Implementing Latent Dirichlet Allocation with Stochastic Variational Inference

Kevin Song

December 7, 2016

1 Introduction

When analyzing a large body of text, we are very rarely interested in what happens at the level of individual words or sentences. Instead, we want to learn some higher-level information about the body of documents. For example, we may want to know what the prevailing opinion is on a certain topic (e.g. analyzing legal documents), we may want to extract some data held in the documents (e.g. analyzing medical literature for treatment outcomes), or, in some cases, when we are handed a stack of documents with no further information, we might just want to know what the documents are talking about.

Topic models, introduced around the turn of the millennium, are a mechanism for solving the last problem: without any additional information about the corpus, aside from maybe the number of topics expected, it can determine which words belong to which topic, and what topic (or set of topics) each document exhibits. Examples of topic models include probabilistic latent semantic indexing (pLSI) and latent dirichlet allocation (LDA).

In this project, I write a small-scale LDA inference program using stochastic variational inference (SVI). This program is applied to a set of test documents which are generated by the rules of the LDA model (explained below). Unfortunately, due to time constraints, I was not able to optimize my code well enough to attempt to feed all of Wikipedia as input, as I had originally planned.

The source code for this project can be found at `https://github.com/chipbuster/SDS385/tree/master/exercises/finalproject`

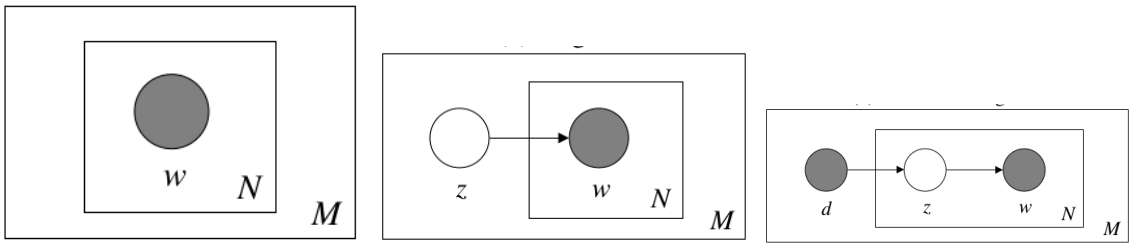
2 Methods Overview

A note on notation: the papers I'm working out of (cited at the end of this document) have horrible notational issues.

2.1 Topic Models and LDA

Topic models tend to assume a generative process, meaning the words in a document are drawn at random according to certain rules. For example, we may assume that there are two topics, that each document exhibits exactly one topic, and the words are drawn from that topic distribution. Solving the topic model involves figuring out what the parameters of the model are, in this case, what the word distribution for each topic is, and which (single) topic each document has.

The challenge is to create a model which is strong enough to make interesting predictions about a document corpus, but can still be solved. To illustrate this, I briefly explore unigram models, the pLSI model, and the LDA model.



(a) Unigram Model (b) Mixture of unigrams (c) pLSI

Figure 1: Graphical representations for topic models

2.1.1 Unigram Model

In the unigram model, there is one global distribution of words, and all words in every document are assumed to be sampled from this distribution. This is not a particularly interesting model, since it essentially means that every document should look the same.

Mathematically, this manifests as the following conditional for the words:

$$p(w) = \prod_{n=1}^{N} p(w_n)$$

We can graphically represent this as simply observing words coming from a distribution, as seen in Figure 1(a).

2.1.2 Mixture of Unigrams

In the mixture of unigrams, each document has its own multinomial distribution. These distributions do not represent topics–they are just a document-specific distribution of words. The generative process draws a parameter z for each document, and then words are drawn from a multinomial conditioned on z. Mathematically, this is represented by

$$p(w) = \sum_{z} p(z) \prod_{n=1}^{N} p(w_n|z)$$

Graphically, this is represented by the process shown in Figure 1(b)

2.1.3 Probabilistic Latent Semantic Indexing

pLSI extends the mixture of unigrams model further, by postulating that a document d's words are independent of the document index given an unobserved topic, z. It is represented graphically by Figure 1(c).

2.1.4 Latent Dirichlet Allocation

Latent Dirichlet Allocation extends all of the previous models by assuming that there is some fixed global parameter which controls the topic distributions of the documents. There is then a per-topic probability of a word showing up in a document, which is modified by another global parameter. Graphically, it is represented by Figure 2.

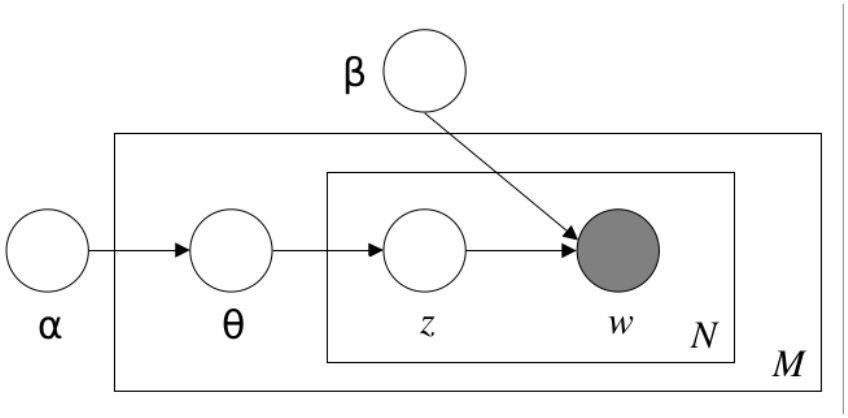


Figure 2: Graphical Model for LDA

Algorithimically, the generative process is assumed to draw topic distributions from a Dirichlet distribution. Then, for each individual document in the corpus, the following happens:

1. Choose the document length N, either as a fixed parameter or from a Poisson distribution.
2. Choose a document topic composition $\theta \sim \text{Dir}(\alpha)$
3. For each of the N words:
   (a) Choose a topic $z_n \sim \text{Multinomial}(\theta, 1)$
   (b) Choose a word w_n from $p(w_n|z_n, \beta_{z_n})$

As can be seen from Figure 2, this model is significantly more complicated than any of its predecessors. What it loses in simplicity, however, it more than makes up for in power. On top of being able to account for a probabilistic distribution of words in a topic, LDA is additionally posits that a document can contain multiple topics, and that there is some global distribution of topics (controlled by α) which the topic mixtures come from.

The question that arises now is this: can we efficiently solve this model? For a long time, the only available solution method was batch variational inference, which is very slow when the number of documents and topics is large. Instead, a more recent invention, stochastic variational inference, can be used to quickly calculate the posterior distribution.

2.2 Stochastic Variational Inference

Variational inference is one technique that can be used when a posterior distribution is intractable to compute explicitly. By introducing some parametrized distribution over the hidden variables and minimizing the difference between the variational distribution and the posterior, we can approximate the posterior distribution.

Variational inference minimizes the evidence lower bound (ELBO), which is equal to the negative Kullback-Leibler divergence plus an additive constant.

To derive the ELBO, we assume we have a probability distribution over all variable in the model (in this example case, x, z, β), and introduce a distribution over some hidden variables, $q(z, \beta)$.

$$
\begin{aligned}
\log p(x) &= \log \int p(x, z, \beta) dz d\beta \\
&= \log \int p(x, z, \beta) \frac{q(z, \beta)}{q(z, \beta)} dz d\beta \\
&= \log \left(\mathbb{E}_q \left[\frac{p(x, z, \beta)}{q(z, \beta)} \right] \right) \\
&\geq \mathbb{E}_q[\log p(x, z, \beta)] - \mathbb{E}_q[\log q(z, \beta)] \\
&\triangleq L(q)
\end{aligned}
$$

By making appropriate choices for the forms of q and modifying the gradient of the PDF by the Fisher matrix (which results in the so-called natural gradient), we can arrive at the general algorithm for stochastic variational inference in Appendix A.

The specific variational inference algorithm for LDA is provided in Listing 1. For ease of understanding, some of the types and sizes of parameters have been added to the algorithm. The numbers used for dimensions are as follows: K is the number of topics, N is the number of distinct words in the document corpus, D is the number of documents, and M is the number of words in a given document. Superscripts in parenthesis are used to refer to the time series, while superscripts without parenthesis are used to refer to a document ID, e.g. w_n^d is the nth word in the dth document. This differs slightly from the original notation because, quite frankly, I wouldn't wish the original notation on my enemies.

In addition, words are represented indicator vectors, that is w_n^d is a vector with N entries, all but one of which are zero. When used as a subscript, it refers to the nonzero index in the indicator vector, instead of the full vector.

If a matrix-valued variable is subscripted with a single subscript, it refers to that column of the matrix, e.g. if X is a matrix, X_i is the ith column of X.

Alpha and eta (α, η) are hyperparameters provided by the user. A sane value is somewhere between 0.1 and 1.

Finally, the variable Ψ is used to refer to the digamma function, which is equal to the derivative of the log of the gamma function:

$$
\Psi(x) = \frac{d}{dx} \log \left(\Gamma(x) \right)
$$

```
Declare λ: a matrix of size k × n
Set ρ^(0) in [0.5, 1)
while λ not converged:
  while φ and γ not converged:
    Sample a document indexed by d randomly from the corpus
    Declare γ^d: a vector with K elements
    Declare φ^d: a matrix of size k × M

    Initialize γ_k^d = 1 for k ∈ {1, ..., K}
    For m ∈ {1, ..., M}:          (words in document)
      For k ∈ 1, ..., K:          (topics)
        Set φ_{m,k}^d ∝ (Ψ(γ_k^d) − Σ_{j=1}^K Ψ(γ_j^d)) − (Ψ(λ_{k,w_n^d}) − Σ_{j=1}^M Ψ(λ_{k,j}))

    Set γ^d = α + Σ_m φ_m^d
  endwhile

  For k ∈ {1, ..., K}:
    Set intermediate topics λ̂_k = η + D Σ_{m=1}^M φ_{mk}^d w_{dm}
  endfor
```

```
21    Update λ^(t) = (1 − ρ^(t))λ^(t−1) + ρ^(t) λ̂
      Update ρ^(t) by some rule, e.g. Robbins Munro or Adagrad
23  endwhile
```

Listing 1: The SVI-LDA algorithm

The per-document parameters γ and ϕ can either be stored during computation and reused, or the global parameter λ can be used to reconstruct them once it is found, using a procedure similar to that between lines 5 and 15 in Listing 1. Once ϕ, γ, and λ are known for a particular document and corpus, they can be used to extract information about the topics in each document and the per-topic distribution of words.

3 Code & Results

Unfortunately, due to time constraints, I was unable to get my SVI running at scale on a large body of documents. Instead, I created a corpus of 10,000 documents from four topics, with about 250 words per topic, using the generative algorithm described in Section 2.1.4. The code to do this is found in `generative.py`, with the topics found in `wordlist1.txt` through `wordlist4.txt`. The results are in the directory `testdocs`. `lda.py` contains the code for doing a simple LDA analysis on the test corpus.

3.1 Speed Results

While getting the lambdas to converge can be done fairly quickly (in approximately half an hour), calculating the per-document γ and ϕ distributions is currently surprisingly expensive. Clearly, this is not code that will run in any reasonable amount of time on 700,000 documents with 7,000 topics.

I suspect that there are three major performance issues with this code:

1. Lack of sparsity

   While λ, γ, and ϕ could reasonably be expected to be dense, the individual updates to them might not be: in particular, since updates to ϕ are proportional to an exponential, if one entry of ϕ were significantly larger than all others, it would be reasonable to expect that the update can be approximated as a sparse indicator vector.

2. Lack of algorithm-specific optimizations

   In accordance with the above point, I don't know if it's a valid thing to always approximate the ϕ update as an indicator vector, or if, in some rare cases, there will be two phi entries with almost-identical magnitudes, in which case the approximation will break down.

   Another such issue arises with the calculation of $\hat{\lambda}_k$. In the naive computation, the elements of $\hat{\lambda}_k$ are iterated over in no particular order, trashing cache locality. Would it be possible to sort the words in the document first and then iterate over them in-order to improve cache locality? Would this make a difference? Without explicitly testing tweaks like these, it is difficult to say, and unfortunately, I ran out of time for testing.

3. No C++ code

   A lot of computation is deeply nested (e.g. calculating the expectations, which requires calling a function, which calls `digamma`, which calls `gamma`, which calls some lower level functions...) or heavily reliant on tight loops (e.g. the loops over the number of topics to update ϕ), both of which are relatively expensive in Python compared to compiled languages like C++. In addition, Eigen offers finer control over sparse linear algebra than numpy/scipy do, potentially increasing sparse performance further.

	WL1	WL2	WL3	WL4
LDA Topic 1	0	0	1	0
LDA Topic 2	0.039	0.961	0	0.019
LDA Topic 3	0.964	0	0.035	0
LDA Topic 4	0.571	0	0	0.429

Table 1: Fraction of words from each wordlist in LDA topics.

3.2 Qualitative Results

Unfortunately, due to time constraints, I was unable to get my SVI running at scale on a large body of documents. Instead, I gathered four wordlists from an english-language-learner's website that had topics, curated them (to avoid compound words and words with punctuation), and used the algorithm on page 3 to generate 10,000 sample documents of 1000 words each. The generative code can be found in `generative.py`, while the topics are found in the `wordlist#.txt` files. The topics contained in each file are listed below, along with examples of words that might come from each.

- `wordlist1.txt`: Food and Cooking Examples:
  - tamale
  - lard
  - sustenance
- `wordlist2.txt`: Tools Examples:
  - adze
  - scythe
  - sharpener
- `wordlist3.txt`: Halloween Examples:
  - cauldron
  - kimono
  - fog
- `wordlist4.txt`: Boats and the Ocean Examples:
  - warship
  - pennant
  - stow

So how well did LDA do? Figure 3 shows samples of the topics generated by LDA. We can see that the upper-left topic corresponds fairly well to the Halloween topic, the upper-right to Tools, the lower-left to Food and Cooking, and the lower-right to some combination of the third and fourth wordlists. This is not particularly surprising, given that the LDA only looked at about 300 documents before converging. If I had forced it to look at all 10,000 documents before allowing $\rho^{(t)}$ to decrease, it's very likely that the fourth topic would have been much more homogeneous.

To test this quantitatively, I checked what percent of words in each LDA topic came from each wordlist. The results are shown in Table 1.

Clearly, the first 3 topics selected by LDA are fairly pure, and are almost all from the same topic, with $> 96\%$ concentration in one topic. They also agree with the qualitative assessment above, e.g. LDA Topic

Figure 3: LDA topic results

1 comes entirely from wordlist 3, which is the Halloween topic. The 4th LDA topic suffers from the issues mentioned above as well, being split almost 50-50 between food and boats.

4 Conclusion

For this project, I wrote a program to calculate the posterior for the Latent Dirichlet Allocation topic model using Stochastic Variational Inference. The program runs decently quickly on smaller data sets, but not nearly fast enough to scale to corpuses like the New York Times or Wikipedia. For this reason, and to test correctness, I generated a smaller test set using the presumed generative model of LDA, and showed that the program performed as-predicted on this smaller set.

References

[1] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[2] Matthew D. Hoffman, David M. Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

5 Appendix A: General Stochastic Variational Inference

This algorithm is taken near-verbatim out of Hoffman et. al., and so uses some of their curious notational conventions.

Initialize $\lambda^{(0)}$ randomly
Set ρ_t
repeat until converged:
  Sample a data point x_i from the data set
  Compute its local variational parameter:
    $\phi = \mathbb{E}_{\lambda^{(t-1)}}[\eta_g(x_i^{(N)}, z_i^{(N)})]$

  Computer intermediate global parameters:
    $\hat{\lambda} = \mathbb{E}_{\phi}[\eta_g(x_i^{(N)}, z_i^{(N)})]$

  Update estimates of lambda:
    $\lambda^{(t)} = (1 - \rho_t)\lambda^{(t-1)} + \rho_t \hat{\lambda}$

  Update ρ_t, e.g. using a Robbins Munro schedule
end