Variational Inference on Latent Dirichlet Allocation

Yunran Chen
Department of Statistical Science
Duke University
yunran.chen@duke.edu

Junwen Huang
Department of Statistical Science
Duke University
junwen.huang@duke.edu

Abstract

One of the core problems of modern statistics is to ***approximate hard-to-compute probability densities****. This problem is especially important in Bayesian statistics for posterior density approximate under complex hierarchical structure setting (the one that includes latent variables).* ***Variational Inference (VI)*** *is a method that approximates probability densities through optimization. VI is faster and easier to scale to large data compared to classical methods, such as MCMC sampling. Here we implemented variantional EM algorithm on Latent Dirichlet allocation (LDA). Latent Dirichlet allocation (LDA) is a three-level hierarchical Bayesian model for collections of discrete data, such as text corpora. We present Variational EM on LDA model based on Blei* et al.*,2003 [2]. We present detailed introduction and proof* [1]*, detailed psudocode, and a package (VIonLDA)*[2]*. We also present the algorithm optimization based on vectorizaiton version which can be extended to not only one-hot-coding matrix. Then we provide an application on a real dataset and get reasonable result. In addition, we present detailed VI algorithm for smoothing LDA. At last, we provide a detailed discussion on ideal data structure.*

Keywords: *probability density approximation, Bayesian hierarchical model, Variational EM, Latent Dirichelet Allocation*

1. Introduction

We reproduce the paper Latent Dirichlet Allocation (LDA) by Blei *et al*. [2]. The reason for choosing this partly comes from interests on text mining. Furthermore, algorithm for efficient approximate for intractable posterior distribution inference and parameter estimation is super useful, which should be inside the statistician's toolbox. Here follows a brief introduction of variational inference (VI), including the basic problems it address, the possible applications, its advantages and disadvantages, the key idea and mathematical background. [1] Then we would give a brief introduction to LDA, including the main goal, advantages and mathematics setting.[2] In section 2, we introduce a thorough illustration on how to apply VI on LDA model and provide corresponding pseudocode. Notice there are typos in the appendix and pseudocode given in the orginal paper.[2]. In section 3, we generate data from LDA. Based on simulation data, we compare the performance of optimization. We tried optimization on data structure, vectorization, JIT and parallel. Then we would compare our best performance algorithm (vectorizaiton

[1] including point out an typo in the original paper.
[2] https://github.com/YunranChen/VIonLDA.git

version) with lda package in sklearn. In section 4, we applied the algorithm to simulated data sets, showing the property for this algorithm. In section 5, we applied the algorithm to real datasets and get reasonable result.

1.1. Variational Inference

One of the core problems of modern statistics is to approximate difficult-to-compute probability densities. This problem is especially important in Bayesian statistics when posterior is not easy to compute under complex hierarchical structure including latent variables.

Generally, consider a joint density of latent variables $\mathbf{z} = z_{1:m}$ and observations $\mathbf{x} = x_{1:n}$.

$$p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$$

In complex Bayesian models, the computation for posterior $p(\mathbf{z}|\mathbf{x})$ often requires approximate inference. The main advantage of VI is efficiency. For large data sets or very complex models, VI is much faster than a simple MCMC algorithm. For mixture models where Gibbs sampling is not an option, VI may perform better than a more general MCMC technique. However, VI is not yet well understood. VI generally underestimates the variance of the posterior density. The relative accuracy of VI and MCMC is still unknown. Several lines of empirical research have shown that VI does not necessarily suffer in accuracy. Still, VI is a valuable tool especially for Bayesian.

The goal of VI is to approximate a conditional density of latent variables given observed variables. The key idea is to use **optimization**.

First, posit a family of approximate densities $\mathcal{L}$. This is a set of densities over the latent variables. The choice of $\mathcal{L}$ should be flexible enough meanwhile simple enough for efficient optimization. Then, find the member of that family that minimizes the Kullback-Leibler(KL) divergence to the exact posterior.

$$q^*(\mathbf{z}) = argmin_{q(\mathbf{z})\in\mathcal{L})} KL(q(\mathbf{z})||p(\mathbf{z}|x))$$

where

$$KL(q(\mathbf{z})||p(\mathbf{z}|x)) = E_q[logq(\mathbf{z})] - E_q[logp(\mathbf{z}|\mathbf{x})]$$

Because we cannot compute the KL, we optimize an alternative objective that is equivalent to KL. We call this the evidence lower bound $ELBO(q)$. Maximizing the $ELBO$ is equivalent to minimizing the KL divergence.

$$ELBO(q) = E_q[logp(\mathbf{z}, \mathbf{x})] - E_q[logq(\mathbf{z})]$$

Finally, approximate the posterior with the optimized member of the family $q^*(\cdot)$.

The variational algorithm transfer the density approximate to an optimization problem. Here we could apply the algorithm for optimization, such as gradient descent, Newton-Raphson method.

We would show the details for applying VI with the example of LDA model.

1.2. Lante Dirichlet Allocation

The main goal for LDA is to model collections of discrete data with proper underlying generative probabilistic semantics and dimensionality reduction. Based on the results from LDA model, we could achieve some basic text

mining such as classification, novelty detection, similarity and relevance judgment and so on.

Basically, LDA is a three-level hierarchical Bayesian model. Due to its Hierarchical structure, it can achieve:

1. dimensionality reduction (using parameter with lower dimension);
2. proper underlying generative probabilistic semantics (hierarchical structure);
3. generalizes easily to new documents (Dirichlet prior to allow positive probability).

Here we define the following terms:

1. A word is the basic unit of discrete data, with one-hot encoding. Specifically, w is a $V \times 1$ vector such that $w^v = 1$ if the word is exactly v and $w^u = 0$ for $u \neq v$.
2. A document is a $V \times N$ matrix $\mathbf{w} = (w_1, w_2, ..., w_N)$ with the word vector as each column.
3. A corpus is a collection of M documents denoted by $D = \{\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_M}\}$
4. A topic is a $k \times 1$ vector such that $z^t = 1$ if a word is considered to be topic t and $z^s = 0$ for $s \neq t$.

LDA assumes the following generative process for each document $\mathbf{w}$ in a corpus D:

1. Choose $N \sim Poisson\{\xi\}$
2. Choose $\theta \sim Dir(\alpha)$
3. For each of the N words w_n:

   Choose a topic $z_n \sim Multinomial(\theta)$

   Choose a word w_n from $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic z_n

Here follows simplifying assumptions:

1. Words and documents are exchangeable. (i.e. conditional i.i.d.)
2. Dimensionality k of the Dirichlet distribution is assumed known and fixed.
3. $w \sim Multinomial(\beta^T z)$, where β is $k \times V$ matrix $\beta_{ij} = p(w^j = 1|z^i = 1)$.
4. β and α are fixed quantities need to be estimated.
5. Poisson assumption is not critical to anything. Here we consider N as constant instead of random variable.

Given the parameters α and β, the joint distribution of a topic mixture θ, a set of N topics $\mathbf{z}$, and a set of N words $\mathbf{w}$ is given by:

$$p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N} p(z_n|\theta) p(w_n|z_n, \beta) \tag{1}$$

The marginal distribution of a document is:

$$p(\mathbf{w}|\alpha, \beta) = \int p(\theta|\alpha) (\prod_{n=1}^{N} \sum_{z_n} p(z_n|\theta) p(w_n|z_n, \beta)) d\theta \tag{2}$$

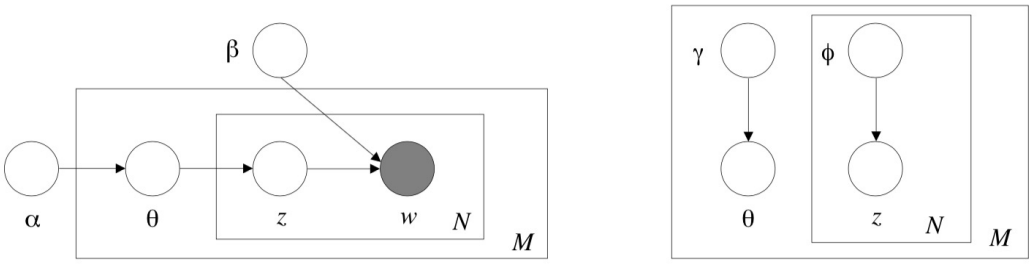

Figure 1. Variational Inference Graph on LDA

The probability of a corpus:

$$p(D|\alpha,\beta) = \prod_{d=1}^{M} \int p(\theta_d|\alpha)(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d)p(w_{dn}|z_{dn},\beta))d\theta_d \tag{3}$$

The hierarchical structure of LDA is shown in figure **??**. The parameters α and β are corpus-level parameters, assumed to be sampled once in the process of generating a corpus. The variables θ_d are document-level variables, sampled once per document. Variables z_{dn} and w_{dn} are word-level variables and are sampled once for each word in each document.

1.3. Inference and Parameter Estimation

The key inferential problem is computing posterior distribution:

$$p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta) = \frac{p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta)}{p(\mathbf{w}|\alpha, \beta)} \tag{4}$$

Due to the coupling between θ and β in the summation over latent topics, $p(\mathbf{w}|\alpha, \beta)$ is intractable.

$$p(\mathbf{w}|\alpha,\beta) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \int (\prod_{i=1}^{k} \theta_i^{\alpha_i - 1})(\prod_{n=1}^{N} \sum_{i=1}^{k} \prod_{j=1}^{V} (\theta_i \beta_{ij})^{w_n^j}) d\theta \tag{5}$$

However, due to the intractable posterior distribution, a wide variety of approximate inference algorithms can be considered, including Laplace approximation, variational approximation, and Markov chain Monte Carlo (Jordan, 1999). Here we mainly implemented variational inference. Specifically, we applied convexity-based variational algorithm by Blei *et al*. [2]. For parameters estimation, we implemented variational EM algorithm, inside also including Newton-Raphson Method. More details see section 2.

2. Variational Inference on LDA

First, we need to posit a tractable family. Second, we transfer it to an optimization problem to maximizing the $ELBO$. Third, we find the optimal variational parameters. Last, we approximate the posterior with the optimized parameters and do parameter estimation.

2.1. Variation Inference

To obtain a tractable family of lower bounds, consider dropping the nodes w,β, and the edge between θ and z. See figure 1. Then we get the family on latent variables, characterized by free variational parameters:

$$q(\theta, \mathbf{z}|\gamma, \phi) = q(\theta|\gamma) \prod_{n=1}^{N} q(z_n|\phi_n) \tag{6}$$

Where $\theta \sim Dir(\gamma)$, $z_n \sim Multinomial(\phi_n)$.

Here we set up the optimization problem by minimizing the Kullback-Leibler(KL) divergence between the variational distribution and the true posterior $p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta)$

$$(\gamma^*, \phi^*) = argmin_{(\gamma,\phi)} D(q(\theta, \mathbf{z}|\gamma, \phi)||p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta))$$

which equals to maximizing the evidence lower bound $ELBO$

$$\text{ELBO}(\gamma, \phi; \alpha, \beta) = E_q[logp(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta)] - E_q[logq(\theta, \mathbf{z}|\gamma, \phi)] \tag{7}$$

Notice, the ELBO can also be derived from Jensens inequality ($log(Ex) \geq E(log(x))$) on the log likelihood $logp(\mathbf{w}|\alpha, \beta)$.

To get optimal ϕ^*_{ni}, consider the item only include ϕ_{ni} in ELBO and add restriction $\sum_{i=1}^{k} \phi_{ni} = 1$. We set up a lagrangian:

$$ELBO_{[\phi_{ni}]} + \lambda_n(\sum_{i=1}^{k} \phi_{ni} - 1)$$

Get the derivative with respect to ϕ_{ni}, we have

$$\phi_{ni} \propto \beta_{iv} exp(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k} \gamma_j)) \tag{8}$$

where $\sum_{i=1}^{k} \phi_{ni} = 1$,$\Psi$ is the first derivative of the $log\Gamma$ which is computable via Taylor approximations.

To get optimal γ^*_i, consider the item only include γ_i, which denote as $ELBO_{[\gamma_i]}$. Get derivative of it and set it to zero, we have

$$\gamma_i = \alpha_i + \sum_{n=1}^{N} \phi_{ni} \tag{9}$$

Applied iterative fixed-point method, we get optimizing variational parameters $(\gamma^*(\mathbf{w}), \phi^*(\mathbf{w}))$ given a document and related density estimation

$$p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta) \approx q(\theta, \mathbf{z}|\gamma^*, \phi^*)$$

2.2. Variational EM

To get parameter estimation α^*, β^*, we need to maximize the log likelihood of the data:

$$L(\alpha, \beta) = \sum_{d=1}^{M} log(\mathbf{w}_d|\alpha, \beta)$$

which is equal to maximize the summation of equation (7):

$$L(\alpha, \beta) = \sum_{d=1}^{M} \text{ELBO}(\gamma_d, \phi_d; \alpha, \beta) \tag{10}$$

To get the optimal β, consider the item only include β in $L(\alpha, \beta)$ and add restriction $\sum_{j=1}^{V} \beta_{ij} = 1$. We set up a lagrangian:

$$L_{[\beta]} + \sum_{i=1}^{k} \lambda_i (\sum_{j=1}^{V} \beta_{ij} - 1)$$

Get the derivative with respect to β, we have

$$\beta_{ij} \propto \sum_{d=1}^{M} \sum_{n=1}^{N_d} \phi_{dni} w_{dn}^{j} \tag{11}$$

where $\sum_{j=1}^{V} \beta_{ij} = 1$.

To get optimal α, consider the item only include α, which denote as $L_{[\alpha]}$. Get derivative of it and set it to zero, we have

$$g_i = \frac{\partial L}{\partial \alpha_i} = M(\Psi(\sum_{j=1}^{k} \alpha_j) - \Psi(\alpha_i)) + \sum_{d=1}^{M} (\Psi(\gamma_{di}) - \Psi(\sum_{j=1}^{k} \gamma_{dj})) \tag{12}$$

Hessian :

$$H_{ij} = \frac{\partial L}{\partial \alpha_i \alpha_j} = -M\Psi'(\alpha_i)\delta(i,j) + M\Psi'(\sum_{j=1}^{k} \alpha_j) \tag{13}$$

where $\delta(i,j) = 1$ when$j = i$, otherwise $\delta(i,j) = 0$. Note that the appendix of the paper gives a wrong derivation of the Hessian and we have offered a corrected version above.

The derivative depends on α, we must use an iterative method to find the optimal α.

Due to the special form for Hessian matrix

$$H = diag(h) + 1z1^T$$

, we could apply **linear-time Newton-Raphson algorithm**.

$$\begin{aligned} \alpha_i^{t+1} &= \alpha_i^t - (H(\alpha^t)^{-1} g(\alpha^t))_i \\ &= \alpha_i^t - \frac{g_i^t - c^t}{h_i^t} \end{aligned}$$

where $c^t = \frac{\sum_{j=1}^k g_j^t / h_j^t}{(z^t)^{-1} + \sum_{j=1}^k (h_j^t)^{-1}}$

With

$$z^t = M\Psi'(\sum_{j=1}^k \alpha_j^t)$$

$$h_i^t = -M\Psi'(\alpha_i^t)$$

Notice, there is a typo in the appendix in the original paper on equation 13.

2.3. Pseudocode

The original paper does not provide the initialization for α, β. Here we use random variable as our initialization. [3] Notice there is a typo in original paper. We made correction.

Input: a set of words in documents $D = \{\mathbf{w_1}, ..., \mathbf{w_M}\}$, number of topics k, initialization parameters
where $\mathrm{w}_1 is \mathrm{N}_d \times V$ one-hot-encoding matrix

Output: Variational parameters ϕ, γ, parameters α,β

Initialize: $\beta_k^0 \sim Dirichlet(\mathbf{1})$;
$\alpha_k^0 \sim Gamma(shape, scale)$;
$\phi_{ni}^0 := 1/k$ for all i and n;
$\gamma_i^0 := \alpha_i + N/k$ for all i

while *the ELBO has not converged* **do**
  repeat
    for *each document* $d = 1$ *to* M **do**
      for *each word* $n = 1$ *to* N_d **do**
        for *each topic* $i = 1$ *to* k **do**
          $\phi_{ni}^{t+1} := \beta_{iw_n}^t exp(\Psi(\gamma_i^t) - \Psi(\sum_{j=1}^k \gamma_j^t))$
          normalize ϕ_n^{t+1} to sum to 1.
        end
        $\gamma_i^{t+1} := \alpha_i^t + \sum_{n=1}^N \phi_{ni}^{t+1}$
      end
    end
  until ϕ,γ *have converged*;
  for *each vocabulary* $j = 1$ *to* V **do**
    $\beta_{ij}^{t+1} := \sum_{d=1}^M \sum_{n=1}^{N_d} \phi_{dni}^{t+1} w_{dn}^j$
    normalize β_{ij}^{t+1} to sum to 1.
  end
  $z^{t+1} = M\Psi'(\sum_{j=1}^k \alpha_j^t)$
  $h^{t+1} = -M\Psi'(\alpha^t)$
  $g^{t+1} = M(\Psi(\sum_{j=1}^k \alpha_j^t) - \Psi(\alpha^t)) + \sum_{d=1}^M (\Psi(\gamma_d^{t+1}) - \Psi(\sum_{j=1}^k \gamma_{dj}^{t+1}))$
  $c^{t+1} = \frac{\sum_{j=1}^k g_j^{t+1}/h^t}{(z^{t+1})^{-1} + \sum_{j=1}^k (h_j^{t+1})^{-1}}$
  $\alpha^{t+1} = \alpha^t - \frac{g^{t+1} - c^{t+1}}{h^{t+1}}$
end

Algorithm 1: Pseudocode for LDA using VI

[3] https://en.wikipedia.org/wiki/Concentration_parameter

3. Optimization for Performance

3.1. Simulated Datasets

We simulated data as follows. We set $M = 500$,$k = 10$,$V = 1000$,$\xi = 40$,$a = 2$,$b = 1$ with seed 123. [4]

Input: M, k, V, ξ , shape a and scale b of the gamma distribution
Output: List of documents (docs), α, β
Sample $\alpha \sim \Gamma(a, b)$
Sample $\beta \sim$ Dir($\mathbf{1}_V$)
Sample $N \sim$ Poisson(ξ)
Sample $\theta \sim$ Dir(α)
for *each document* $d = 1$ *to* M **do**
  for *each word* $n = 1$ *to* N_d **do**
    Sample a topic $z_{dn} \sim$ Multinomial(θ_d)
    Sample a word $w_{dn} \sim$ Multinomial($z_{dn}\beta$)
  end
end

Algorithm 2: Pseudocode for data simulation

3.2. Optimization for Performance

Here the default of the algorithm is set as:

```
M_step(docs=docs,k=10,tol=1e-3,tol_estep=1e-3,max_iter=100,
initial_alpha_shape=100,initial_alpha_scale=0.01)
```

Plain Version: We first implement plain Python in straightforward way from the description of the algorithm. For the plain version, we take $V = 100$ instead of $V = 1000$. But still it takes about half an hour to run.

Vectorization: We optimize the algorithm with vectorization. To be more specific, we replace most of the for-loops with matrix operation and broadcast. There would be no for-loop inside each E-step: For each document, we only use matrix operation and broadcast to update ϕ and γ. This version use for loop only when iterating through documents: In detail, we replace the nested for loop in Algorithm 3 with matrix multiplication as shown below. **In addition, the input of can be extended, not only one-hot coding matrix.** Each document could be represented as a $N'_d \times V$ matrix, where N'_d is the number of unique words. Each row has only one non-zero entry indicate the count of word n in document d. **The calculation can be eased in this way.**

JIT and Cython: To make our code faster, we have tried Jit and Cython. However, their performance have not improved. A reasonable explanation is that all the operation inside the vectorized python versions are carried out through numpy, which is already based on C++. Also, JIT would not help for matrix multiplication and broadcast. Therefore, there is not much to do for Jit or Cython. In fact, using Jit or Cython here may even make the algorithm slower.

Parallel: We've also tried parallel. In our code, the only part can be paralleled is when implementing E-step for each document because for other parts the latter iteration relies on the result of the former iteration. So Variational EM algorithm is not suitable for parallel. As shown in Table 3.2. Here we can tell that the CPU time is much less. However, wall time is much more. This might be due to more time are spent to allocate and store resources

[4] Refer to function simulation_data in our package

Input: $\mathbf{w_d}$ is $N'_d \times V$ matrix, the nonzero entry is the count of word in that document.
while *the ELBO has not converged* **do**
  repeat
    for *each document* $d = 1$ *to* M **do**
      while ϕ, γ *has not converged* **do**
        repeat
          $\phi = (\mathbf{w_d}\beta^T) \times e^{(\Psi(\gamma)-\Psi(\sum\gamma))}$
          normalize ϕ to let row sum equal to 1.
          $\gamma = \alpha + \sum_k \Phi_{\cdot,k}$
        until ϕ, γ *have converged*;
      end
      β = β + ϕ^T γ
    end
    normalize β to let row sum equal to 1.
  until α,β *have converged*;
  update α in the same way as above
end

Algorithm 3: Pseudocode for the vectorized version

during the parallel. Notice We use package `ray`. Please `pip install ray` first before running out examples. Need to notice this version sometimes cannot be run in VM but can be run locally. Although parallel vesion is not suitable, we can use parallel for running vectorizaiton version several times with different initialization on the same datasets to get stable results.

The table 3.2 show the time for each version. We time Vectorization version for 10 times and get the mean of the time. For plain version, we set $V = 100$ instead of $V = 1000$ and run only once. For parallel version we run only once. Taking time efficiency and simplicity of code into consideration, **we chose Vectorization Version as our final version and take it for application.** All the versions can be found in our package **VIonLDA**. We also include detailed possible data structure improvement on discussion session.

Version	Plain Version	Vectorizaiton Version	Parallel Version
CPU total	31min25s	53.1s	11.8s
CPU user	31min18s	53.0s	7.21s
CPU sys	6.66s	40ms	4.59s
Wall time	31min38s	53.3s	2min34s

Table 1. Optimization for Performance.

4. Applications to Simulated Datasets

Here based on the simulation data above, we compare our result with the true parameters. Notice we use $MSE_\beta = \sum_{i=1}^{k}\sum_{j=1}^{V}(\beta_{ij}^{est} - \beta_{ij}^{true})^2$ to measure the performance for estimator. For α, we use $MSE_\alpha = \sum_{i=1}^{k}(\alpha_i^{*est} -$

$\alpha_i^{*true})^2$. Here $\alpha_i^* = \frac{\alpha_i}{\sum_{i=1}^k \alpha_i}$. Because for $\theta \sim Dir(\alpha)$,$E(\theta_i) = \frac{\alpha_i}{\sum_{i=1}^k \alpha_i}$. α is concentration parameters[5], we care more on the relative value instead of the scale. We applied Variational EM to the simulation dataset with the setting as follows (repeated 100 times):

`M_step(docs=docs,k=10,tol=1e-3,tol_estep=1e-3`

For reasonable guess, we set `initial_alpha_shape=100,initial_alpha_scale=0.01).`

For true initialization, we set the initialization of α the true value of α.

The result as follows:[6]

Initialization	Reasonable Guess Iteration=100	True Initialization Iteration=100	True Initialization Iteration=1000
α	7.197e-03$\pm 5.184e-04$	1.948e-03$\pm 4.613e-04$	4.139e-03$\pm 8.928e-04$
β	7.598e-06$\pm 3.433e-07$	8.471e-06$\pm 9.445e-07$	5.706e-06$\pm 5.111e-07$

Table 2. MSE for Parameters Estimation

Initialization	Reasonable Guess Iteration=100	True Initialization Iteration=100	True Initialization Iteration=1000
α	0.420	0.114	0.241
β	3.852	4.295	2.893

Table 3. Relative MMSE for Parameters Estimation

From the table 2 we can see the performance of parameter estimation highly depends on the initialization. The intrinsic property of EM algorithm is it cannot achieve the global optimal point. The initialization play an important role on its performance. With the true α as initialization, the MSE for α decrease significantly. Also, increasing iteration would bring improvement. Here we can see the estimation on β is not well performed. This may due to the initialization. We set $\beta^0 \sim Dir(\mathbf{1})$ (True $\beta \sim Dir(\mathbf{1})$). Also, it may due to the iteration is not enough for the algorithm to converge.

Actually, we only care about the rank statistics of α and β. Here we present a visualization on this . As figure 2 shows, the rank each row is relative similar. Notice we only set `max_iter=100`, the discrepancy may due to the algorithm has not yet converge. Actually, we applied the algorithm on real dataset, it performs well.

[5]https://en.wikipedia.org/wiki/Concentration_parameter

[6]We applied parallel for this part, it takes 12min 50s in total. Without parallel, it would take more than 100 minutes.

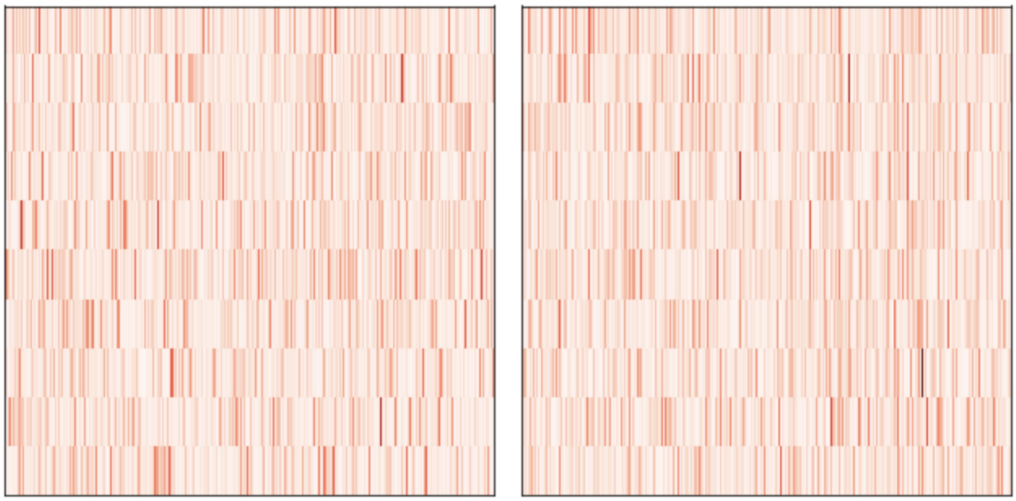

Figure 2. Comparison on BETA and estimated BETA.The left plot is true BETA. The right plot is estimated BETA.

5. Application to Real Datasets

Same as applying the algorithm to the simulated data set, we apply the algorithm to the real dataset mentioned in the paper and investigate the results returned. Here to avoid overflow problem, we've normalized γ in E-step by dividing each of its elements by the minimum value. So you can see from our Github that apart from the `Vectorization` version we also provide a `Realdata` version.

Our data are documents from the TREC AP corpus(Harman, 1992). After removing a standard list of stop words, numbers and punctuation, we used the Vectorization version of Variational EM algorithm descried above to find the Dirichelet and conditional multinomial parameters α and β for a 10-topic LDA model. Notice here we only take the first 500 documents as an illustration. We applied the algorithm to the dataset with the setting as follows:

```
M_step(docs=docs,k=10,tol=1e-3,tol_estep=1e-3,max_iter=100,
initial_alpha_shape=100,initial_alpha_scale=0.01)
```

For each of the topic, we look at β to get the top 50 words that is likely to be classified to be in that topic and decide what that topic should be. The topics and some of their potential component words are shown below:

	Economy	Legislation	Diplomacy	Finance	Domestic Politics
1	market	law	president	dollar	bush
2	prices	drug	agreement	bid	presidential
3	orders	court	soviet	bank	campaign
4	economy	rights	aid	yen	state
5	index	committee	south	london	democratic
6	rate	prison	foreign	million	national
7	bank	legislation	summit	trust	republican
8	business	attorney	iraq	gold	senate
9	average	abortion	peace	price	candidate
10	company	legal	statement	board	governor

Table 4. Topics and related words I

	Crime	Entertainment	Global Politics	Welfare	Military
1	police	new	government	air	government
2	area	rating	president	systems	military
3	killed	hunt	soviet	water	party
4	authorities	cancer(zodiac?)	black	energy	death
5	victims	art	chairman	public	defense
6	reported	fish	agreement	black	army
7	arrested	play	africa	help	soldiers
8	death	group	officials	community	leaders
9	angeles	like	spokesman	rights	soviet
10	church	heat	white	security	force

Table 5. Topics and related words II

It is obvious that our algorithm is doing a good job in terms of parameter estimation.

6. Comparative Analysis with Competing Algorithm

After verifying that our algorithm is doing a good job in terms of estimation, we compare the time efficiency of our algorithm with the function in `sklearn`. We use the dataset stated in section 3.1. The algorithm for `LatentDirichletAllocation` in `sklearn.decomposition` is not fully based on Blei *et al.*, 2003. It applied Latent Dirichlet Allocation with batch variational Bayes algorithm. It is expected that algorithms using batch will have better performance since in batch structure large repeated jobs are given to the system and we dont have to interact with computer to tell the system that you have to do that job after finishing that job. In other words, batch structure is super useful for large jobs are done in sequence by the system. We timed both algorithms and the results are shown below:

Version	Vectorization version	sklearn
CPU total	58s	16.3s
CPU user	40ms	40ms
CPU sys	58.1s	16.4s
Wall time	58.4s	16.5s

Table 6. Detailed Time Comparison with sklearn

Version	time(3 runs, 5 loops)
vectorized	58.2 s $\pm$ 950 ms per loop
sklearn	15.5 s $\pm$ 199 ms per loop

Table 7. Time Comparison with sklearn

7. Additional: Smoothing LDA

For the smoothing version of LDA, we give a prior on β. See graphical model firgure 3. Blei *et al*.,2003 does not provide much details in the paper. Here we provide brief idea, mathematical proof and detailed pseudocode.

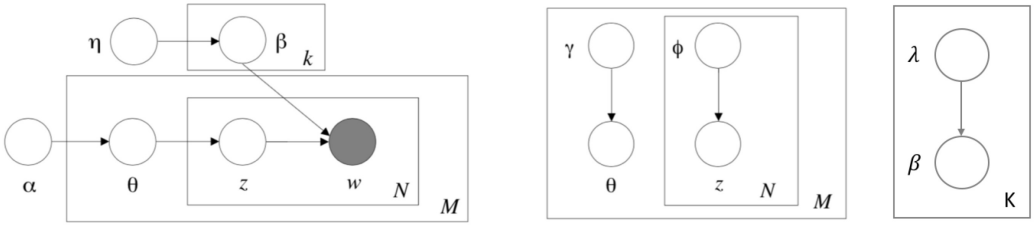

Figure 3. Graphical Model Representation of Smoothed LDA model

Similarly, compared to LDA model, we add one more variational parameter λ. So compared to equation 7, the optimization function becomes:

$$\begin{aligned}\text{ELBO}(\gamma,\phi,\lambda;\alpha,\eta) &= E_q[logp(\theta,\mathbf{z},\mathbf{w},\beta|\alpha,\eta)] - E_q[logq(\theta,\mathbf{z},\beta)] \\ &= E_q[logp(\theta|\alpha)] + E_q[logp(\mathbf{z}|\theta)] + E_q[logp(\mathbf{w}|\mathbf{z},\beta)] + E_q[logp(\beta|\eta)] \\ &\quad -E_q[log\prod_{i=1}^{k} q(\beta_i|\lambda_i)\prod_{d=1}^{M}(q(\theta_d|\gamma_d)\prod_{n=1}^{N_d} q(\mathbf{Z_{dn}}|\phi_{\mathbf{dn}}))]\end{aligned}$$

Similarly, we get the partial derivative with respect to ϕ,γ,λ,α,η. Notice the update of γ and α still remain the same as LDA update, for they do not have "direct" connection to β. So the update equation is as follows:

$$\phi_{dn,i} \propto exp(\Psi(\gamma_{di}) - \Psi(\sum_i \gamma_{di}) + \Psi(\lambda_{i,j}) - \Psi(\sum_{j=1}^{V} \lambda_{i,j})$$

$$\gamma_d = \alpha + \sum_{n=1}^{N_d} \phi_{dn}$$

$$\lambda_i = \eta + \sum_{d=1}^{M}\sum_{n=1}^{N_d}\phi_{dn}^i w_{dn}$$

To update α and η, we need to apply linear-time Newton-Raphson algorithm. The update α remains the same as LDA. Here we present the update for η.

$$\begin{aligned}\eta_j^{t+1} &= \eta_j^t - (H(\eta^t)^{-1}g(\eta^t))_j \\ &= \eta_j^t - \frac{g_j^t - c^t}{h_j^t}\end{aligned}$$

where $c^t = \frac{\sum_{j=1}^{V} g_j^t/h_j^t}{(z^t)^{-1}+\sum_{j=1}^{V}(h_j^t)^{-1}}$

With

$$z^t = K\Psi'(\sum_{j=1}^{V}\eta_j^t)$$

$$h_j^t = -K\Psi'(\eta_j^t)$$

$$g_j^t = K(\Psi(\sum_{j=1}^{V}\eta_j) - \Psi(\eta_j)) + \sum_{i=1}^{K}(\Psi(\lambda_{ij}) - \Psi(\sum_{j=1}^{V}\lambda_{ij}))$$

Here we provide the vectorization version for the pesudocode.

Input: a set of words in documents $D = \{\mathbf{w_1}, ..., \mathbf{w_M}\}$, number of topics k, initialization parameters

Output: Variational parameters ϕ, γ, parameters α,β

Initialize:$\eta_j^0 \sim Gamma(shape, scale)$;

$\alpha_k^0 \sim Gamma(shape, scale)$;

$\phi_{ni}^0 := 1/k$ for all i and n;

$\gamma_i^0 := \alpha_i + N/k$ for all i

$\lambda_i^0 \sim Gamma(shape, scale)$ for all i

while *the ELBO has not converged* **do**

  repeat

    for *each document* $d = 1$ *to* M **do**

      $\phi = (\mathbf{w_d}@(exp(\Psi(\lambda) - (\Psi(\sum_j \lambda)[:, None]))).T) * exp(\Psi(\gamma) - \Psi(\sum_i \gamma))$

      normalize $\sum_k \phi$ to 1

      $\gamma = \alpha + \sum_n \phi$

    end

  until ϕ,γ *have converged*;

  $\lambda = \sum_{d=1}^{M} \phi_d^T@\mathbf{w_d}$

  normalize $\sum_j \beta_{ij}$ to 1.

  update α, η

end

Algorithm 4: Pseudocode for Smoothing LDA Using VI

8. Conclusion

We applied Variational EM algorithm on LDA model. We achieved 1)Generate collections of documents according to LDA model. 2)Optimize the performance of the algorithm. We show that using a matrix multiplication, vectorization and broadcast would improve a lot. JIT and parallel would not contribute much to the algorithm optimization. However, if we want to repeat algorithm several times when testing on simulation datasets. Parallel (package `ray`) would be recommended. 3) We applied the algorithm to simulated data, showing the initialization matters a lot on Variational EM. 4) We applied algorithm to real dataset and get reasonable results. 5)We provide Variational EM on smoothing LDA in details. 6) All the versions of Variational EM are packaged. We also provide tests and examples on github. Notice the function with ray would not always work in VM. But it would work locally.

9. Discussion

More improvements can be done to algorithm performance. 1)The data structure. The highlight of our report is using matrix multiplication, vectorization and broadcast to reduce the for loop. However, in our vectorization version, each document is a $N'_d \times V$ matrix.[7] This would be a sparse matrix, each row has only 1 non-zero entry. Storing large matrix and sparse matrix multiplication would be inefficient. If we consider documents as $M \times V$ matrix, each document as a $V * 1$ vector, and design the algorithm on such data structure would improve a lot. 2)The algorithm. We could see from the pseudocode, each step depend on the result of last step. It is hard to achieve parallel. 3)Initialization. As we show in table 2, initialization really matters for the Variational EM algorithm can only achieve local optimization. Package `sklearn` outperform ours from the data structure and the algorithm, it use a batch VI bayes method or online VI bayes method, and take $M \times V$ as input.

More discussion on data structure improvement: In the Vectorization version, large computation comes from the multiplication between large matrix $\mathbf{w_d}$ and β when updating ϕ, and multiplication between large matrix $\mathbf{w_d}$ and ϕ when updating β. In addition, the matrix $\mathbf{w_d}$ is a $N'_d \times V$ sparse matrix, with only one nonzero entry each row. The matrix multiplication is "aggregation effect": "select the corresponding vocabulary from each document and aggregate them together". or "select the corresponding topic from each document and aggregate them together." So we can add the "same vocabulary effect" together among different documents to update the parameter ϕ and β. If we could use a vector instead of a sparse matrix to represent a document, and design algorithm on it. The computation would be reduced a lot. The ideal data structure for documents is $M \times V$ matrix. Each row is a vector indicating the count for vocabulary. The main obstacle for the algorithm comes from the asynchronous update of E-step and M-step. We need to update α and β based on optimal ϕ and γ. If we loose the condition, to update ϕ,γ,α and β synchronously, we could design our algorithm based on $M \times V$ matrix as input.

Here we just provide a small example on the efficiency of the vector compared to matrix multiplication. In the structure version, we use a $N_d \times 1$ vector instead of $N_d \times V$ matrix as input for E-step. We can see around 3 senconds improvement.

[7] N'_d is number of unique words

Version	time(3 runs, 5 loops)
Vectorization	58.2 s $\pm$ 950 ms per loop
Structure	54.6 s $\pm$ 458 ms per loop

Table 8. Time Comparison with Structure

10. Code

Github: https://github.com/YunranChen/VIonLDA

Pacakge: `pip install --index-url https://test.pypi.org/simple/ VIonLDA`

References

[1] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

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