Evaluating Progress in Probabilistic Programming through Topic Models

Francis Ferraro
Dept. of Computer Science
Johns Hopkins University

Benjamin Van Durme
HLTCOE
Johns Hopkins University

Yanif Ahmad
Dept. of Computer Science
Johns Hopkins University

Abstract

Topic models have proven versatile over the past decade, particularly as partial embeddings within more intricate models. These models present challenges that are analytic, computational and engineering in nature. Advances in probabilistic programming have the potential to circumvent a number of these issues, but researchers need a way to coarsely evaluate these frameworks. We identify three axes of a successful framework and argue that computational efficiency of straight LDA provides one such lens. We provide and release a modular open-source testbed to systematically capture one aspect of current probabilistic programming and discuss initial results on both heavily-controlled and "real" data.

1 Introduction

LDA [1] is often the "bread and butter" of more complex models, explicitly as a subgraph or as an implicit conduit for modeling and parameter estimation techniques. The past decade has simultaneously seen considerable effort both to reduce the computation requirements of existing techniques, as well as adapt sampling methods to new models. While both have depended on model topology analysis and clever computational simplifications, there are a number of less interesting, but possibly tedious and error-prone, details that can hinder increased complexity and novel application.

Frameworks for probabilistic programming can abstract away the details of computation with random variables, so the model designer can treat them just as a programmer treats `ints` or `doubles`. Researchers can explore models more felicitous to data without worrying how weaker independence assumptions affect implementation. General inference techniques, those used time and again, can be predefined and optimized, while at the same time allowing for development of novel ones (since much of the bookkeeping may be in place). But until probabilistic compilers are able to analyze models on a deep enough level, these frameworks can be expected to be less efficient than hand-rolled code. It is with these tradeoffs in mind that we compare frameworks in both controlled and "real" settings. We also identify three axes on which a framework may be judged: scalability, ease-of-development, and extensibility (e.g., how easy it is to define new sampling methods).[1]

Our goal is to take a first step at performing a reproducible and extensible evaluation of probabilistic programming frameworks in active development; it is not to benchmark specific implementations of LDA. We note that much time and effort have been devoted to creating scalable implementations of LDA [2, 3, i.a.]. Further, our evaluation cannot be comprehensive across the plethora of probabilistic frameworks under active development. Crucially, the frameworks are subject to high churn; we would be hesistant to make conclusive, possibly damaging statements regarding a framework's *potential* based on a particular (possibly pre-beta) snapshot. Thus, we opt for a deeper, controlled, *reproducible* evaluation of a much smaller set of frameworks that explores computational efficiency on a number of axes relevant to topic modeling; future work would do well to continue our efforts

[1] An undercurrent of interest in these frameworks is gaining momentum; see DARPA PPAML.

on a broader scale.[2] We release a modular testing framework, capable of running these experiment for additional frameworks.[3]

2 What is a Probabilistic Programming Framework?

A framework can be a mix between a language and a library: a library is auxiliary to an existing language, and to its functionality/resources, while a probabilistic language "compiles" models based on its own syntax (and hence semantics). Some frameworks can be nicely categorized, while others are hybrids; the distinction helps highlight framework differences.

Due to space requirements, we focus our evaluation on one language (BLOG [4]), one library (FACTORIE [5]) and one "hybrid" (STAN [6]). **BLOG** can be thought of as a generalization of probabilistic relational models [7] where any given random variable may have an infinite number of parents and the assumptions regarding unique names and domain closure have been removed. The intuitive syntax makes developing new models relatively easy, but a lack of easy IO makes larger scale testing cumbersome. On the other hand, **FactorIE** [5] is a Scala library that uses the object-oriented paradigm to define relational factor graphs. Due to the expressivity of Scala, researchers can define models declaratively, though a more hands-on imperative approach is generally used in practice. Inheritance allows extending existing inference methods, if desired. Both BLOG and FACTORIE are under active development; FACTORIE in particular has an active userbase. (**Figaro** [8], like FACTORIE, is a Scala library relying on many of the same principles.) **Stan** [6] is an appealing framework based upon the "No U-Turn Sampler," a more robust extension of Hamiltonian MC. STAN can declaratively define models and serve as a C++/R library; unfortunately, current versions of STAN (1.3, and 2.0.1) do not fully handle mixture models parametrized by discrete latent variables. While STAN can marginalize out the discrete variables, one must (manually) write model-specific code in order to compute posteriors or to label unsupervised data.

The **Hierarchical Bayes Compiler (HBC)** [9] focuses on prototypability for discrete and continuous hierarchical Bayesian models, declaratively written in a simple language and compiled into C++. HBC knows of a number of inference techniques and optimizations, but its extensibility is highly limited. Recent NLP applications have used HBC as a way to generate initial sampling code [10, 11]. **Church**, a language embedded within Scheme with medium prototypability, samples computation traces based on an adaptive proposal distribution, making sampler extensions potentially difficult [12]. Linear extrapolation of benchmarks from a toy corpus suggests the two most prominent implementations are not sufficient [13].

Finally, we wish to call attention to two other frameworks especially worthy of future consideration. First, **Infer.NET** [14] is a C# library akin to FACTORIE and FIGARO. It has been used successfully, with reported high prototypability and a reasonable memory footprint (by modern standards) [15]. Second, **libDAI** [16] is an open source C++ library that supports a number of inference algorithms for factor graphs over discrete variables. It has many preimplemented techniques, both exact and approximate, and aims to ease research of novel inference algorithms.

3 Evaluation Considerations

We use MALLET, a well-known Java library for, among other tasks, topic modeling and graphical models, as our "gold standard" [17]. For LDA especially, this a very difficult baseline to compare against, as it has been finely tuned.[4] We attempted to remove all costly operations not directly relevant to inference, such as printing of intermediate results. Further, to ensure a fair comparison, we prevented all systems from using builtin preprocessers, such as stop-word removal.

[2] A supplementary axis of evaluation would be on a framework's memory footprint. While memory use is certainly an issue, we adopt the maxim that "space is cheap(er)" and leave this consideration to future work, as we believe a framework's absolute speed will generally be the first hurdle that must be overcome.

[3] `http://cs.jhu.edu/~ferraro`

[4] "Out-of-domain" researchers, for instance, those in the digital humanities, have found MALLET to be a useful tool [18, i.a.]. Comparing against MALLET could, in future evaluations, serve as a proxy for off-the-shelf usability; we do not explicitly consider this type of evaluation.

Our quantitative evaluation metric is seconds *per iteration*. To get a consistent measure, we average multiple runs, each budgeted a maximum of 1,000 iterations, for up to twelve hours. Note that we are not comparing the *aggregate* time for 1,000 iterations of, e.g., FACTORIE vs. the *aggregate* time for 1,000 iterations of, e.g., BLOG. While one can draw this conclusion from our results, we stress that such inferences may be myopic. Most importantly, not all samplers are created equal: some, such as NUTS, may require more computation per iteration than others, such as collapsed Gibbs, but as compensation make more progress, per iteration, toward a better solution. Nevertheless, we maintain that this time/iteration metric, which can applied to any framework in any stage of development, is a valid basis for comparison, particulary since one of the first winnowing questions one may have is how well a framework can generally scale.

Of course, measuring performance in seconds/iteration is not infallible. LDA *is* very common, and development teams may (or may not) internally benchmark against it. Therefore, there is a possibility of overfitting to LDA. An alternative quantitative metric, which captures how well a framework's sampler works, could be held-out log-likelihood. However, this metric is problematic: in addition to any standard concerns, it is susceptible to framework overfitting as well. Further, it does not adequately capture the inference capabilities of systems: a number, such as BLOG and FACTORIE, allow the programmer to write custom (better) inference techniques, resulting in tighter fit but at increased effort. Third, at evaluation time, not all frameworks allowed for these types of queries, resulting in non-comprehensive comparisons. Given these issues, as well as space considerations, we focus solely on the time/iteration measure for quantitative evaluation.

For qualitative measures, we supplement the analyses of the three frameworks according to their prototypability and extensibility. It is tempting to present the standard top k words per topic. However, this suffers from the same problems as the above predictive quantitative proposal: frameworks can be overfit to LDA, better inference techniques may result in better topic distributions, and not all frameworks can perform the required discrete labeling.

4 Experimental Setup and Results

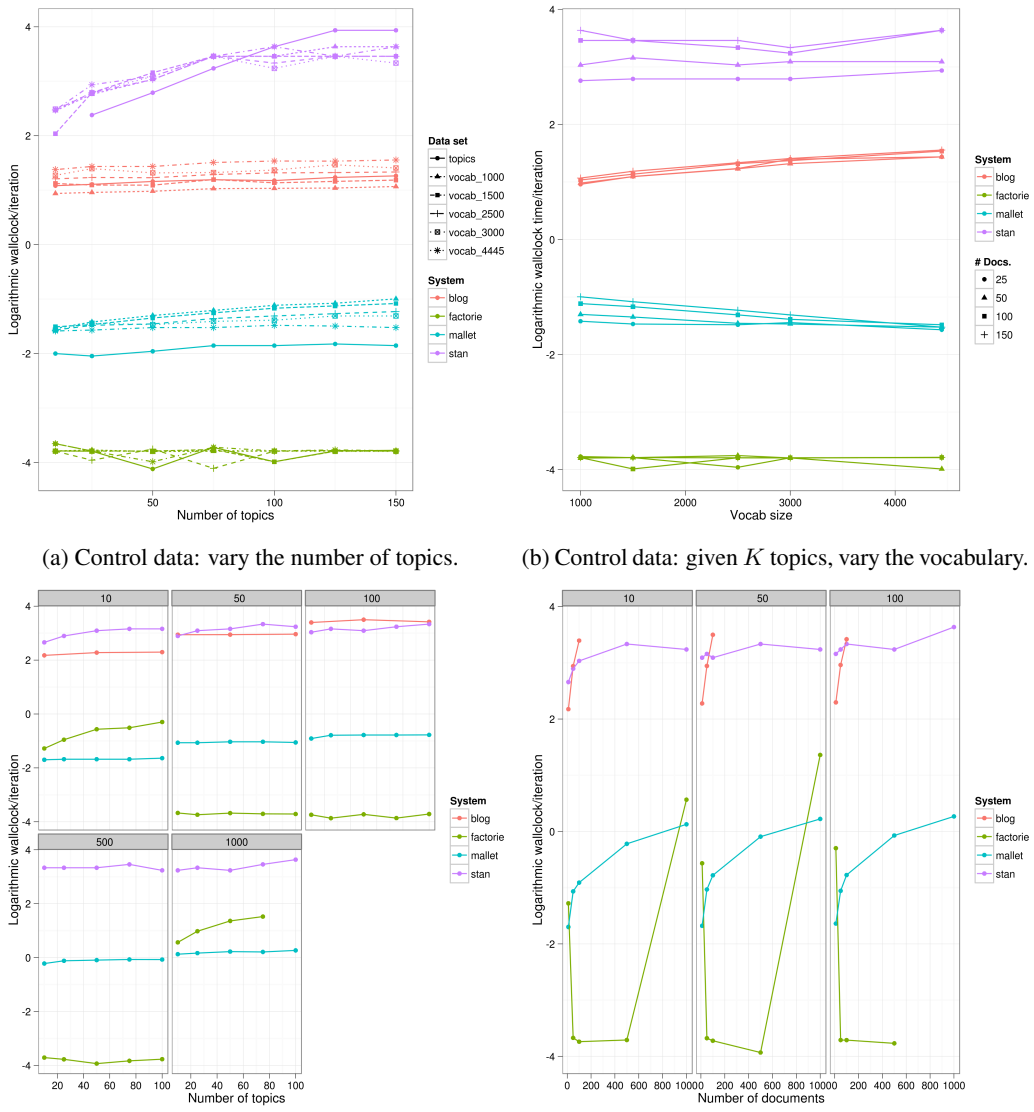
We partition our evaluations into those with "lab-style" (heavily controlled) conditions and those with "real world" conditions, each with separate datasets available upon publication. Our heavily-controlled experiments, (Fig. 1, top), operate on a fixed set of 250 documents from the training portion of the 20news corpus; we vary the size of the vocabulary and the number of topics. All documents have a lower-cased reduced vocabulary $\mathcal{V}$, where the top 35% of IDF-weighted words, as well as those appearing fewer than fifty times, have been removed. We experiment with a vocabulary of size V ("topics" curve of Fig. 1b) by fixing documents to 150 tokens (slightly less the average length of documents in the processed training portion) sampled uniformly from the document, and replacing all words not in the most frequent $V - 1$ with a special catch-all symbol. To experiment with the number of topics K (Fig. 1a), we further constrain $\mathcal{V}$ to the top 5000 word types ($\mathcal{V}'$) using the above UNK scheme and fix the document length to 150 tokens.[5]

The "real" conditions (Fig. 1, bottom) consider a more limited set of variables: the number of documents vs. topics. While here it can be harder to draw any firm conclusions — the document length, vocabulary and the number of tokens can vary drastically — these experiments reflect more realistic, "end-user" concerns. The "real" data are increasingly larger sets of Annotated Gigaword documents [19], selected according to the number of coreference chains a document has.[6]

The per-iteration timing results are shown in Figure 1. Perhaps surprisingly, FACTORIE surpasses MALLET on both the controlled tests as well as on most of the real data. The relatively flat "topics" curve in Figure 1a indicates that the frameworks *can* scale well for certain data. When considering the time per-iteration, there is still a lot of room for improvement. However, it is prudent to keep in mind our earlier comment regarding comparing samplers: whether a framework returns a good enough solution sufficiently fast is application dependent. The orders of magnitude differences may be acceptable if, e.g, a framework is sufficiently usable (for quick model development), or fewer iterations are needed to get the "general idea" of a particular model's appropriateness for given data.

[5] One could imagine varying the number of (per document) observations N as well. While for simplicity we opted against this evaluation, extending our framework would be simple enough.

[6] This reflects a recent tendency to define LDA-inspired models over shallow semantic annotations [11, 20].



(a) Control data: vary the number of topics.

(b) Control data: given K topics, vary the vocabulary.

(c) Vary the number of topics on the "real" data.

(d) Vary the number of "real" [19] documents.

Figure 1: Average logarithmic wallclock results, in seconds per iteration, for evaluated systems on controlled (top) and real sets (bottom) per iteration. Missing data are due to 12 hour timeout.

The remaining Figure 1a curves suggest that for this control data, BLOG is slightly less sensitive than MALLET at changing both V and K, while STAN most likely suffers from having to marginalize. However, Figure 1, bottom, indicates how real, unnormalized data can affect performance, including that a twelve hour timeout may not yield timing data. Finally, note that these tests only compare *per-iteration runtime* and do not examine quality of solution. While for some frameworks this is easy, for others effective ways of judging quality have not yet been implemented; this and storage requirements indicate areas for future quantitative evaluation.

On the qualitative side, we note that the degree of (sampler) extensibility in all evaluated frameworks requires one to go low-level. FACTORIE, as a Scala library, may be slightly easier than BLOG, which can be extended but may need recompilation of the entire framework. Judging (model) prototypability should focus on fundamental issues – such as model specification limitations or sampler assumptions – from more easily fixable ones – such as improved documentation and commented examples – though untangling these can be difficult. On the more fundamental aspect, we found, as is standard, that more declarative frameworks tended to have higher prototypability, e.g., both BLOG's and STAN's syntax encourage "writing down the math" in transparent ways.

5 Conclusion

We have presented a way to use topic models to partially evaluate probabilistic programming frameworks. Our open-source modular testbed easily allows future comparisons of additional probabilistic frameworks, an aspect that is crucial given their active development.

Acknowledgments We would like to thank the two anonymous reviewers for their helpful comments and perspectives. This material is based on research sponsored by DARPA under agreement number FA8750-13-2-0017 (the DEFT program). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes. The views and conclusions contained in this publication are those of the authors and should not be interpreted as representing official policies or endorsements of DARPA or the U.S. Government. A National Science Foundation Graduate Research Fellowship, under Grant No. DGE-1232825, supported the first author.

References

[1] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[2] Matt Hoffman, David M Blei, and David M Mimno. Sparse stochastic inference for latent dirichlet allocation. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1599–1606, 2012.

[3] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein, and C. Welton. Mad skills: new analysis practices for big data. *VLDB*, 2(2):1481–1492, 2009.

[4] B. Milch, B. Marthi, S. Russell, D. Sontag, D.L. Ong, and A. Kolobov. BLOG: Probabilistic Models with Unknown Objects. *Introduction to statistical relational learning*, 2007.

[5] A. McCallum, K. Schultz, and S. Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *NIPS*, page 137, 2009.

[6] Stan Development Team. Stan: A c++ library for probability and sampling, version 1.3, 2013.

[7] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[8] A. Pfeffer. Figaro: An object-oriented probabilistic programming language. Technical report.

[9] H. Daumé III. HBC: Hierarchical Bayes Compiler. *http://hal3.name/HBC*, 2007.

[10] T. Yano, W. Cohen, and N. Smith. Predicting response to political blog posts with topic models. In *NAACL*, pages 477–485, 2009.

[11] A. Ritter, Mausam, and O. Etzioni. A latent dirichlet allocation method for selectional preferences. In *ACL*.

[12] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum. Church: a language for generative models. In *UAI*, 2008.

[13] D. Wingate, A. Stuhlmüller, and N. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *AIStats*.

[14] J. Winn. Infer.NET and CSOFT. In *NIPS*, 2008.

[15] L. Dietz. *Exploiting Graph-Structured Data in Generative Probabilistic Models*. PhD thesis, Max-Planck-Institut für Informatik, 2011.

[16] J. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11, August 2010.

[17] A. McCallum. MALLET: A Machine Learning for Language Toolkit. http://mallet.cs.umass.edu, 2002.

[18] J. Hagood. A brief introduction to data mining projects in the humanities. *Bulletin of the American Society for Information Science and Technology*, 38(4):20–23, 2012.

[19] C. Napoles, M. Gormley, and B. Van Durme. Annotated Gigaword. In *AKBC*, 2012.

[20] N. Chambers. Event schema induction with a probabilistic entity-driven model. In *EMNLP*, 2013.