Designing an IDE for Probabilistic Programming: Challenges and a Prototype

Sameer Singh[§] **Sebastian Riedel**[†] **Luke Hewitt**[†] **Tim Rocktäschel**[†]
[§] University of Washington, Seattle WA [†] University College London, London UK

By providing intuitive, declarative, and expressive languages, probabilistic programming languages (PPLs) such as FACTORIE [McCallum et al., 2009], Church [Goodman et al., 2008], Infer.NET [Minka et al., 2010], Figaro [Pfeffer, 2009], and MLNs [Richardson and Domingos, 2006] have made significant strides in closing the gap between conventional programming and complex probabilistic modeling. However, development is not a *program once and done* process, instead often requires many iterations of programming, testing, and debugging. Iterative programming is even more common for developing sophisticated probabilistic models over complex data. Initial intuitions that go into model design are often inaccurate, requiring incremental model tweaking based on performance. Even when the model is accurate, different inference approaches target different approximations; the final performance depends quite critically on the choice of the algorithms and its hyper parameters. Lastly, bugs are often introduced by the user, many of which may not even reflect in the performance. For example, an error in feature computation does not necessarily lead to a completely inaccurate classifier. Supporting iterative probabilistic programming is hence crucial for real-world adoption of PPLs.

One approach to facilitate iterative programming is to allow the user to customize aspects of the model and inference. Existing PPLs vary considerably in this respect. For the sake of simplicity, many PPLs such as Church [Goodman et al., 2008] and MLNs [Richardson and Domingos, 2006] focus on declarative model descriptions that allow little flexibility for the users to inspect the inner workings of the language[1]. Other languages, such as Infer.NET [Minka et al., 2010], provide predefined *hooks* into the models; the user is free to use the default implementations, but can inject custom code to debug/improve the performance. FACTORIE [McCallum et al., 2009] also supports such hooks, but further enables the users to override arbitrary aspects of inference to provide unrestricted access. Although powerful, enabling user modifications is of limited use unless appropriate feedback on the performance of the model and inference on user's data is surfaced.

In this paper, we study the challenges in designing an integrated development environment (IDE) for probabilistic programming languages. Along with supporting the features of conventional IDEs such as automated build tools, sophisticated code editor, and the traditional debugger, we argue that building a PPL-IDE raises unique challenges such as visualization of distributions over structured data, debugging performance of model and inference, and adapting to user's expertise in machine learning. We also present a prototype implementation of an IDE for the WOLFE [Riedel et al., 2014] language that aims to address some of these concerns. Our implementation allows visualization of a variety of structured distributions and multiple granularity of debugging inference, while also providing other conventional IDE features such as syntax highlighted code editing with auto completion, server-side execution of inference, and an easy to use, platform-independent interface.

1 Design Challenges

As we describe above, an IDE for probabilistic programming at the very least needs to support all the features of conventional IDEs since much of the same requirements apply to PPLs. But further, as the connections between model and inference approximations and the final performance defies even machine learning experts, users need to iterate on the choices of modeling, inference, and hyper-parameters, till satisfactory performance is obtained. To this end, interfaces for machine

[1] Venture, an interactive console for debugging Church, is under development and seems promising.

learning have been proposed [Talbot et al., 2009, Patel et al., 2010, Amershi et al., 2011], however they focus primarily on classification. Inspired by these, and conventional programming IDEs, we outline a number of challenges that need to be addressed to support a PPL-IDE.

1.1 Visualizing Probabilistic Model and Inference Algorithms

The target audience for PPLs includes users with a variety of expertise in machine learning. Many users are often newcomers to machine learning, familiar with only the basics of probabilistic modeling. Even users that are familiar with machine learning, many are not experts (have only basic comprehension of graphical modeling and inference). An IDE designed for PPLs should adapt to each of these classes of users, that is, surface only data and predictions for newcomers, provide overview of the model (e.g. plate description) and inference (e.g. convergence properties) to the non-experts, and present detailed description for experts.

Models. As we describe above, the IDE needs to visualize the model at multiple levels of granularity: consider interpretable representation of the model for users to quickly check, create plate model representations, or for the expert users, a detailed grounded model with description of the potentials (as tables, functional expressions, or plots, as appropriate). Further, the IDE should allow the users to quickly identify the regions of interest, for example, by linking predictions to parts of the model that are used in its computation.

Inference. Even more so than the model, adapting to user expertise is key for debugging inference. Many PPLs treat inference as a black-box, however many choices need to align for practical solutions. IDE needs to encourage an understanding of the performance of inference, and the effect of these choices on the predictions. To support a variety of inference algorithms, the IDE needs generic, inference-agnostic visualizations, perhaps based on the optimization perspective of inference (plotting the objective and measures of convergence), and support for hyper parameter optimization. For specific inference algorithms, IDE should present measures that would be informative to the expert users, for example visualizing the messages, schedules, and the residuals for message passing.

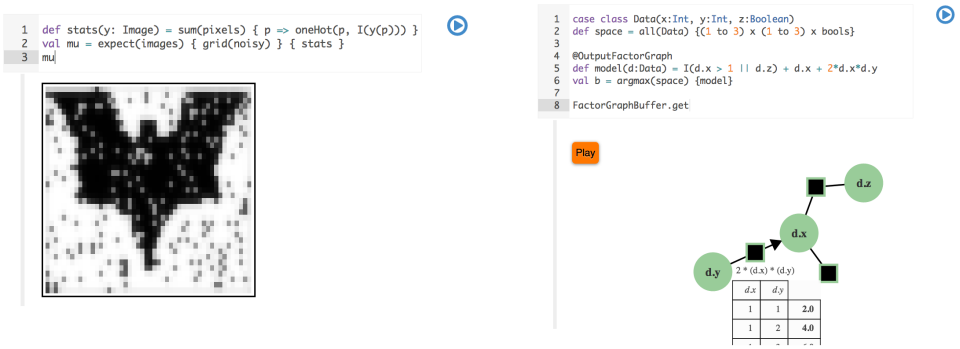
1.2 Language Agnosticism

Active research in the field has resulted in a large number of PPLs, each with their own set of advantages and disadvantages. To identify which language is appropriate for an application at hand, an IDE that supports multiple languages would significantly benefit the user in comparing and contrasting the various PPLs. Thus we should aim to design IDEs that are universal. This is another feature that is common in conventional IDEs, however they often rely on a common *intermediate representation* (IR) that all the supported languages compile to. An IR for PPLs is a challenging endeavor that is in nascent stages of research, and since it is not even clear whether such a representation is possible, the possibility of IDEs that support multiple PPLs seems bleak. On the other hand, much of our desired features may be obtained by defining a common API that allows PPLs to present their underlying graphical model, inference execution traces, predicted distributions on the data, etc. for visualization in an IDE.

1.3 Structured Data and Marginals Visualization

An IDE for PPL needs to support a variety of data types that span a majority of tasks and domains, without necessarily relying on the user to provide the code for visualization. Traditional IDEs support simple visualizations for primitive types, such as strings, numbers, and collections of primitive types, which a PPL-IDE will have to support. However, for PPL applications, data is often quite complex and structured, for example sequences and trees in NLP, graphs in social networking, matrices and images in computer vision, vectors in distributed representations, distributions, and so on.

Structured Marginals: The PPL-IDE should not only be able to visualize the datasets themselves, but also, when possible, distributions over these data structures. The IDE should include visualization of commonly-used parameterized distributions, such as Gaussian, Dirichlet, etc. but also intuitive presentation of empirical distributions, in the form of easy-to-slice histograms.



(a) Visualization of prediction marginals

(b) Factors, potentials, & message animation

Figure 1: **A Prototype IDE for Probabilistic Programming:** REPL interface with (a) visualization of marginals, and (b) inference animation on the factor graph. We will provide a URL of a live demonstration.

2 Prototype Implementation for WOLFE

Following the guidelines above, we describe a fairly mature prototype of a PPL-IDE here. Our IDE is heavily inspired by iPython notebook, allowing an easy-to-use, browser-based read-eval-print-loop (REPL) for the WOLFE language that supports a variety of interactive visualizations.

Along with support for primitive types, and of collections and case classes recursively, the IDE can visualize interactive graphs, sequences, matrices, plots, and a number of file formats such as PDFs and images. By focusing on a variety of data types, we feel the IDE covers a majority of possible use cases, or will require minimal transformation code to a supported format. For distributions, the IDE supports interactive visualizations of histograms, and of structured distributions if it can be converted to a supported representation, for example marginals for binary matrix are a real-valued matrix.

For an interactive visualization of the model, we currently support only a single level of granularity: the grounded model with all the factors and potential tables. Inference visualization has two levels: at the trace level we show plots of convergence metrics over time, and at the detailed level, for messages passing on a factor graph, we present the execution as an animation, showing the messages, schedule, and the marginals for each node.

The IDE also offers a number of additional, useful features. The implementation is browser-based, allowing platform-independence for the user while achieving scalability as inference takes place on a server that may be deployed on high-performance machines. The interface is simple and intuitive, and like iPython notebook, can be used for documentation of the probabilistic programming process. Finally, the interface has support for complete Scala, allowing the user to use the same environment for data preprocessing, modeling, and downstream applications.

Future Work: There are a number of limitations we would like to address. The IDE supports a variety of model, inference, and prediction visualizations, but is unable to link them to each other. This is partly due to the strict adherence to the REPL framework; a slight deviation may allow more interactive and real-time feedback. We would like to better adapt to user expertise, for example explore automated plate model generation from the declarative description. Finally, we would like to define a simple API that allows support for multiple PPLs.

3 Conclusions

IDEs leads to effective software engineering by allowing users to build and debug programs iteratively using intuitive user interfaces. PPLs have matured to the point that practical implementations require IDEs to comprehend, debug, and improve modeling and inference choices. In this paper, we present the unique challenges required for developing such a system, for example visualization of structured marginals, presentation of the model and inference algorithms, designing PPL-agnostic IDEs, and scalability to user expertise. We also present an open-source prototype IDE implementation for WOLFE that aims to address many of these issues.

References

Andrew McCallum, Karl Schultz, and Sameer Singh. FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *Neural Information Processing Systems (NIPS)*, 2009.

Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence (UAI)*, 2008.

Tom Minka, John M. Winn, John P. Guiver, and David A. Knowles. Infer.NET 2.4, 2010. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Avi Pfeffer. Figaro: An Object-Oriented Probabilistic Programming Language. Technical report, Charles River Analytics, 2009.

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2): 107–136, 2006. ISSN 0885-6125.

Sebastian Riedel, Sameer Singh, Vivek Srikumar, Tim Rocktaschel, Larysa Visengeriyeva, and Jan Noessner. Wolfe: Strength reduction and approximate programming for probabilistic programming. In *International Workshop on Statistical Relational AI (StarAI)*, 2014.

Justin Talbot, Bongshin Lee, Ashish Kapoor, and Desney S Tan. Ensemblematrix: Interactive visualization to support machine learning with multiple classifiers. In *ACM Conference on Computer Human Interfaces (CHI)*, 2009.

Saleema Amershi, James Fogarty, Ashish Kapoor, and Desney S. Tan. Effective end-user interaction with machine learning. In *AAAI Conference on Artificial Intelligence, Nectar Track*, 2011.

Kayur Patel, Naomi Bancroft, Steven M. Drucker, James Fogarty, Andrew J. Ko, and James A. Landay. Gestalt: Integrated support for implementation and analysis in machine learning processes. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2010.