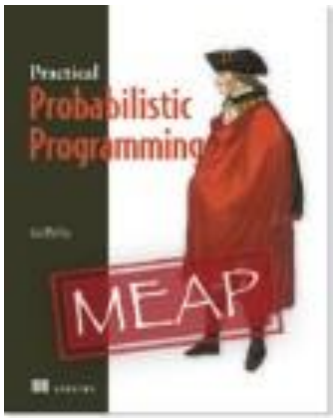
What Probabilistic Programming is and How to Use it

By Avi Pfeffer, Practical Probabilistic Programming



Probabilistic programming is a way to create systems that help us make decisions in the face of uncertainty. 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 explain probabilistic programming, we'll start out looking at decision making under uncertainty and the judgment calls involved. Then we'll see how probabilistic reasoning can help make these decisions. We'll look at three specific kinds of reasoning that probabilistic reasoning systems can do. Then we'll be able to understand probabilistic programming and how it can be used to build probabilistic reasoning systems through the power of programming languages.

How Do We Make Judgment Calls?

In the real world, there are rarely clear yes or no answers to the questions we care about. If we're launching a new product, we want to know if it will sell well. We might think it will be successful, because we believe it is well designed and our market research indicates there is a need for it, but we can't be sure. Maybe our competitor will come out with an even better product, or maybe it has some fatal flaw that will turn off the market, or maybe the economy will take a sudden turn for the worse. If we rely on being 100 percent sure, we will not be able to make the decision of whether or not to launch the product (Figure 1).

Figure 1 - Last year everyone loved my product, what will happen next year?

The language of probability can help make decisions like these. When launching a product, we can use prior experience with similar products to estimate the probability the product will be successful. We can then use this probability to help decide whether to go ahead in launching the product. We can provide the probabilities of different outcomes to make more informed decisions.

Okay, so probabilistic thinking can help us make hard decisions and judgment calls. But how do we do that? Here's the general principle:

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

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

So, we have to have a way of specifying the knowledge, and we have to have logic for getting answers to our questions 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 category of probabilistic reasoning system, which provides the basic means to specify knowledge and provide logic.

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

Figure 2 shows how you would use a probabilistic reasoning system to find the answer. You would encode your knowledge about corner kicks and all the relevant factors in a corner kick model. You would 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 a goal will be scored with probability 20 percent.

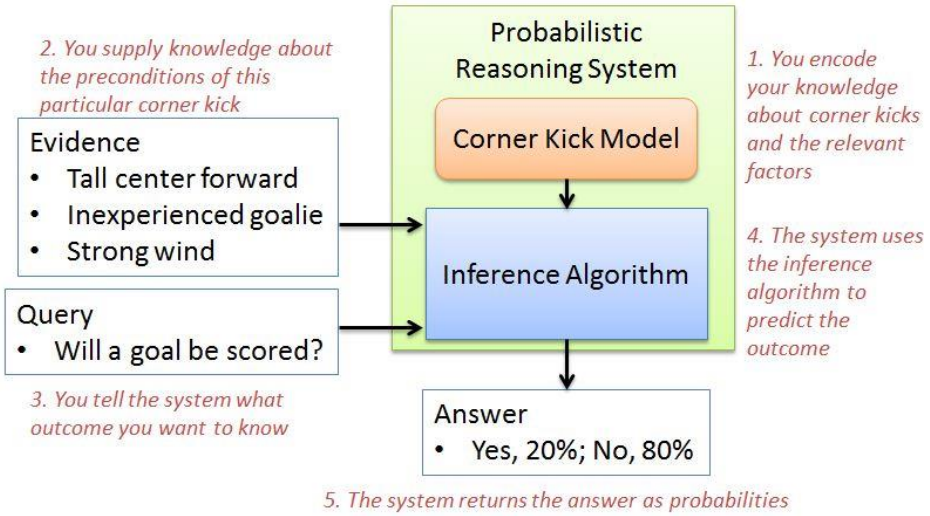


Figure 2: How a probabilistic reasoning system predicts the outcome of a corner kick

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

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 3 summarizes what we've just learned.

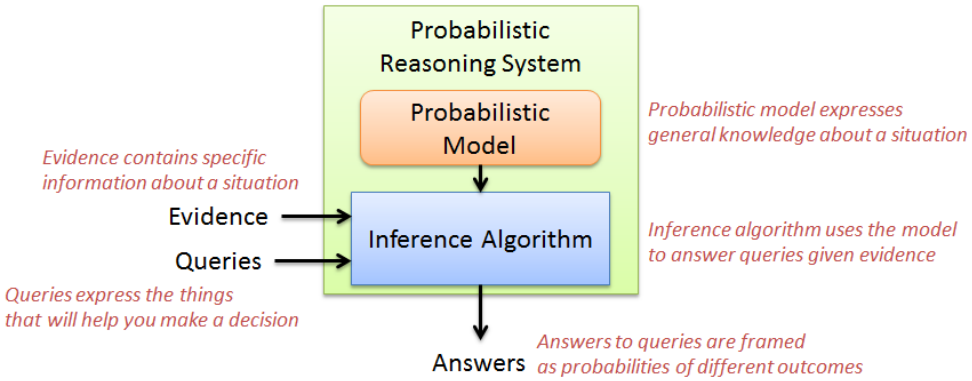


Figure 3: The basic components of a probabilistic reasoning system

So those, in a nutshell, 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 make decisions? The next section describes three kinds of reasoning that can be performed by a probabilistic reasoning system.

Probabilistic Reasoning Systems Can Reason In Three Ways

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

1. **Predict future events.** We've already seen this in Figure 2, where we 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.

2. **Infer the cause of events.** Fast forward ten 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 is poorly skilled? 4 shows how you would use a probabilistic reasoning system to answer these questions. 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 can be used to predict a future result as to infer what caused that result afterwards.) The evidence here is the same as before, together with the fact that a goal was scored. The query is the quality of the goalie, and the answer provides the probability of various qualities.

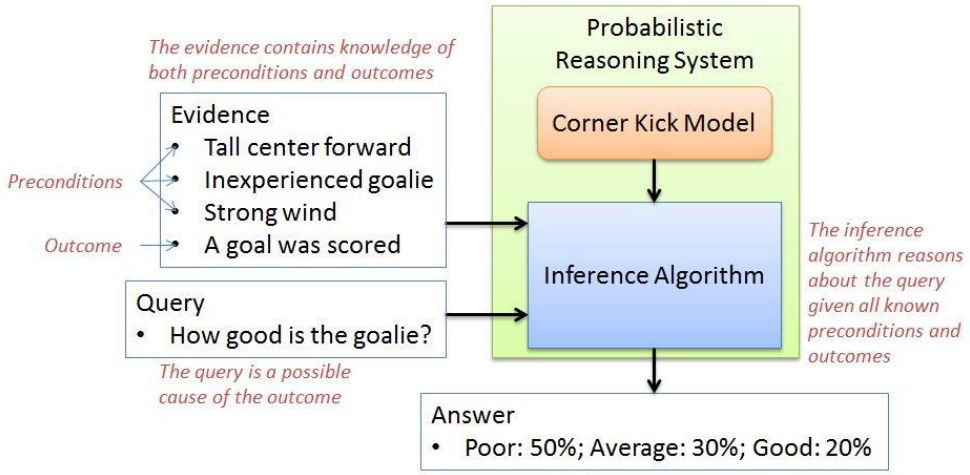


Figure 4: By altering the query and evidence, the system can now infer why a goal was scored

3. **Learn from past events to better predict future events.** Now fast forward another ten 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 5 shows how you can 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 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 quality of the center forward

and goalie. It then uses these updated properties to make a prediction about the new situation.

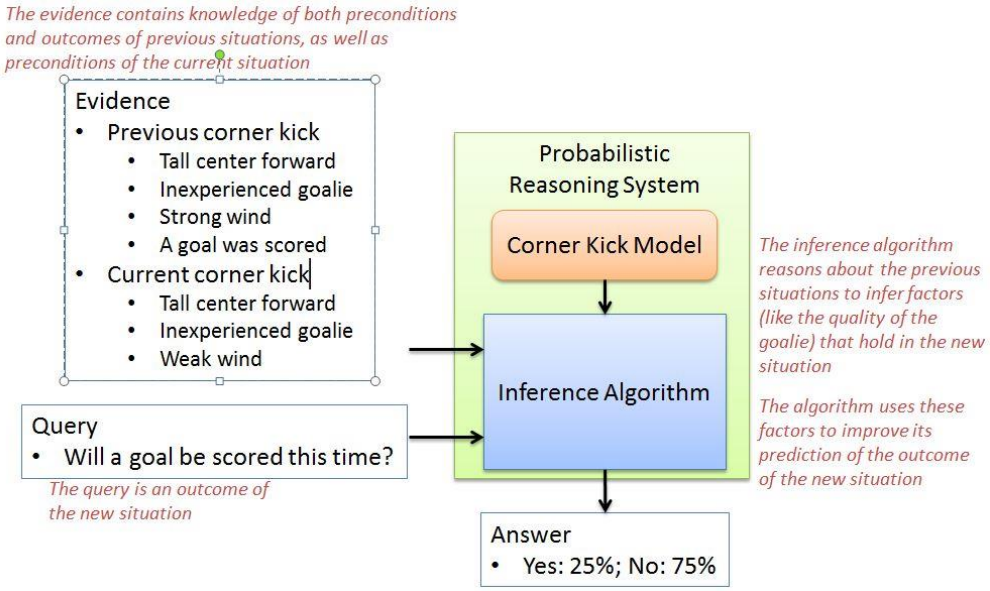


Figure 5: By taking into account evidence from the outcome of the last corner kick, the probabilistic reasoning system can produce a better prediction of the next corner kick

If we think about this last kind of reasoning, we can see that this is a kind of *machine learning*. The system is learning from past events to better predict future events. In our example, we just learned from a single past event, but in general, we might have many past events, like a whole season's worth of soccer games, to learn from.

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 important the original model is. 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 only have one game, you will need to start out with a good idea of the factors to be able to

make accurate predictions about that game. In general, probabilistic reasoning systems will make good use of the given model and available data to make as accurate a prediction as possible.

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

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

So now we know what probabilistic reasoning is. What then, is probabilistic programming?

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, we'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 language, like variables, a rich variety of data types, control flow, functions, and so on. As we'll come to see, probabilistic programming languages are able to express an extremely wide variety of probabilistic models and go far beyond most traditional probabilistic reasoning frameworks. In other words, probabilistic programming languages have very large expressive power.

Figure 6 illustrates the relationship between probabilistic programming systems and probabilistic reasoning systems in general. The figure is based on Figure 3 and the annotations in red are taken exactly from that figure. The annotations in blue show what changes in a probabilistic programming system. 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. So, 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.

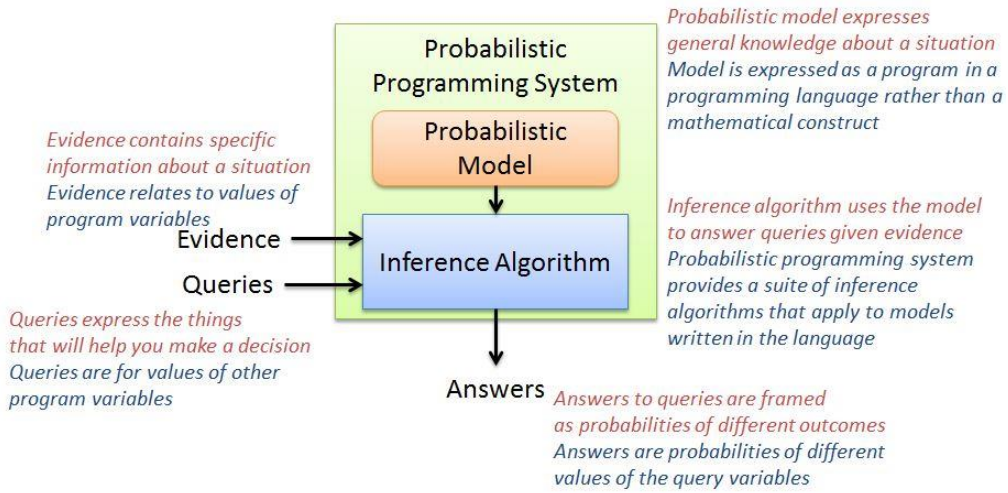


Figure 6: A probabilistic programming system is a probabilistic reasoning system that uses a programming language to represent probabilistic models

Although there are many kinds of probabilistic programming systems, the book on which this article is based, Practical Probabilistic Programming, focuses on *functional*, *Turing-complete* systems. *Functional* means that they are based on functional programming, but don't let that scare you—you don't need to know concepts like 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 models. Meanwhile, Turing-complete is jargon for a programming language that can encode any computation that can be done on a digital computer. In other words, if something can be done on a digital computer, it can be done with any Turing-complete language. Most of the programming languages you are familiar with, such as C, Java, and Python, are Turing-complete. Since probabilistic programming languages are built on Turing-complete programming languages, they are extremely flexible in the types of models that can be built. Click here to check out the book.