Propositional Probability I: Syntax & semantics[1]

Steven Holtzen
s.holtzen@northeastern.edu

September 14, 2023

[1] CS7470 Fall 2023: Foundations of Probabilistic Programming.

"Programming languages are languages, a means of expressing computations in a form comprehensible to both people and machines." Harper [2016]

The goal of these next 3 lectures is to build our our first "probabilistic programming language". I put that in quotes because it is not going to look like a typical programming language like C or Java. We will build our way up to programming languages that look more familiar to you, but today we will encounter a "baby" PPL that is nonetheless surprisingly useful. That baby PPL will be based on *propositional logic*. Along the way, we will see the basics of (discrete) probability that we will need for the first part of the course.

1 *Probability*

"Whereas truth values in logic characterize the formulas under discussion, uncertainty characterizes *invisible* facts." Pearl [1988]

The formal notion of probability used here was introduced by Kolmogorov; see Kolmogorov and Bharucha-Reid [2018]. There have been many introductions and interesting discussions of formal notions of probability over the years; see Jaynes [2003] for an interesting discussion. I also like the discussion in Graham et al. [1989, Chapter 8]

We begin with some formal definitions from *probability*, which will give us a formal basis for describing relative degrees of certainty we give to facts about the world:

Definition 1 *Let Ω be a set called the **sample space**; assume that $|\Omega|$ is at most countably-infinite; we will deal with larger classes of sample spaces later on. A function $\Pr : \Omega \to [0,1]$ is called a **probability measure on** Ω if it satisfies that $\sum_{\omega \in \Omega} \Pr(\omega) = 1$. The pair $(\Omega, \Pr)$ is called a **discrete probability space**.*

For example, a sample space could be the set of all pairs of six-sided dice rolls:

$$\Omega_{dice} = \{(1\ 1), (1\ 2), (1\ 3), \cdots, (6\ 6)\}$$

Then, we would assign a *uniform probability distribution* over all dice rolls:

$$\text{For all } \omega \in \Omega_{dice}, \Pr_{dice}(\omega) = 1/36.$$

Then, we have a probability space $(\Omega_{dice}, \Pr_{dice})$. Or, continuing with our propositional example, Table 1 gives an example of a probability table that describes a probability distribution over the sample space of instances for a 2-variable propositional sentence involving variables x and y.

ω	$\Pr(\omega)$
(`true`, `true`)	0.1
(`true`, `false`)	0.2
(`false`, `true`)	0.3
(`false`, `false`)	0.4

Table 1: A simple probability table.

There is a substantial amount of standard notation for working with probability spaces, which we briefly review here; a nice summary resource is Darwiche [2009, Chapter 3]. An **event** is a subset of the sample space $E \subseteq \Omega$. We define the probability of an event to be the sum total of all the elements of Ω that are contained in the event:

$$\Pr(E) \triangleq \sum_{\omega \in E} \Pr(\omega). \tag{1}$$

Continuing with our above example, $\Pr(\{(d_1, d_2) \mid d_1 = 1\})$ is 1/6.

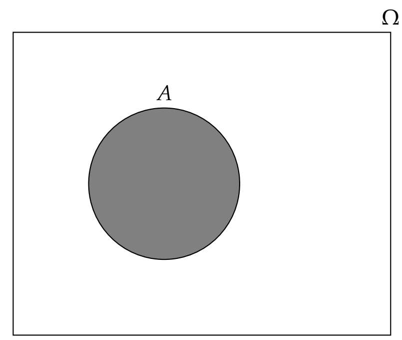

Figure 1: An event is a subset of the sample space Ω. Here, the event A is visualized using a Venn-diagram as a subset of the sample space Ω.

Typically in probability we work with quantities that are derived from the sample space; these are called random variables. Formally, a **random variable** is a map out of the sample space. For instance, we can define a random variable $S : \Omega_{dice} \to \mathbb{N}$ to be to sum two dice rolls π_1 to be the first die's value, and π_2 to be the second,where $\mathbb{N}$ is the set of natural numbers:

$$S((d_1, d_2)) = d_1 + d_2, \quad \pi_1((d_1, d_2)) = d_1, \quad \pi_2((d_1, d_2)) = d_2. \tag{2}$$

A random variable induces a probability distribution on its codomain. For a random variable $X : \Omega \to D$ acting on a probability space $(\Omega, \Pr)$, we can compute the probability of a particular element $d \in D$ by computing the total probability mass in Ω that is mapped to d, i.e.:

$$\Pr(X = x) \triangleq \Pr(X^{-1}(x)) = \sum_{\{\omega \in \Omega \mid X(\omega) = x\}} \Pr(\omega). \tag{3}$$

Another way of interpreting this notation is that the event $X = x$ is the set of elements of Ω such that $X(\omega) = x$.

Given two random variables X and Y defined on the same sample space Ω, we quite often want to characterize their joint behavior. The **joint probability distribution** of two random variables relates them:

$$\Pr(X = x, Y = y) \triangleq \Pr(X^{-1}(x) \cap Y^{-1}(y)). \tag{4}$$

For example, returning to the dice situation above, we can compute the joint probability for any particular pair of dice rolls:

$$\Pr(\pi_1 = 1, \pi_2 = 1) = \sum_{\{(d_1, d_2) \in \Omega_{dice} \mid d_1 = 1, d_2 = 1\}} \Pr((d_1, d_2)) = 1/36.$$

Given a joint probability distribution $\Pr(X, Y)$, it is often useful to "forget" one of the random variables and consider only the distribution over the remaining variables. This is called the **marginal**

probability, and it is computed by summing over all possible values the random variable can take on. Concretely, the marginal probability of X is given by "summing out" Y:

$$\Pr(X = x) \triangleq \sum_y \Pr(X = x, Y = y) \tag{5}$$

One of the most common operations one performs in probability is *conditioning*: computing the probability that some event happens *given an observation* that some other event has occurred. For instance, one might ask, "what is the probability of that the sum of a pair of dice rolls is even given that one of them is 1"? Formally, these questions are instances of *conditional probability*; notationally, the above question would be written as $\Pr(S \text{ is even} \mid \pi_1 = 1)$; the bar separates two events, and is read "given".

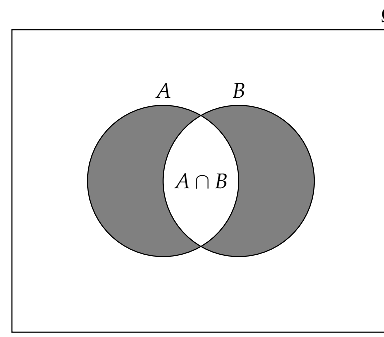

Figure 2: It is useful to visualize set intersection when understanding conditioning. By definition, $\Pr(A \mid B) = \Pr(A \cap B) / \Pr(B)$. Intuitively, this corresponds to considering the event at the intersection of these two events, and *re-normalizing* that probability based on the viewpoint of the restricted new sample space B.

Given a joint probability distribution $\Pr(X, Y)$, we define the **conditional probability** of X given Y is:

$$\Pr(X = x \mid Y = y) \triangleq \frac{\Pr(X = x, Y = y)}{\Pr(Y = y)}. \tag{6}$$

There are a number of relationships between these quantities that are commonly used. One is the law of total probability, which can be derived by combining equations (6) and (5):

$$\Pr(X = x) = \sum_y \Pr(X = x \mid Y = y) \Pr(y). \tag{7}$$

It is sometimes the case that conditioning on one random variable has no effect: for instance, in our dice-roll example, observing the value of the first die tells you nothing about the value of the second die. Formally, we say two random variables X and Y are **independent**, written $X \perp\!\!\!\perp Y$, if the following holds:

$$X \perp\!\!\!\perp Y \text{ if for all } x, y \text{ it holds that } \Pr(X = x \mid Y = y) = \Pr(X = x).$$

Exercise 1 (⋆) *See Equation 2. What is* $\Pr(S = 3 \mid \pi_1 = 1)$*?*

Exercise 2 (⋆) *See Equation 2. Show that* $S \not\perp\!\!\!\perp \pi_1$ *and* $\pi_1 \perp\!\!\!\perp \pi_2$.

Exercise 3 (⋆) *Let X and Y be random variables defined on the same sample space. Show that, if $X \perp\!\!\!\perp Y$, then* $\Pr(X = x, Y = y) = \Pr(X = x) \times \Pr(Y = y)$.

2 *Probabilistic models*

One of our main goals in this class is to automate probabilistic reasoning on a computer. For this purpose, our discussion so far of probability has been too abstract: we are told nothing about *how* these various probabilistic quantities can be computed, or how we can represent probability distributions to the computer. The means of describing a distribution to a computer is called a **probabilistic model**; it is a data-structure. The simplest kind of probabilistic model is a look-up table (or dictionary):

Definition 2 *A **probability lookup table** $\mathcal{T}$ is a lookup-table that maps elements of the sample space $\omega \in \Omega$ to the probability of that element $\Pr(\omega)$. Lookup tables support constant-time lookups.*

For instance, if we are given a discrete sample space with 4 elements $\Omega = \{1, 2, 3, 4\}$, then we can represent this as a dictionary or look-up table:

$$\mathcal{T} = [1 \mapsto 0.1, 2 \mapsto 0.2, 3 \mapsto 0.3, 4 \mapsto 0.4], \tag{8}$$

The **size** is the amount of memory required to represent the model to the computer; it is denoted $|\cdot|$. In the case of a look-up table, the size is equal to $|\Omega|$.

Querying

Now that we have a concrete description of a probability distribution, we can describe algorithms for automatically *querying* for properties about these representations. Different kinds of probabilistic models will support different kinds of queries, and have different complexity-theoretic properties of those queries. cWe have already seen the most basic kind of query: given some event $E \subseteq \Omega$, compute the probability of that event $\Pr(E)$. This immediately raises the question: how do we represent events? The most basic approach is a **set-event**, which supports constant-time membership checks (i.e., given some event $\omega \in \Omega$, a set-event E can answer the question of whether or not $\omega \in E$ in constant time). Now, we can give an algorithm for computing the probability of a set-event for a particular probability look-up table:

Algorithm 1: QueryEvent$_1$(Pr, E)

Data: A probability look-up table Pr; an set-event E
Result: The probability of the event $\Pr(E)$

$a \leftarrow 0$;
for *each* $\omega \in \Omega$ **do**
    if $\omega \in E$ **then** $a \leftarrow a + \Pr(\omega)$;
end
return a

What is the time-complexity of running $\text{QueryEvent}_1(\Pr, E)$? Since evaluating $\Pr(\omega)$ and checking membership $\omega \in E$ are both constant time, the overall time complexity is $|\Omega|$: the cost of iterating the outer loop from Lines 2–4. Is this an acceptable cost? Typically, *no*! The size of the sample space Ω is typically prohibitively large: it represents the set of *all* possibilities.

This leads us to *the fundamental design challenge of probabilistic modeling*. We desire a *concise* (i.e., small in representation size, for instance polynomial in $\log |\Omega|$) language for describing probability distributions and queries; ideally, this language is usable and interpretable by a user. Additionally, we desire we want our queries to be *tractable*: when we compute the probability of an event (or perhaps some other kind of query), we want it to be as efficient as possible in the representation size. We will see that these two attributes, tractability and conciseness, are in tension. In the case of lookup-table distributions and set-events, queries are tractable – they are typically linear $|\mathcal{T}|$, but these representations are not concise: they are as large as the sample space.

3 *Propositional logic*

What is a good language for describing probability distributions that is an alternative to lookup-table distributions and set-events? Throughout the course we will slowly grow our vocabulary for describing these things, but for now we begin with a simple language for describing sets of possibilities: *propositional logic*.

Propositional logic is a formal language and has two components: syntax and semantics. The syntax tells us how to write valid statements in propositional logic, and the semantics tell us how to interpret those statements. For example, we may wish to express the logical statement "if it rains, then it is wet". To do this, we will introduce two **propositional variables** r and w: r is true if and only if it is raining, and w is true if and only if it is wet. We can relate these two variables by using **propositional connectives**. In the above example, we will use the **implication connective** $r \Rightarrow w$, which states that if r is true then so is w.

We can summarize these connectives using the following grammar, which tells us how to form any **propositional formula** φ, α, or β in propositional logic by combining propositional variables (lower-case letters) with any of the standard propositional connectives (conjunction *land*, disjunction $\vee$, unary negation $\neg$, and bi-implication $\Leftrightarrow$):[2]

$$\varphi, \alpha, \beta ::= V \mid \texttt{true} \mid \texttt{false} \mid \neg\varphi \mid \alpha \wedge \beta \mid \alpha \vee \beta \mid \alpha \Rightarrow \beta \mid \alpha \Leftrightarrow \beta \mid (\alpha) \tag{9}$$

[2] This style of grammar is often called Backus-Naur style.

Using the above grammar, we can write down some more formulae, like $(a \wedge b) \vee (c \wedge \neg d)$ and $(a \Leftrightarrow b) \wedge w$.

Semantics of propositional logic

We now know how to write down valid syntactic formulae in propositional logic, but we have not yet described what they *mean*: how these formulae relate to the original quantities in the real world, and how to unambiguously interpret these various connectives? Importantly, we want to be able to interpret propositional formulae in terms of sets: this will let us connect back to our earlier goals for efficiently representing events and distributions.

One way to interpret a formula is by asking if, for a particular state of the world, is the formula true? To do this, we can fix an **interpretation** $I : V \to \{\texttt{true}, \texttt{false}\}$, which is a map that maps any variable $v \in V$ to a truth assignment; intuitively, this interpretation will capture some state of the world. For example, in our running example, an interpretation that states that it is currently raining and not wet would be:[3]

$$I_{ex} \triangleq [r \mapsto \texttt{true}, w \mapsto \texttt{false}] \triangleq [r, \overline{w}]. \tag{10}$$

[3] As notation, we will sometimes abbreviate $r \mapsto \texttt{true}$ as an unadorned r, and $r \mapsto \texttt{false}$ as $\overline{r}$. Additionally, we use the notation $\triangleq$ to denote definitional equality.

Now we can interpret propositional formulae for a particular interpretation I. This will be our formal notion of semantics:

Definition 3 (Semantics of propositional formulae) *An instance I is called a **model** for a formula φ, written $I \models \varphi$, if the following inductive description holds:*

$$\begin{aligned}
&I \models a && \textit{iff } I(a) = \texttt{true} \\
&I \models \neg\varphi && \textit{iff } I \not\models \varphi, \textit{ i.e. it is not the case that } I \models \varphi \\
&I \models \alpha \wedge \beta && \textit{iff } I \models \alpha \textit{ and } I \models \beta \\
&I \models \alpha \vee \beta && \textit{iff } I \models \alpha \textit{ or } I \models \beta \\
&I \models \alpha \Rightarrow \beta && \textit{iff } I \not\models \alpha \textit{ or } I \models \beta \\
&I \models \alpha \Leftrightarrow \beta && \textit{iff } I \models \alpha, I \models \beta \textit{ or } I \not\models \alpha, I \not\models \beta. \\
&I \models \texttt{true} && \\
&I \not\models \texttt{false} &&
\end{aligned}$$

Now we have a nice formal description of when a formula holds for a given interpretation I: this is our first example in the course of a formal language semantics. Here is an example:

$$[x, \overline{y}] \models x \vee y, \text{ since } [x, \overline{y}](x) = \texttt{true}.$$

GIVEN A LOGICAL FORMULA, it is often useful to determine *the set of all models* for that formula. In some sense, this tells us everything there is to know about that formula. A common way to do this is via a **truth table**, which lists (1) all possible interpretations for a fixed set of propositional variables, and (2) whether or not each interpretation models a particular formula. For example, we can give the truth table for the formula $x \vee \overline{y}$ as:

Truth tables as a way of reasoning about propositional formulae were introduced by Wittgenstein and Monk [2013].

I	$I \models x \vee y$?
$x\ y$	`true`
$x\ \overline{y}$	`true`
$\overline{x}\ y$	`true`
$\overline{x}\ \overline{y}$	`false`

Table 2: A simple truth table.

Exercise 4 ($\star$) *Give the truth table for* $(a \Leftrightarrow b) \wedge (\neg c)$.

References

Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.

Ronald L Graham, Donald E Knuth, Oren Patashnik, and Stanley Liu. Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3(5):106–107, 1989.

Robert Harper. *Practical foundations for programming languages*. Cambridge University Press, 2016.

Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

Andreĭ Nikolaevich Kolmogorov and Albert T Bharucha-Reid. *Foundations of the theory of probability: Second English Edition*. Courier Dover Publications, 2018.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

Ludwig Wittgenstein and Ray Monk. *Tractatus logico-philosophicus*. Routledge, 2013.