Propositional Probability II: Reasoning[1]

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

Steven Holtzen
s.holtzen@northeastern.edu

September 14, 2023

Now we are ready to return to our earlier discussion and 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.

Let's tackle (1) first. What is an alternative representation of a distribution that avoids the space-explosion we saw in the lookup-table representation? We will need to be more clever about how we represent probabilities. One useful choice is to assume that all random variables are *independent from each other*, and therefore we can concisely describe a joint distribution over all of them:

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 $\Pr(X_1, X_2, \cdots, X_n) \triangleq \prod_i^n \Pr(X_i)$.*

Observe that a fully-factorized model only requires $O(\sum_i |X_i|)$ space, which is significantly smaller than $|\Omega|$.[2] This is a big improvement over plain lookup tables, but it comes at a cost of expressivity: there are some distributions that cannot be represented in a fully-factorized way.

[2] The notation $|X|$ refers to the size of a random variable's co-domain.

In the context of propositional logic, there is a natural notion of a fully-factorized model. Let Ω be the collection of n-element instances. Then, each propositional variable corresponds to a projection out of that sample space. Let's call this the **fully factorized propositional distribution**: each variable in the theory will be associated with an independent probability of being `true` or `false`. For example, we may have to propositional variables x and y; the fully factorized

distribution could be given by two independent lookup tables:

$$[x \mapsto 0.1, \overline{x} \mapsto 0.9], \quad [y \mapsto 0.2, \overline{y} \mapsto 0.8]. \tag{1}$$

Then, the joint probability $\Pr(x, \overline{y})$ is 0.1×0.8.

NOW WE RETURN TO AN EARLIER QUESTION: how do we efficiently represent events in the propositional setting? Events are a subset of the sample space. The sample space in propositional probability is the set of all instances. Then, we can represent events by *propositional formulae*:

$$\Pr(\varphi) \triangleq \sum_{I \models \varphi} \Pr(\varphi). \tag{2}$$

For instance, for the fully-factorized distribution in Eq. 1, we can compute the probability of a propositional sentence $x \vee y$ by summing the probability mass over each instance that models this formula:

$$\Pr(x \vee y) = \Pr([x, y]) + \Pr([x, \overline{y}]) + \Pr(\overline{x}, y). \tag{3}$$

This is implemented by the following algorithm:

Algorithm 1: QueryPropositionalEvent$_1(\Pr, \varphi)$

Data: A fully-factorized distribution Pr; a propositional formula φ
Result: The probability of the event $\Pr(\varphi)$

1 $a \leftarrow 0$;
2 **for** *each* $I \in \Omega$ **do**
3 | **if** $I \models \varphi$ **then** $a \leftarrow a + \Pr(\omega)$;
4 **end**
5 **return** a

Unfortunately, QueryPropositionalEvent is *still linear* in the size of $|\Omega|$ due to the outer loop on Lines 2–4. In all cases, regardless of the structure of φ, we have to consider all all the instances, which will not scale.

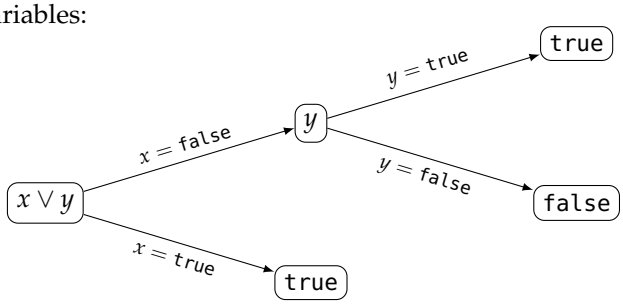
Efficiently evaluating queries via search

How do we avoid scaling in $|\Omega|$ during querying? In other words: is it possible to design a scalable automated reasoning strategy for our tiny probabilistic programming language that we've made so far, with fully-factorized propositional distributions and queries given as propositional formulae? In the worst case, it may not be possible

to avoid considering all possible instances, but we can try to avoid this worst-case explosion by delicately *exploiting the problem structure*: in particular, we want to simultaneously exploit the fully-factorized structure of the distribution along with the fact that all events are written as propositional formulae.

One approach is to *exploit the structure of models of the query* φ: sometimes, a query may only have a single model, or perhaps most instances are models. In these cases, we waste a lot of effort enumerating all models; it is much more efficient to *search* for the few satisfying instances.

For example, consider the formula $x \vee y$. We can recursively decompose this formula by branching on assignments to propositional variables:



The root of this binary tree is the original formula $x \vee y$. Each branch assigns a propositional variable to a Boolean value. The children are the resulting formula evaluated on the *partial instance* given by the path up until that point: for instance, if we assign $x = \texttt{false}$, we are left with the formula $\texttt{false} \vee y = y$. Leaves of the tree denote whether or not a particular assignment is a model.[3]

Algorithm 2: PrEvent(Pr, φ)

Data: A fully-factorized distribution Pr; a propositional formula φ; a decision order σ.
Result: The probability $\Pr(\varphi)$.

```
1 if φ = true then return 1;
2 if φ = false then return 0;
3 v ← head(σ);
4 σ' ← tail(σ);
5 φ_v ← fix v = true in φ;
6 φ_v̄ ← fix v = false in φ;
7 return Pr(v)×PrEvent(Pr, φ_v) + Pr(v̄)×PrEvent(Pr, φ_v̄)
```

You should convince yourself that the above algorithm is correct by trying it on a few examples. It is a good exercise to prove it correct inductively. What is the runtime of Algorithm 7? In the worst-case, it is still $|\Omega|$. But, sometimes – depending on the structure of φ – it

[3] (under construction)

can do better by terminating the recursion earlier in a base-case on Lines 1 and 2, rather than exhaustively exploring all instances like Algorithm 5.

Exercise 1 ($\star$) *Assume the following fully-factorized distribution on propositional variables:*

$$[x \mapsto 0.1, \overline{x} \mapsto 0.9], \quad [y \mapsto 0.3, \overline{y} \mapsto 0.7].$$

Use an exhaustive search tree to compute $\Pr(x \Leftrightarrow y)$.

Exercise 2 ($\star\star$) *Prove Algorithm 7 correct.*

References