main method that performs all the tasks

I'll wait until the next chapter to explain this code in detail. For now, I want to point out that it solves the two problems with the Java code. First, the model definition describes exactly the structure of the model, in correspondence with table 1.1. You define four variables: `sunnyToday`, `greetingToday`, `sunnyTomorrow`, and `greetingTomorrow`. Each has a definition that corresponds to table 1.1. For example, here's the definition of `greetingToday`:

```
val greetingToday = If(sunnyToday,
      Select(0.6 -> "Hello, world!", 0.4 -> "Howdy, universe!"),
      Select(0.2 -> "Hello, world!", 0.8 -> "Oh no, not again"))
```

This says that if today is sunny, today's greeting is "Hello, world!" with probability 0.6 and "Howdy, universe!" with probability 0.4. If today isn't sunny, today's greeting is "Hello, world!" with probability 0.2 and "Oh no, not again" with probability 0.8. This is exactly what table 1.1 says for today's greeting. Because the code explicitly describes the model, the codes is much easier to construct, read, and maintain. And if you need to change the model (for example, by adding a `sleepQuality` variable), this can be done in a modular way.

Now let's look at the code to perform the reasoning tasks. It doesn't contain any calculations. Instead, it instantiates an algorithm (in this case, the variable elimination algorithm, one of several algorithms available in Figaro) and queries the algorithm to get the probability you want. Now, as described in part 3, this algorithm is based on the same rules of probabilistic inference that the Java program uses. All the hard work of organizing and applying the rules of inference is taken care of by the algorithm. Even for a large and complex model, you can run the algorithm, and all the inference is taken care of.

www.itbook.store/books/9781617292330

1.4 Summary

- Making judgment calls requires knowledge + logic.
- In probabilistic reasoning, a probabilistic model expresses the knowledge, and an inference algorithm encodes the logic.
- Probabilistic reasoning can be used to predict future events, infer causes of past events, and learn from past events to improve predictions.
- Probabilistic programming is probabilistic reasoning, where the probabilistic model is expressed using a programming language.
- A probabilistic programming system uses a Turing-complete programming language to represent models and provides inference algorithms to use the models.
- Figaro is a probabilistic programming system implemented in Scala that provides functional and object-oriented programming styles.

1.5 Exercises

Solutions to selected exercises are available online at www.manning.com/books/practical-probabilistic-programming.

1. Imagine that you want to use a probabilistic reasoning system to reason about the outcome of poker hands.
   a. What kind of general knowledge could you encode in your model?
   b. Describe how you might use the system to predict the future. What's the evidence? What's the query?
   c. Describe how you might use the system to infer past causes of current observations. What's the evidence? What's the query?
   d. Describe how the inferred past causes can help you with your future predictions.
2. In the Hello World example, change the probability that today's weather is sunny according to the following table. How do the outputs of the program change? Why do you think they change this way?

Today's weather	
Sunny	0.9
Not sunny	0.1

3. Modify the Hello World example to add a new greeting: "Hi, galaxy!" Give this greeting some probability when the weather is sunny, making sure to reduce the probability of the other greetings so the total probability is 1. Also, modify the program so that all the queries print the probability of "Hi, galaxy!" instead of "Hello, world!" Try to do this for both the Java and Figaro versions of the Hello World program. Compare the process for the two languages.

www.itbook.store/books/9781617292330

PROGRAMMING/DATA SCIENCE

Practical Probabilistic Programming

Avi Pfeffer

The data you accumulate about your customers, products, and website users can not only help you interpret your past, it can help you predict your future! Probabilistic programming uses code to draw probabilistic inferences from data. By applying specialized algorithms, your programs assign degrees of probability to conclusions. This means you can forecast future events like sales trends, computer system failures, experimental outcomes, and many other critical concerns.

Practical Probabilistic Programming introduces the working programmer to probabilistic programming. In this book, you'll immediately work on practical examples like building a spam filter, diagnosing computer system data problems, and recovering digital images. You'll discover probabilistic inference, where algorithms help make extended predictions about issues like social media usage. Along the way, you'll learn to use functional-style programming for text analysis, object-oriented models to predict social phenomena like the spread of tweets, and open universe models to gauge real-life social media usage. The book also has chapters on how probabilistic models can help in decision making and modeling of dynamic systems.

What's Inside

- Introduction to probabilistic modeling
- Writing probabilistic programs in Figaro
- Building Bayesian networks
- Predicting product lifecycles
- Decision-making algorithms

This book assumes no prior exposure to probabilistic programming. Knowledge of Scala is helpful.

Avi Pfeffer is the principal developer of the Figaro language for probabilistic programming.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit www.manning.com/books/practical-probabilistic-programming

MANNING

$59.99 / Can $68.99 [INCLUDING eBOOK]

"An important step in moving probabilistic programming from research laboratories out into the real world."
—From the Foreword by Stuart Russell, UC Berkeley

"Clear examples and down-to-earth explanations of a difficult and complex topic."
—Mark Elston, Advantest America

"Coherent, practical, and accessible. A fantastic hands-on book on probabilistic programming with Scala."
—Kostas Passadis, IPTO

"Probabilistic programming is complex! Avi makes the subject straightforward and intuitive to learn."
—Earl Bingham, Eyelock

ISBN 13: 978-1-61729-233-0
ISBN 10: 1-61729-233-8

www.itbook.store/books/9781617292330