Propositional Probability II: Reasoning[1]

Steven Holtzen
s.holtzen@northeastern.edu

September 15, 2023

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

Now we are ready to ask: *can we use propositional logic as the basis for an effective probabilistic model*? The first question is: what is our sample space? It's clear so far that a logical choice of sample space is the set of all possibly instances for a fixed set of Boolean formulae; this sample space will have size 2^n, where n is the number of propositional variables. Now, if we want to efficiently evaluate queries over this sample space, we need need two more components:

1. A way to efficiently represent a probability distribution on the sample space;
2. A way to efficiently represent queries over that sample space.

Definition 1 (Fully factorized probabilistic model)**.** *Let $X_1, X_2, \cdots, X_n$ be jointly independent random variables (i.e., for any pair X_i, X_j, it is the case that $X_i \perp\!\!\!\perp X_j$). Then, a fully-factorized probabilistic model is a collection of n probability lookup tables $\Pr(X_i)$, one for each i. The joint probability is computed as $\Pr(X_1, X_2, \cdots, X_n) \triangleq \prod_i^n \Pr(X_i)$.*

1 *A Propositional PPL (Prop)*

Syntax of Prop:

```
φ ::= x | φ ∧ φ | φ ∨ φ | ¬φ | true | false
w ::= [x ↦ θ_x, y ↦ θ_y, ···]
p ::= (φ, w)
```

Example Prop program:

```
(x ∨ y, [x ↦ 0.1, y ↦ 0.2])
```

We will assume that all well-formed Prop programs give a probability to each variable, and in those note we will restrict ourselves to considering well-formed programs.

- Denotational semantics of Prop: we want to associate every program in Prop with the probability that φ holds according to the fully-factorized distribution described by w
- **Goal**: Define a map $[\![\cdot]\!] : \texttt{p} \to [0, 1]$ that is a map from Prop terms to real values.

For more on the history and context of different styles of semantics, see Pierce [2002, Chapter 3]

- We will need semantics for φ and the map m in order to interpret $\texttt{p}$. They have the following types:

- $[\![\varphi]\!]$ maps propositional formulae to the set of all instances that model them
- $[\![w]\!]$ produces a map from literals (propositional variables and a truth assignment) to real values

$$[\![\varphi]\!] \triangleq \{I \mid I \models \varphi\} \tag{1}$$
$$[\![[x \mapsto \theta, \cdots]]\!](x) \triangleq \theta \tag{2}$$
$$[\![[x \mapsto \theta, \cdots]]\!](\overline{x}) \triangleq 1 - \theta \tag{3}$$
$$[\![[]]\!](x) \triangleq 0 \tag{4}$$
$$[\![[y \mapsto \theta_y, r]]\!](x) \triangleq [\![[r]]\!](x) \tag{5}$$

We need to assume a propositional universe so that these equations are well-defined. Assume that all instances are defined on all free variables in `p`.

- We define the probability of an instance $[\![w]\!](I)$ as the product of probabilities of each variable in the instance. For instance,

$$[\![(x \mapsto 0.1, y \mapsto 0.3)]\!](x, \overline{y}) = 0.1 * 0.7.$$

  Note that we are interpreting the probability of a negated variable as 1 minus the probability of the positive variable.

- Now we are ready to interpret Prop programs as the sum of the probabilities of each model:

$$[\![(\varphi, w)]\!] \triangleq \sum_{I \in [\![\varphi]\!]} \prod_{\ell \in I} [\![w]\!](\ell). \tag{6}$$

- Is this a *useful* PPL? What kinds of programs can we write in it?

2 *Big-step semantics*

- Our denotational model does not tell us how to efficiently compute probabilities
- **Goal**: given a Prop program, *efficiently evaluate it*

  Running example: The simple Prop program:

```
(x ∨ y, [x ↦ 0.1, y ↦ 0.3])
```

- Worst-case hardness: computing $\Pr(\varphi_{ex})$ is NP-hard for arbitrary formulae.

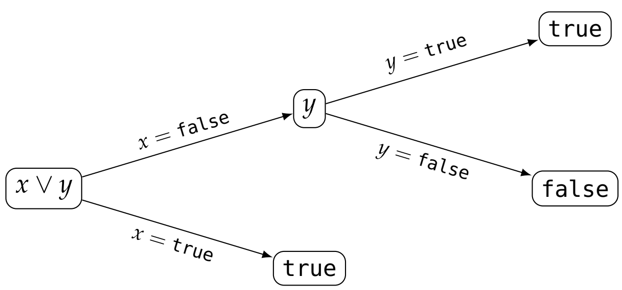
  We will show this by reduction to the **SAT problem**: the problem of determining whether or not a formula has a model.

  Reduction: Let φ be a formula. Assign a probability of 0.5 to every assignment of variables. Then, φ has a model if and only if $\Pr(\varphi) > 0$.

- In fact, computing $\Pr(\varphi_{ex})$ is #P-hard. #P is the class of problems that are polytime reducible to counting the number of solutions to an arbitrary Boolean formula. See Goldreich [2008, Chapter 6] for more discussion on this complexity class, as well as Roth [1996].

Evaluating queries via search

- *Idea*: We want to *search* through the space of models of the formula to potentially avoid exploring all possible worlds. Aim for better average-case/common-case behavior than the naive table enumeration.



- We will describe a relation $(\pi, \texttt{p}) \Downarrow^e \mathbb{R}$:
  - π is an ordered list of propositional variables. We denote list concatenation as $x :: \pi$.
  - $\texttt{p}$ is a Prop program
  - The result $\mathbb{R}$ will be equal to $[\![\texttt{p}]\!]$

- Define $\varphi[x \mapsto v]$ as the substitution where we replace all instances of x with the value $v \in \{\texttt{true}, \texttt{false}\}$ in φ, and "simplify" φ by evaluating connectives until either (1) there are no more `true` or `false` constants, or (2) the formula is equal to `true` or `false`. Example: $(x \vee y)[x \mapsto \texttt{true}] = y$.

We are being a bit informal here to save time. If we have time, we can discuss this substitution operation more formally.

- Now, let's describe our procedure for searching to solve our inference problem. We will define $\Downarrow^e$ inductively. The base-cases will be formulas without any free variables:

$$\text{(True)}\quad (\pi, (\texttt{true}, w)) \Downarrow^e 1 \qquad\qquad \text{(False)}\quad (\pi, (\texttt{false}, w)) \Downarrow^e 0$$

- Then, the inductive step:

$$\text{(Split)}\quad \frac{(\pi, (\varphi[x \mapsto \texttt{true}], w)) \Downarrow^e v_1 \qquad (\pi, (\varphi[x \mapsto \texttt{false}], w)) \Downarrow v_2}{(x :: \pi, (\varphi, w)) \Downarrow^e w(x)v_1 + w(\overline{x})v_2}$$

- Example derivation tree for our running example for the order $\pi = [x, y]$ and our program $(x \vee y, [x \mapsto 0.1, y \mapsto 0.3])$
- The **cost** of evaluating a formula φ for order π under these semantics is equal to the total number of derivations in the derivation tree.

Theorem 1. *Let* $\mathtt{p}$ *be* Prop *program and* π *be an ordering on all variables in* $\mathtt{p}$*, and* $\Pr$ *be a fully-factorized joint distribution on all variables in* φ*. If* $(\pi, \mathtt{p}) \Downarrow v$*, then* $[\![\mathtt{p}]\!] = v$*.*

Proof. We will show this by structural induction on $\mathtt{p}$. The base cases are straightforward: If $\mathtt{p} = (\mathtt{true}, w)$, then $(\pi, (\mathtt{true}, w)) \Downarrow^e 1$. By definition, $[\![(\mathtt{true}, w)]\!] = \sum_{I \models \mathtt{true}} w(I) = 1$; similar for the false case.

Induction hypothesis (IH): Let $(x :: \pi, (\varphi[x \mapsto \mathtt{true}], w)) \Downarrow^e v_1$ and $(x :: \pi, (\varphi[x \mapsto \mathtt{false}], w)) \Downarrow^e v_2$, and $(\pi, (\varphi, w)) \Downarrow^e v$. Assume by induction that $v_1 = [\![(\varphi[x \mapsto \mathtt{false}], w)]\!]$ and $v_2 = [\![(\varphi[x \mapsto \mathtt{true}], w)]\!]$. Then,

$$\begin{aligned}
[\![(\varphi, w)]\!] &= \sum_{I \in [\![\varphi]\!]} w(I) \\
&= \underbrace{\sum_{\{I | I \in [\![\varphi]\!], I(x) = \mathtt{true}\}} w(I)}_{(A)} + \underbrace{\sum_{\{I | I \in [\![\varphi]\!], I(x) = \mathtt{false}\}} w(I)}_{(B)}
\end{aligned}$$

Example: consider the formula $x \vee y$. Then:

$$[\![(x \vee y, w)]\!] = \underbrace{w(xy) + w(x\overline{y})}_{(A)} + \underbrace{w(\overline{x}y)}_{(B)}$$

Since $v = w(x)v_1 + w(\overline{x})v_2 = w(x)[\![(\varphi[x \mapsto \mathtt{true}], w)]\!] + w(\overline{x}[\![(\varphi[x \mapsto \mathtt{false}], w)]\!])$ by IH, we are done if we can show that $(A) = w(x)[\![(\varphi[x \mapsto \mathtt{true}], w)]\!]$ and $(B) = w(\overline{x})[\![(\varphi[x \mapsto \mathtt{false}], w)]\!]$.

To show this observe that after substituting, x is no longer in $\varphi[x \mapsto v]$. Example:

$$\begin{aligned}
[\![(x \vee y)[x \mapsto \mathtt{false}]]\!] &= [\![y]\!] = w(xy) + w(\overline{x}y) \\
[\![(x \vee y)[x \mapsto \mathtt{true}]]\!] &= [\![\mathtt{true}]\!]
\end{aligned}$$

Clearly there are permitted models after we restrict in this way. But, due to the factorization of w, we can prove that:

$$[\![(\varphi[x \mapsto v], w)]\!] = w(x)[\![(\varphi, w)]\!].$$

□

- What is the cost of enumerating the formula $x \vee y \vee z \vee w$ with these semantics? It will be linear-time for any order; a big improvement over the exhaustive enumeration!
- However, what about a formula like $(a \wedge b) \vee (c \wedge d)$?

3 *Memoization*

- This process is sometimes also called *variable elimination*
- Add a context to the reduction relation $\rho : \varphi \to [0,1]$ that maps formulae to their probability of being true
- Relation now has the shape:

$$(\pi, \rho, \mathsf{p}) \Downarrow^m (\mathbb{R}, \rho')$$

References

Oded Goldreich. Computational complexity: a conceptual perspective. *ACM Sigact News*, 39(3):35–39, 2008.

Benjamin C Pierce. *Types and programming languages*. MIT press, 2002.

Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.