An Agenda for Probabilistic Programming: Usable, Portable, and Ubiquitous

Andrew D. Gordon
Microsoft Research and University of Edinburgh
January 2013

The idea of probabilistic programming is that the user writes a probabilistic model for a Bayesian inference problem as a short piece of code, while the compiler turns this code into an efficient inference routine. Probabilistic programming systems include BUGS, IBAL, BLOG, Church, STAN, Infer.NET, and Factorie, amongst others. Application areas include scientific modelling, information retrieval, bioinformatics, epidemiology, vision, seismic analysis, semantic web, business intelligence, security, human cognition, and more. Probabilistic programming goes beyond the standard abstractions of classification and regression, and allows inference based on custom distributions definable in code, including recommendation systems, rating systems (as in TrueSkill, for instance), and other graphical models. The promise is that not just machine learning researchers, but also data scientists, domain experts, and general developers can specify custom machine learning solutions by authoring small probabilistic programs, while leaving the inference mechanisms to the compiler. Delivering on this promise would democratize machine learning, by empowering orders of magnitude more users to apply it to their problems.

So what are the obstacles to overcome?

Performance of inference backends has been the subject of much research and innovation, including hardware-based solutions as well as software enhancements. For example, many talks at the recent Probabilistic Programming workshop at NIPS 2012 addressed performance, including scalability to large datasets.

Beyond performance, here are three problem areas where we need to make progress if probabilistic programming is truly to democratize machine learning.

Usability, Usability, Usability

The first problem is usability. A straw poll suggests that when doing probabilistic programming only 10% of the developer's time is spent writing the model, while the rest is spent on getting inference to run robustly and efficiently. It is instructive to run programs forward to generate synthetic data, but very frustrating to discover that inference fails on a program even though ordinary execution succeeds. Like other forms of declarative programming (such as logic programing), programmers may at first hope that they need only understand the abstract declarative semantics of a program, but soon discover they must also understand the underlying inference mechanism. Much time is spent tuning hyperparameters of prior distributions, numbers of iterations, tuning performance, and understanding results of inference.

Poor usability could be a show-stopper for probabilistic programming: the worry is that experts produce a series of elegant, working models, but then users discover that apparently insignificant changes breaks inference, and give up on probabilistic programming.

A Common Language for Portable Probabilistic Programming

The second problem is fragmentation: there are almost as many probabilistic languages as there are probabilistic compilers. A common language with substantial benchmark suites and multiple implementations would benefit system developers wishing to benchmark performance, and would benefit end users wishing to port their models between systems. BUGS and Church are the probabilistic programming languages with multiple implementations at present. Still, a well-engineered probabilistic fragment of some existing widely used language could become a very successful probabilistic language. R is the language of choice of the current population of data scientists, and indeed R is on the curriculum of the raft of new courses training up the next generation of data scientists. As far as I know, there are no implementations yet of a Probabilistic R, but the widespread adoption of R makes a probabilistic version next to inevitable. It will be a great opportunity to establish benchmark suites in Probabilistic R to test rival probabilistic programming systems via regular competitions.

Still, subsetting an existing programming language has usability problems, as users may stray outside the supported subset. Moreover, different backends for a common language may support different subsets. For example, we could drive both Church and Infer.NET systems from R models, but these two systems would support rather different subsets of R (because Church depends on general recursion and memoization, while Infer.NET supports graphical models based on iteration over arrays). IDEs should help users track and understand compatibility between their code and multiple backends – ideas from the DrRacket IDE, which supports multiple "language levels" (such as Beginning Student, Intermediate Student, and so forth) may help.

Probabilistic Metaprogramming for Ubiquitous Machine Learning

The third problem is that potential beneficiaries of the flexible models enabled by probabilistic programming may not have the time, inclination, or aptitude to learn to write and debug probabilistic programs.

A promising direction is *probabilistic metaprogramming,* where we automatically synthesize probabilistic programs to model given datasets. One early example is Singh and Graepel's InfernoDB, which given the schema of a relational database, automatically constructs an Infer.NET model suitable for inferring missing data and detecting outliers in the database. Another example is Veritable, a predictive database developed by PriorKnowledge, recently acquired by SalesForce.com, which aims to let developers make predictions as easily as they do joins; developers use a SQL-like query language which hides the details of an automatically generated probabilistic program.

Mark Weiser envisaged an era of ubiquitous computing in which information processing is thoroughly integrated into physical objects and activities, and human users may be unaware that they are interacting with computers. By analogy, we envisage an era of *ubiquitous machine learning* in which statistical inference is thoroughly integrated into our interactions with datasets, be they spreadsheets, or databases, or data structures in programs. InfernoDB and Veritable are early steps towards this era, when machine learning will be thoroughly democratized.