Latent Dirichlet Allocation Implementation Report

Yang Bao, Wenlin Wu

April 25, 2018

Github Repository: https://github.com/yangbaovera/latent-dirichlet-allocation.git (https://github.com/yangbaovera/latent-dirichlet-allocation.git)

1 Abstract

Latent Dirichlet allocation(LDA) is a generative topic probabilistic model first proposed by David M.Blei, Andrew Y.Ng, and Michae l. Jordan to find latent topics in a text corpus.

This algorithm is a three-level hierarchical Bayesian model. It has three basic concepts: word, topic, and document. Actually, LDA mimics the method of human write an article and uses the process to reverse inference the distribution of topics when we have an article.Variational Inference is used for approximating intractable integrals arising in a Bayesian network. Therefore, in the inference steps, variational Inference can be seen as an extension of the EM algorithm which computes the entire posterior distribution of latent variables. In the meanwhile, LDA uses Newton-Raphson method and gradient descent to calculate the minimum of a lower bound. Also, the Hessian matrix is necessary for gradient descent procedures.

In this report, we mathematically described and analyzed the LDA model. And then implement all the core functions in LDA algorithm. Additionally, we used two different ways to optimize our model. One is from the algorithm's aspect, and the other is using JIT. Finally, a simulation data and real-world data were used to test the accuracy of our functions.

2 Background

2.1 Bayesian Model

$$prior + likelihood = posterior$$

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)}$$

2.2 Dirichlet distribution and Multinomial distribution

The distributions go like this:

$$multi(m_1, m_2, m_3 \mid n, p_1, p_2, p_3) = \frac{n!}{m_1!m_2!m_3!} p_1^{m_1} p_2^{m_2} p_3^{m_3}$$

$$Dirichlet(p_1, p_2, p_3 \mid \alpha_1, \alpha_2, \alpha_3) = \frac{\Gamma(\alpha_1 + \alpha_2 + \alpha_3)}{\Gamma(\alpha_1)\Gamma(\alpha_2)\Gamma(\alpha_3)} p_1^{\alpha_1 - 1} p_2^{\alpha_2 - 1} p_3^{\alpha_3 - 1}$$

For higher dimension, we have

$$Dirichlet(\vec{p} \mid \vec{\alpha}) = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} p_k^{\alpha_k - 1}$$

With the conjugate relationship, we have:

$$Dirichlet(\vec{p} \mid \vec{\alpha}) + Multi(\vec{m}) = Dirichlet(\vec{p} \mid \vec{\alpha} + \vec{m})$$

And the expectation of Dirichlet distribution is:

$$E(Diri(\vec{p} \mid \vec{\alpha}) = \left(\frac{\alpha_1}{\sum_{k=1}^{K} \alpha_k}, \frac{\alpha_2}{\sum_{k=1}^{K} \alpha_k}, \ldots \frac{\alpha_K}{\sum_{k=1}^{K} \alpha_k} \right)$$

3 LDA Model

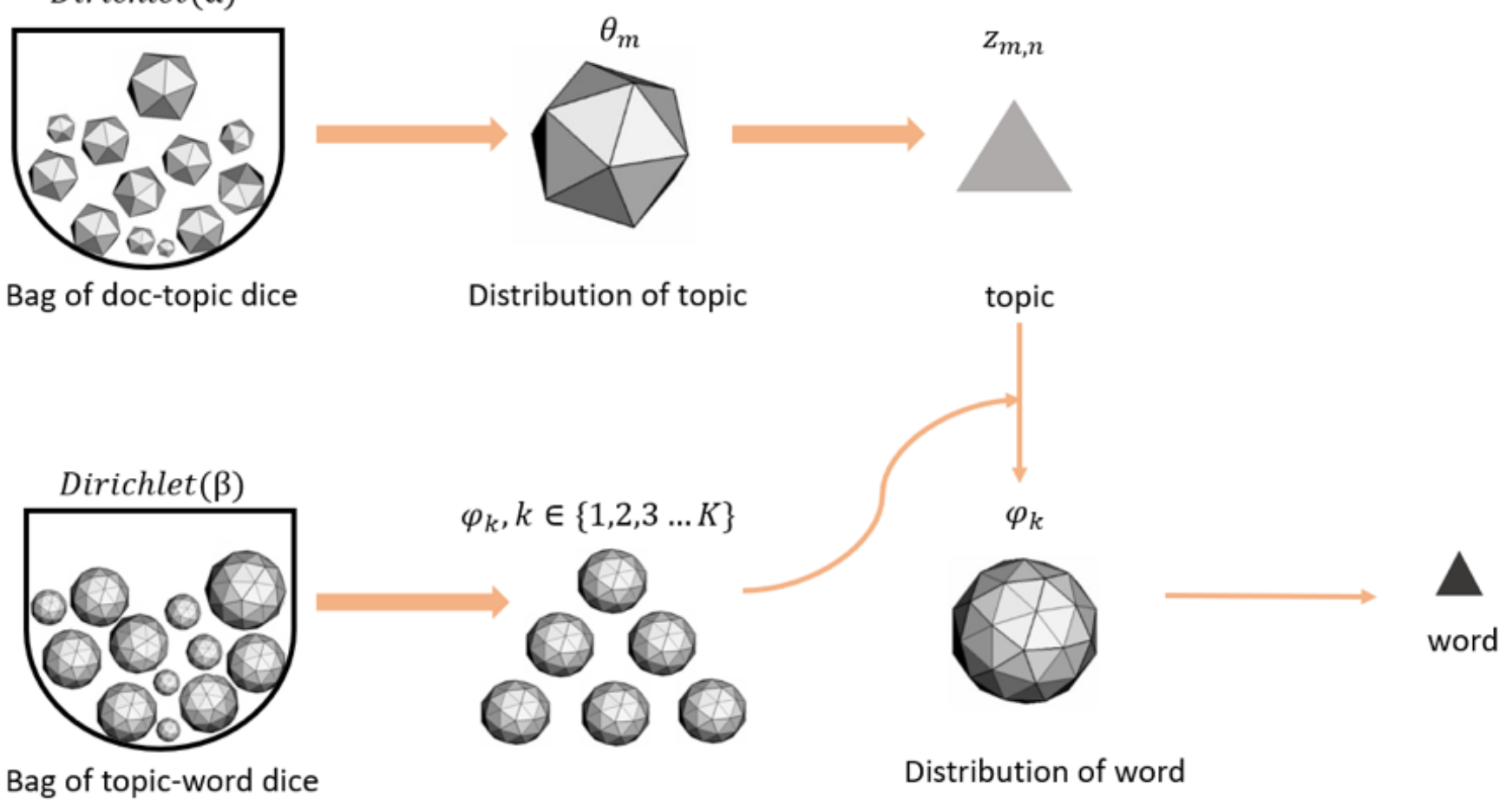

Figure 1: LDA Model

The language we used of text collections throughout the paper, referring to entities such as “words,” “documents,” and “corpora.”(p995) Our goal is to find the topic dictribution of each document and the word distribution of each topic. Also, if we are given a new document, we need to use some parameters trained

from our model, to inference the topic of taht new document.

To mimic the process of human wrinting an article, the author always needs to have some fixed topics and then uses words withch are strongly related to those topics. So when it comes to machine learning, we also pretend that we need to have the topic at first and then we can generate words under that topic.

To describe the algorithm in an esier way, we can treat distribution as a multi-sided dice, each side represents a topic(word), the probabilities of being up of each side are defferent. Added Bayesian idea, the LDA algorihm can be showed as: - There are two big boxes, one is full of “doc-topic” dices, and another box is full of “topic-word” dices. - Pick K “topic-word” dices from the second box, label tham as $1, 2, \ldots, K$. - For each document, we pick one “doc-topic” dice from the first box and repeat followings to get words: - through this dice, obtain the topic z - chooes the “doc-topic” dice whose lable is z and through it, then we can obtain one word

In other words, LDA algorithm assumes the following generative process for each document $\overrightarrow{w}$:

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

   (a). Choose a topic $z_n \sim Multi(\theta)$

   (b). Choose a word w_n from $p(w_n \mid z_n, \beta)$, a multinomal probability conditioned on th etopic z_n.

3.1 Mathematical analysis

First of all, let’s define some terms which will use in the following text:

· D(a corpus) is the collection of M documents;

· V is the total number of words from a vocabulary;

· M is the number of documents;

· K is number of topics of the corpus;

· N_m is the number of words in the $m^{th} document$;

· w_{mn} is the n^{th} word in docuent m;

· z_{mn} is the topic of word w_{mn};

· θ_m is the distribution for document m;

LDA assumes that the prior distribution of topics is a Dirichlet distribution, i.e., for any document d, the distribution of its topic is

$$\theta_d = Dirichlet(\vec{\alpha})$$

where α is a K-dimension vector.

A k-dimensional Dirichlet random variable θ can take values in the $(k-1)$-simplex (a k-vector lies in the $(k-1)$-simplex if $\theta_i \geq 0$, $\sum_{i=1}^{k} \theta_i = 1$), and has the following probability density on the simplex:

$$p(\theta \mid \alpha) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \theta_1^{\alpha_1 - 1} \dots \theta_k^{\alpha_k - 1}$$

Given the parameters α and β, the joint distribution of a topic mixture θ, a set of N topics $\mathbf{z} = (z_1, z_2, \dots z_N)$ and a set of N words $\mathbf{w} = (w_1, w_2, \dots w_N)$ is given by:

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

Integrating over θ and summing over z, we obtain the marginal distribution of a document $\mathbf{w}$:

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

Finally, taking the product of the marginal probabilities of single documents, we obtain the probability of a corpus $\mathbf{D} = (\mathbf{w}_1, \mathbf{w}_2, \dots \mathbf{w}_M)$:

$$\begin{aligned} p(\mathbf{D} \mid \alpha, \beta) &= \prod_{d=1}^{M} p(\mathbf{w}_d \mid \alpha, \beta) \\ &= \prod_{d=1}^{M} \int p(\theta_d \mid \alpha) \prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn} \mid \theta) p(w_{dn} \mid z_{dn}, \beta) d\,\theta_d \end{aligned}$$

3.2 Variational EM

(All code related to this part are in variational_EM.ipynb)

To approach the posterior probability $p(\theta, \mathbf{z} \mid \mathbf{w}, \alpha, \beta)$, we use Variational EM method.

3.2.1 Parameter declaration

v: vocabulary, $v = 1, 2, \dots, V$

z: topic , $z = 1, 2, \dots, K$

m: document, $m = 1, 2, \dots, M$

w: word, $w = 1, 2, \dots N_1, N_1 + 1, \dots N_1 + N_2, \dots, \sum_{m=1}^{N} N_m$

α: a K dim vector, represents the topic distribution of K topics

β: a $K \times V$ matrix, β_{ij} represents probability of the j^{th} word of i^{th} topic.

ϕ: a $M \times N \times K$ matrix, where $N = \sum_{m=1}^{N} N_m$, represents probability of each topic for words in each document

γ: a $M \times K$ matrix, represents the probability of topic for each document

3.2.2 Variation Inference (E-Step)

We need to compute the posterior distribution of hidden variables:

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

But this distribution is intractable due to the coupling between θ and β. So we use the variation distribution on latent variables:

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

An optimization problem that determins the values of γ and ϕ with respect to KL-Divergence:

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

Then we denote $q(\theta, \mathbf{z} \mid \gamma, \phi)$ by q, the KL-Divergence between q and $p(\theta, \mathbf{z} \mid \mathbf{w}, \alpha, \beta)$ is

$$\begin{aligned} D(q||p(\theta, \mathbf{z} \mid \mathbf{w}, \alpha, \beta)) &= E_q[log\, q] - E_q[log\, p(\theta, \mathbf{z} \mid \mathbf{w}, \alpha, \beta)] \\ &= E_q[log\, q] - E_q[log\, p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta)] + log\, p(\mathbf{w} \mid \alpha, \beta) \end{aligned}$$

Using Jensen's inequality, we have the bound as $log\, p(\mathbf{w} \mid \alpha, \beta) \geq E_q[log\, p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta)] - E_q[log(\theta, \mathbf{z})]$. And by denoting $E_q[log(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta)] - E_q[log\, q(\theta, \mathbf{z})]$ by $L(\gamma, \phi; \alpha, \beta)$, we have

$$log\, P(\mathbf{w} \mid \alpha, \beta) = L(\gamma, \phi; \alpha, \beta) + D(q(\theta, \mathbf{z} \mid \gamma, \phi) || p(\theta, \mathbf{z} \mid \mathbf{w}, \alpha, \beta))$$

So minimizeing the KL-Divergence D is equivalent to maximizing likelihood L, then we need to maximize the following five parts one by one:

$$L(\gamma, \phi; \alpha, \beta) = E_q[log\, p(\theta \mid \alpha)] + E_q[log\, p(\mathbf{z} \mid \theta)] + E_q[log\, p(\mathbf{w} \mid \mathbf{z}, \beta)] - E_q[log\, q(\theta) - E_q[log\, q(\mathbf{z})]]$$

Finally, by computing the derivatives of L and set them equals to 0, we obtain update equations:

$$\phi_{ni} \propto \beta_{iv}\, exp\{\psi(\gamma_i) - \psi(\sum_{j=1}^{K} \gamma_j)\}$$

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

where ψ is the digamma function, the first derivative of the log Gamma function.

Variational Inference Algorithm

(Source code in Package- lda_package)

- initialize $\phi_{ni}^0 = \frac{1}{K}$ for all i and n.
- initialize $\gamma_i^0 = \alpha_i + \frac{N}{K}$ for all i.
- repeat
    - for $n = 1$ to N
    - for $i = 1$ to K
        - $\phi_{ni}^{t+1} = \beta_{iw_n} exp\{\psi(\gamma_i^t)\}$.
        - normalize ϕ_{ni}^{t+1} to sum 1.
    - $\gamma^{t+1} = \alpha + \sum_{n=1}^{N} \phi_n^{t+1}$.
- until convergence

3.2.3 Parameter estimation (M-step)

Now, given a corpus of documents $\mathbf{D} = (\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_M)$, we hope to find proper parameters. So, in this M-step, maximize the bound with respect to the model parameters α and β.

Firstly, we maximize $L(\gamma, \phi; \alpha, \beta)$ with respect to β, by taking the derivative with respect to β_{ij} and setting it to zero, we have:

$$\beta_{ij} \propto \sum_{m=1}^{M} \sum_{n=1}^{N_m} \phi_{m,n,i} w_{mn}^j$$

here we need to attention that β is a probability matrix whose rows represent words probability of one particular topic, so the sum of each rows should equal one.

Secondly, we maximize $L(\gamma, \phi; \alpha, \beta)$ with respect to α, and then take the derivative with respect to α_i. However, it is difficult to compute α_i by setting the derivative to zero:

$$\frac{\partial L}{\partial \alpha_i} = M \left(\psi(\sum_{j=1}^{K} \alpha_i) - \psi(\alpha_i) \right) + \sum_{m=1}^{M} \left(\psi_{mi} - \psi \left(\sum_{j=1}^{K} \gamma_{mj} \right) \right) := g_i$$

Because this derivative depends on α_i, where $j \neq i$. So we need to use an iterative method (eg. Newton-Raphson method) to find the extreme point. Then we can compute the Hessian Matrix as:

$$\frac{\partial^2 L}{\partial \alpha_i \partial \alpha_j} = -\delta(i,j) M \psi'(\alpha_i) + M \psi' \left(\sum_{j=1}^{K} \alpha_i \right) := H_{ij}$$

Finally, we only need to use Newton-Raphson methods to update α and obtain the answer when converge. The Newton-Raphson optimization technique finds a stationary point of a function by iterating

$$\alpha_{new} = \alpha_{old} - H(\alpha_{old})^{-1} g(\alpha_{old})$$

where $H(\alpha)$ is the Hessian matrix and $g(\alpha)$ is the gradient vector. If the Hessian matrix is of form $H = diag(h) + \mathbf{1}\, z\, \mathbf{1}^T$, then we can culculate that

$$(H^{-1}g)_i = \frac{g_i - c}{h_i}$$

$$c = \frac{\sum_{j=1}^{K} g_j/h_j}{z^{-1} + \sum_{j=1}^{K} h_j^{-1}}$$

3.2.4 Combination (Variational EM)

After the procedures above, now we can combine all the algorithms into a summary:

1. (E-step) For each document, find the optimizing values of the variational parameters $\gamma_d^*, \phi_d^* : \ d \in D$. This is done as described in the previous section.

2. (M-step) Maximize the resulting lower bound on the log likelihood with respect to the model parameters α and β. This corresponds to finding maximum likelihood estimates with expected sufficient statistics for each document under the approximate posterior which is computed in the E-step.

These two steps are repeated until the lower bound on the log likelihood converges.

Variational EM Algorithm

Step1: input:

- K: the number of topics
- M: the number of documents
- D: contains all words of documents

Step2: initialize α and β

Step3: start Varitional EM algorithm:

- Use variational inference algorithm to update γ and ϕ:
- (1). initialize $\phi_{ni}^0 = \frac{1}{K}$ for all i and n.
- (2). initialize $\gamma_i^0 = \alpha_i + \frac{N}{K}$ for all i.

- repeat
- for $n = 1$ to N
- for $i = 1$ to K
    - $\phi_{ni}^{t+1} = \beta_{iw_n} exp\{\psi(\gamma_i^t)\}$.
    - normalize ϕ_{ni}^{t+1} to sum 1.
- $\gamma^{t+1} = \alpha + \sum_{n=1}^{N} \phi_n^{t+1}$.
- until convergence

- Use parameter estimation to estimate α and β:
- use the converged ϕ and γ to update
- until convergence

Step4: when all parameters ocnvergede, break.

Step5: output posterior parameters α, β, γ and ϕ, end.

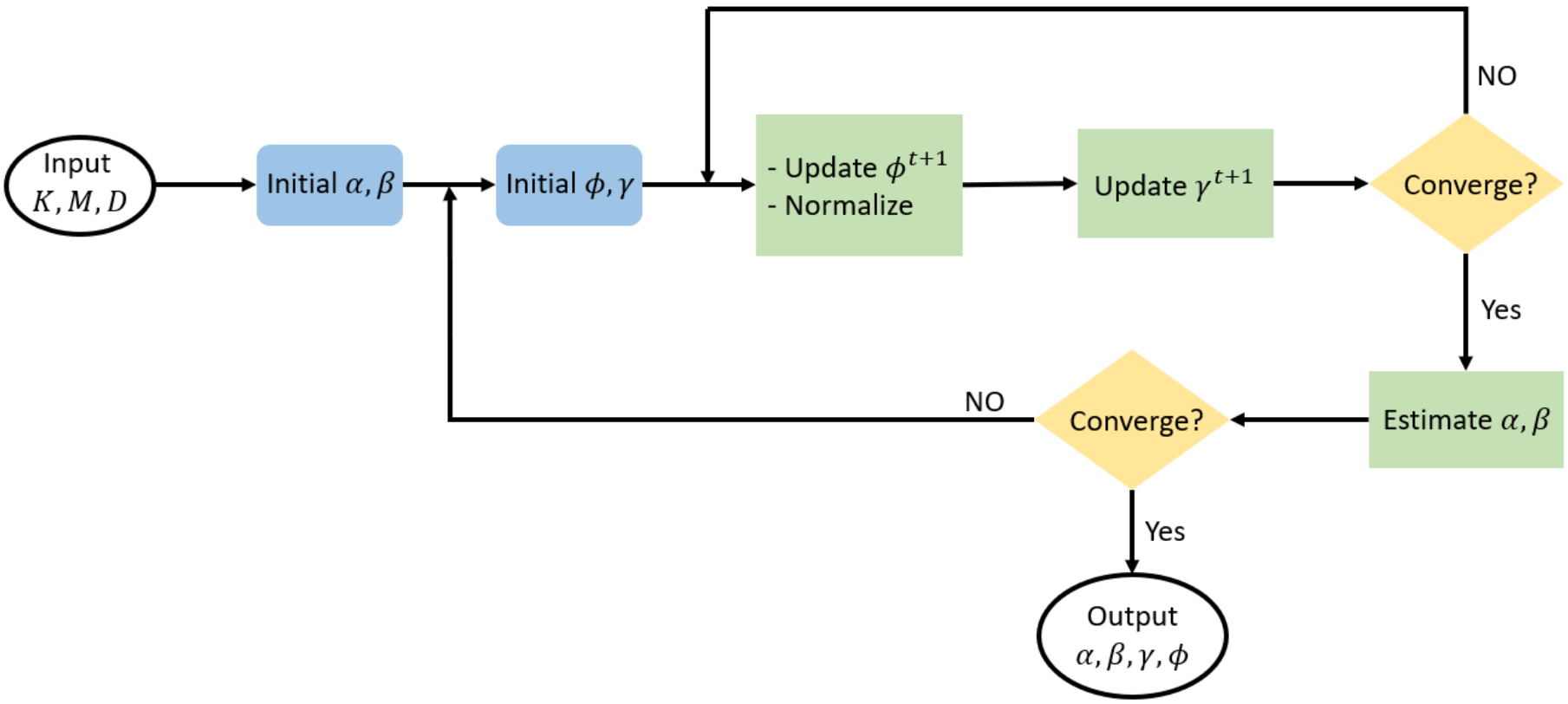

Figure 2: Variational EM

4 Optimization

(All code related to this part are in LDA-optimization.ipynb)

4.1 Overview

To optimize our model, we consider two different aspects- Algorithm Optimization and Code Optimization

Algorithm Optimization

- Use $log(a+b) = log(a) + log(1 + e^{log(b)-log(a)})$ to optimize the nomalization process
- Use $\gamma_i^{t+1} = \alpha_i + \sum_{n=1}^{N} \phi_{dn}^{t+1} = \gamma_i^t + \sum_{n=1}^{N} (\phi_{dn}^{t+1} - \phi_{dn}^t)$ to optimize the gamma update process
- Use vetorization and replace the for loop

Parellel Optimization

- Use `numba.jit`

Below are the optimization details and mathematical analysis

4.2 Algorithm optimization

First of all, we hope to improve our functions in order to speed up. When look back at the algorithm, we find there are some places that can be replaced by other optimal methods.

4.2.1 Optimize the Normalization Process

After updating ϕ, we need to normalize ϕ to make it a probabiliy. So we have,

$$\phi_i = \frac{\phi_i}{\phi 1 + \phi_2 + \ldots + \phi_K}$$

Actually we can use $\log$ function to optimize this process,

$$log(\phi_i) = \log(\phi_i) - \log(\phi_1 + \phi_2 + \ldots + \phi_K)$$

So in normalization, this method can add in pairs during each iteration after geting $log(\phi_1), log(\phi_2), \ldots, log(\phi_K)$ and finally speed up the summation of $log(\phi_1 + \phi_2 + \ldots + \phi_K)$ And because of the target value has become $log(\phi_1)$, the iterative formula of ϕ should be changed into:

$$log(\phi_{dni}^{t+1}) = log(\beta_{i,w_n}) + \psi(\gamma_{di}^t)$$

Optimize Code Sample

```
for i in range(K):
    log_phi[m][n,i] = np.log(beta[i,np.int(words[m][n])]) + digamma(gamma[m,i])
    logsum = log_sum(logsum, log_phi[m][n,i])
log_phi_mn = log_phi[m][n,:] - logsum         # use new metohd to implement nomalizat
ion
log_phi[m][n,:] = log_phi_mn

phi[m][n,:] = np.exp(log_phi_mn)
```

4.2.2 Optimize γ Updating Process

When we are updating the γ, we use

$$\gamma_i^0 = \alpha_i + \frac{N}{K}$$

$$\gamma_i^{t+1} = \alpha_i + \sum_{n=1}^{N} \phi_{dn}^{t+1}$$

But we can find that the γ^t has the α information γ^{t+1} needs, so we can optimize this process

$$\begin{aligned}
\gamma_i^{t+1} &= \alpha_i + \sum_{n=1}^{N} \phi_{dn}^{t+1} \\
&= \alpha_i + \sum_{n=1}^{N} (\phi_n^t + \triangle \phi_{dn}^{t+1}) \\
&= (\alpha_i + \sum_{n=1}^{N} \phi_n^t) + \sum_{n=1}^{N} \triangle \phi_{dn}^{t+1}) \\
&= \gamma_i^t + \sum_{n=1}^{N} \left(\phi_{dn}^{t+1} - \phi_{dn}^t\right)
\end{aligned}$$

Optimize Code Sample

```
d_phi = phi[m] - phi_old[m]
gamma[m,:]  = gamma[m,:] + np.sum(d_phi, axis = 0)
```

4.2.3 Vectorization

We also use vector/matrix manipulation to replace the for loop

Optimize Code Sample

```
# Example

# for-loop
for i in range(K):
            g1 = M*(digamma(np.sum(alpha))-digamma(alpha[i]))
            g2 = 0
            for d in range(M):
                g2 += digamma(gamma[d,i])-digamma(np.sum(gamma[d,:]))
            g[i] = g1 + g2
# Vector
g = M*(digamma(np.sum(alpha))-digamma(alpha))
        + np.sum(digamma(gamma) -np.tile(digamma(np.sum(gamma,axis=1)),(K,1)).T,axis=
0)
```

4.2.4 Test Efficiency of Algorithm Optimization

The test data we used was a small generated data

```
doc_a = "Brocolli is good to eat. My brother likes to eat good brocolli, but not my m
other."
doc_b = "My mother spends a lot of time driving my brother around to baseball practic
e."
doc_c = "Some health experts suggest that driving may cause increased tension and blo
od pressure."
doc_d = "I often feel pressure to perform well at school, but my mother never seems t
o drive my brother to do better."
doc_e = "Health professionals say that brocolli is good for your health."
```

Here is the running time comparasion before and after we use algorithm optimization.

```
%%time
ans1 = lda_package.variation_EM(M, k, text_, N, V_words, alpha, beta, gamma, phi, iteration = 1000)
```

```
CPU times: user 12 ms, sys: 4 ms, total: 16 ms
Wall time: 15.7 ms
```

```
%%time
ans2 = lda_package.variation_EM_new(M, k, text_, N, V_words, alpha, beta, gamma, phi, iteration = 1000)
```

```
CPU times: user 12 ms, sys: 0 ns, total: 12 ms
Wall time: 10.8 ms
```

Figure 3: time comparison-algorithm optimization

4.3 Parellel optimization

As LDA is an algorithm which trains multiple documents, the analysis of documents can be worked on in different partitions at the same time. Therefore we also use JIT to implement the paralleing. We replace `range` with `prange` and also call `parallel=True` when using @jit.And here is the time comparasion before and after JIT:

```
%%time
ans3 = lda_package.variation_EM_new(M, k, text_, N, V_words, alpha, beta, gamma, phi, iteration = 1000)
```

```
CPU times: user 8 ms, sys: 8 ms, total: 16 ms
Wall time: 10.9 ms
```

```
%%time
ans4 = variation_EM_new_jit(M, k, text_, N, V_words, alpha, beta, gamma, phi, iteration = 1000)
```

```
CPU times: user 8 ms, sys: 4 ms, total: 12 ms
Wall time: 8.84 ms
```

Figure 4: time comparison-JIT optimization

Even though we only use a small data to test our code, which means it won't take long time even for the original one, we can still see there is a huge increase based on our optimization.

5 Unit test

(All code related to this part are in unit_test.ipynb)

In this section, we test our five main functions:

a. test_alpha.py : we test whether the elements of output are all greater than 0.

b. test_beta.py: we test whether the elements of output are all greater than 0. And we also test whether the row sums equal 0.

c. test_converge1.py : we test if the `old` and `new` are identity, whther the output is True.

d. test_converge2.py: we test if the `old` and `new` are identity, whther the output is True.

e. test_optimize_vp.py: since the output contains two part, `phi` and `gamma` , so we test whether all elements of `phi` and `gamma` are greater than 0 and whether the row sums of `gamma` equal 0.

The test results are showed as below:

```
============================= test session starts ==============================
platform linux -- Python 3.6.5, pytest-3.5.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/jovyan/work/latent-dirichlet-allocation, inifile:
collected 5 items

test_alpha.py .                                                          [ 20%]
test_beta.py .                                                           [ 40%]
test_converge1.py .                                                      [ 60%]
test_converge2.py .                                                      [ 80%]
test_optimize_vp.py .                                                    [100%]

========================== 5 passed in 0.34 seconds ===========================
```

Figure 5: unit test result

6 Experimental test

6.1 Simulated Data

In this part we use a simulated data from the paper, which we have already known the “real” topic. There are 72 unique words and 4 topics. We divide the paragraph into six sentences and treat each as a document.

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. “Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services,” Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center’s share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

Figure 6: simulated data from LDA paper

6.1.1 Test Code

```
# Create sample documents
doc_a = """TheWilliam Randolph Hearst Foundation will give 1.25 million to Lincoln Ce
nter, Metropolitan
        Opera Co., New York Philharmonic and Juilliard School."""
doc_b = """ Our board felt that we had a real opportunity to make a mark on the futur
e of the performing
           arts with these grants an act every bit as important as our traditional ar
eas of support in health,
           medical research, education and the social services, Hearst Foundation Pre
sident Randolph A. Hearst
           said Monday in announcing the grants."""
doc_c = """Lincoln Center's share will be 200000 for its new building, which
           will house young artists and provide new public facilities."""
doc_d = """The Metropolitan Opera Co. and New York Philharmonic will receive 400000 e
ach."""
doc_e = """The Juilliard School, where music and the performing arts are taught, will
get 250000. """
doc_f = """The Hearst Foundation, a leading supporter of the Lincoln Center Consolida
ted Corporate Fund,
           will make its usual annual $100,000 donation, too."""

doc_set = [doc_a, doc_b, doc_c, doc_d, doc_e, doc_f]
```

```
# Use the data preprocessing function from our package
texts, dictionary, corpus = lda_package.data_clean(doc_set)
text_ = lda_package.data_process(texts, dictionary)

# Initialize parameter
M = 6
k = 4
N = np.array(list(map(len, text_)))
V = len(dictionary)
V_words = range(V)
alpha = np.random.dirichlet(10*np.ones(k),1)[0]
beta = np.random.dirichlet(np.ones(V),k)
phi = np.array([1/k*np.ones([N[m],k]) for m in range(M)])
gamma = np.tile(alpha,(M,1)) + np.tile(N/k,(k,1)).T
```

6.1.2 Results

After running our functions, we get the probability distribution of words of each topic. We can see from this picture-Figure7, there are some words in each topic that are preferrable. And we can go more details in the next picture-Figure8 to see what specific words preferred by each topic.

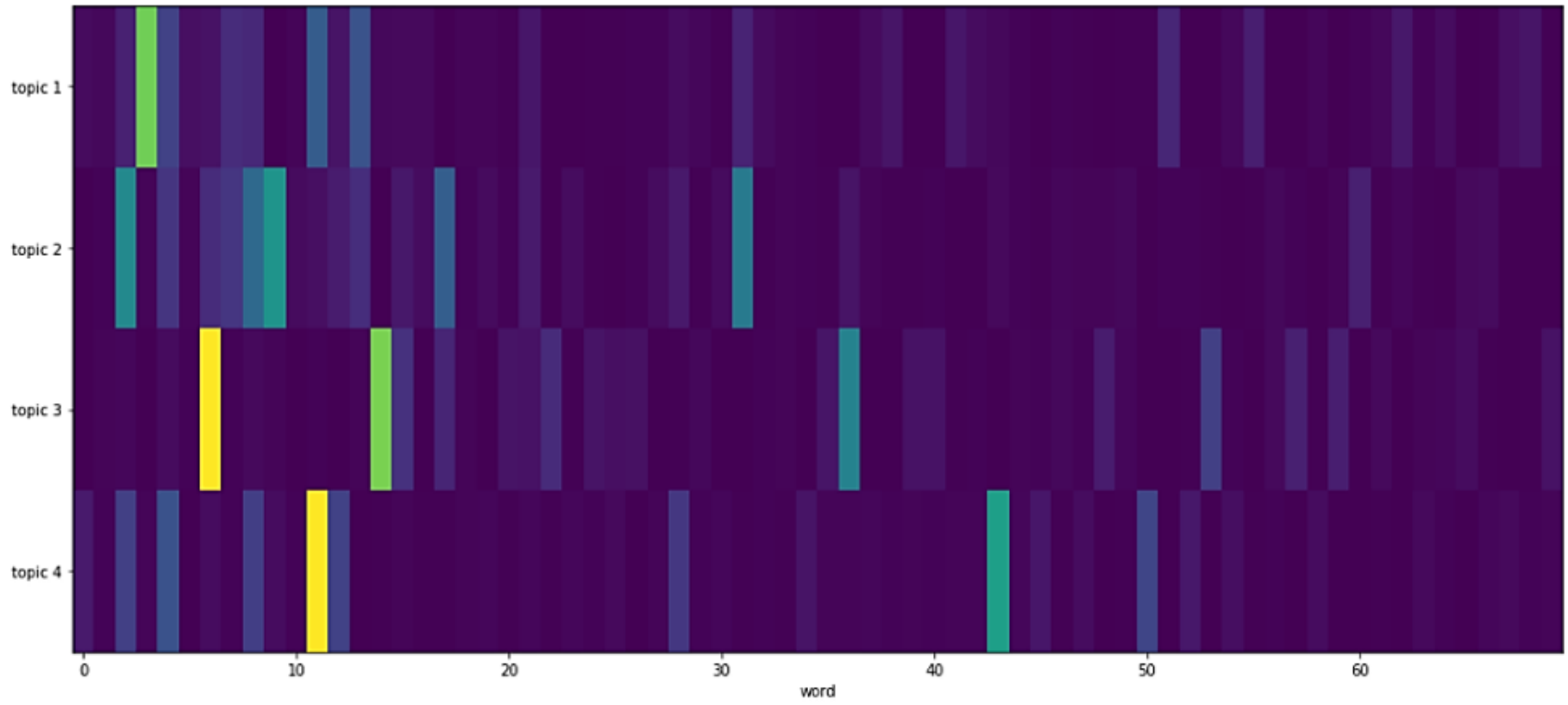

Figure 7: 6.1.2 probability matrix

Because LDA is a “soft-clustering” algorithm, it only gives the clusters(topics) but not summaries the name of each cluster. So in Figure 8, we just call these four topics as “topic 1”, “topic 2”,“topic 3” and “topic 4”. We print the four most popular words of each topic. Even though this data set is small, the result is decent, espacially topic 3 and topic 4. In topic 3 we have “hearst”, “randolph”, “perform”, which are all belong to “Children”. In topic 4 we have “new”, “support”, “foundation” and “provid” which are all belong to “Budgets”.

```
topic 1
 0.06258035061866102 * co + 0.0411663679334841 * new + 0.03960874829765505 * philharmon + 0.03625888930214377 * foundat

topic 2
 0.0507455950628512 * metropolitan + 0.048846017700947916 * center + 0.04632554625666445 * make + 0.043617625645694255 * lincol
n

topic 3
 0.09738722371507769 * hearst + 0.06356043359084296 * randolph + 0.04782145548524448 * perform + 0.03565054837060334 * share

topic 4
 0.10152731463363168 * new + 0.052564459004376 * support + 0.03927570181943087 * foundat + 0.03687693265489178 * provid
```

Figure 8: 6.1.2 probability expression

6.2 Real-World Data

6.2.1 Dataset

In this part we use a real-world data from the website(http://www.cs.columbia.edu/~blei/lda-c/ (http://www.cs.columbia.edu/~blei/lda-c/)) to test the model. The whole data set is in the folder named Test_data, the file named ap.text. Now we have 13,500 documents and the corpus has more than 25,000 words in total.Then we choose the number of topics as ten. This time we use the stop word list more complete than we used in the previous example (see stop_word.txt).

6.2.2 Results

By using our function, we calculate the topic-word distribution. Here we use the same trick to make the visualization clearer. Figure 9 shows the words probability distribution of each topic.

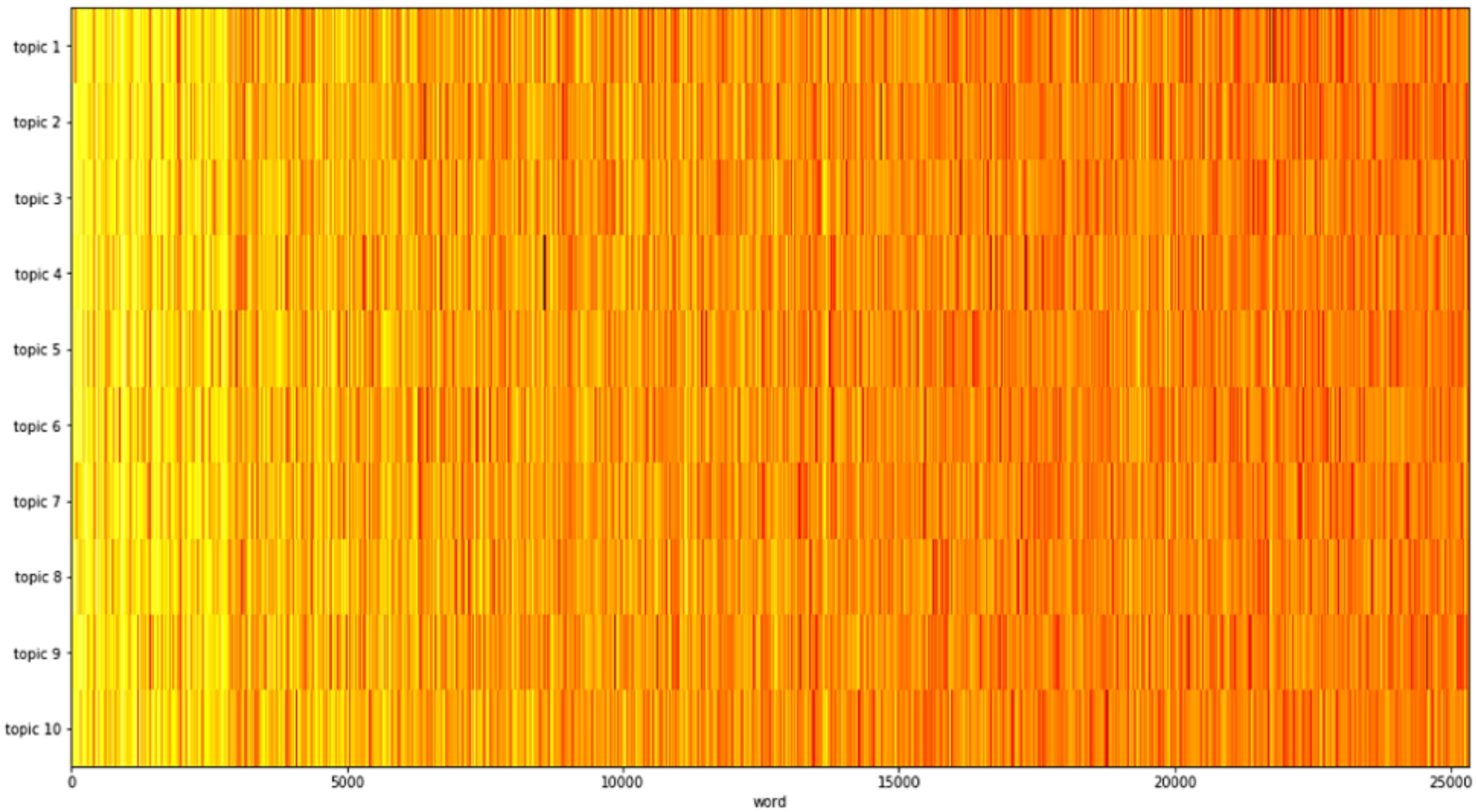

Figure 9: 6.2.2 topic-word distribution

From Figure 9, we find that although for each topic, the preferable words spread widely, most of them lie in the first 5000. So we hope to zoom in this part and see detail distributions, which is showed in figure 10.

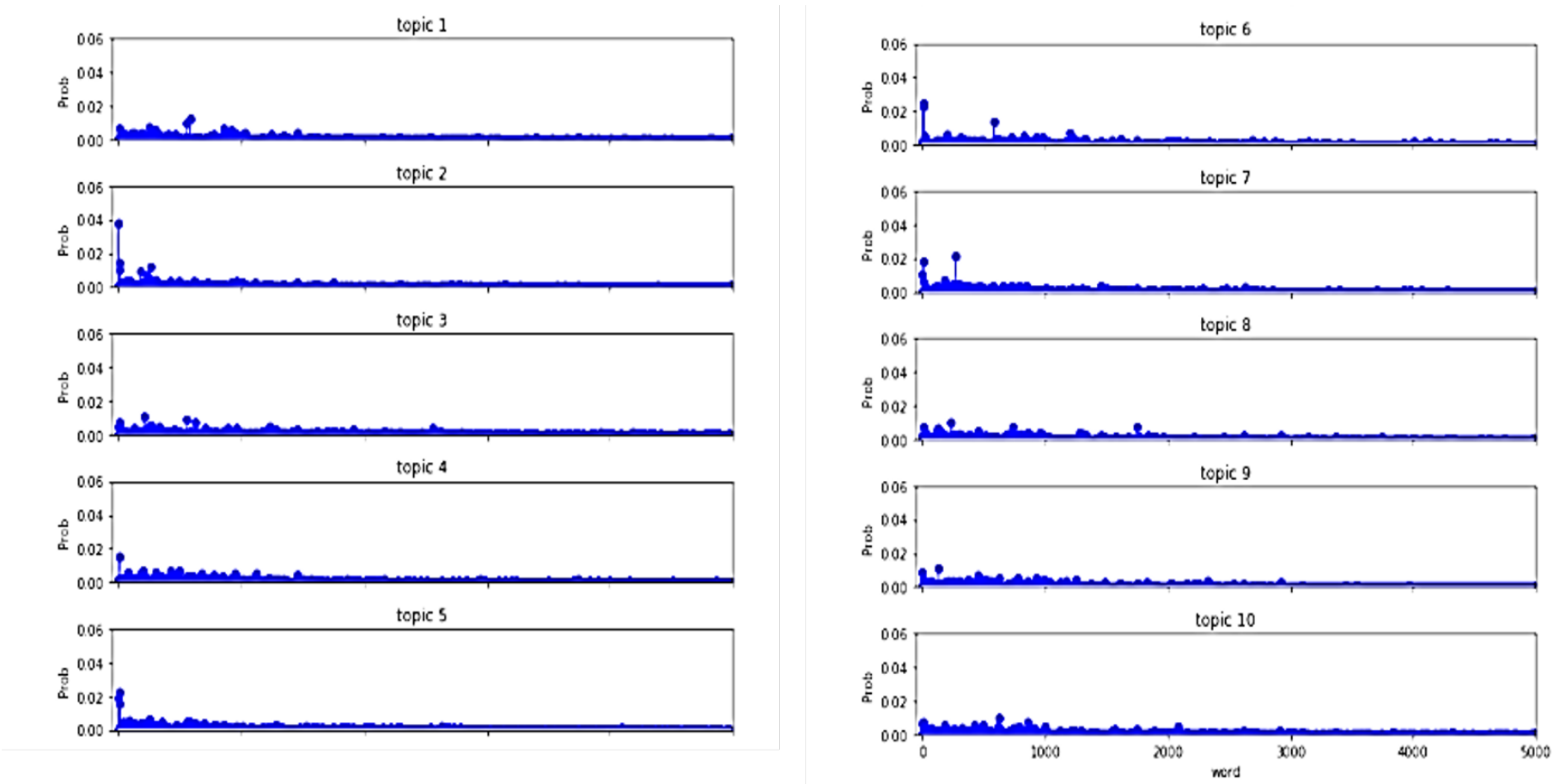

Figure 10: 6.2.2 topic-word distribution(first 5000)

From this result, we can definitely obtain the insights that different topic have the preference for different words. And then our function calculate the result of document-topic distribution as well, which you can see from the picture below in figure 11. We randomly select the documents and see their topic distribution. From this plot we can say that document 1 is most likely about topic 4 and 5, document 29 perhaps is all about topic 1

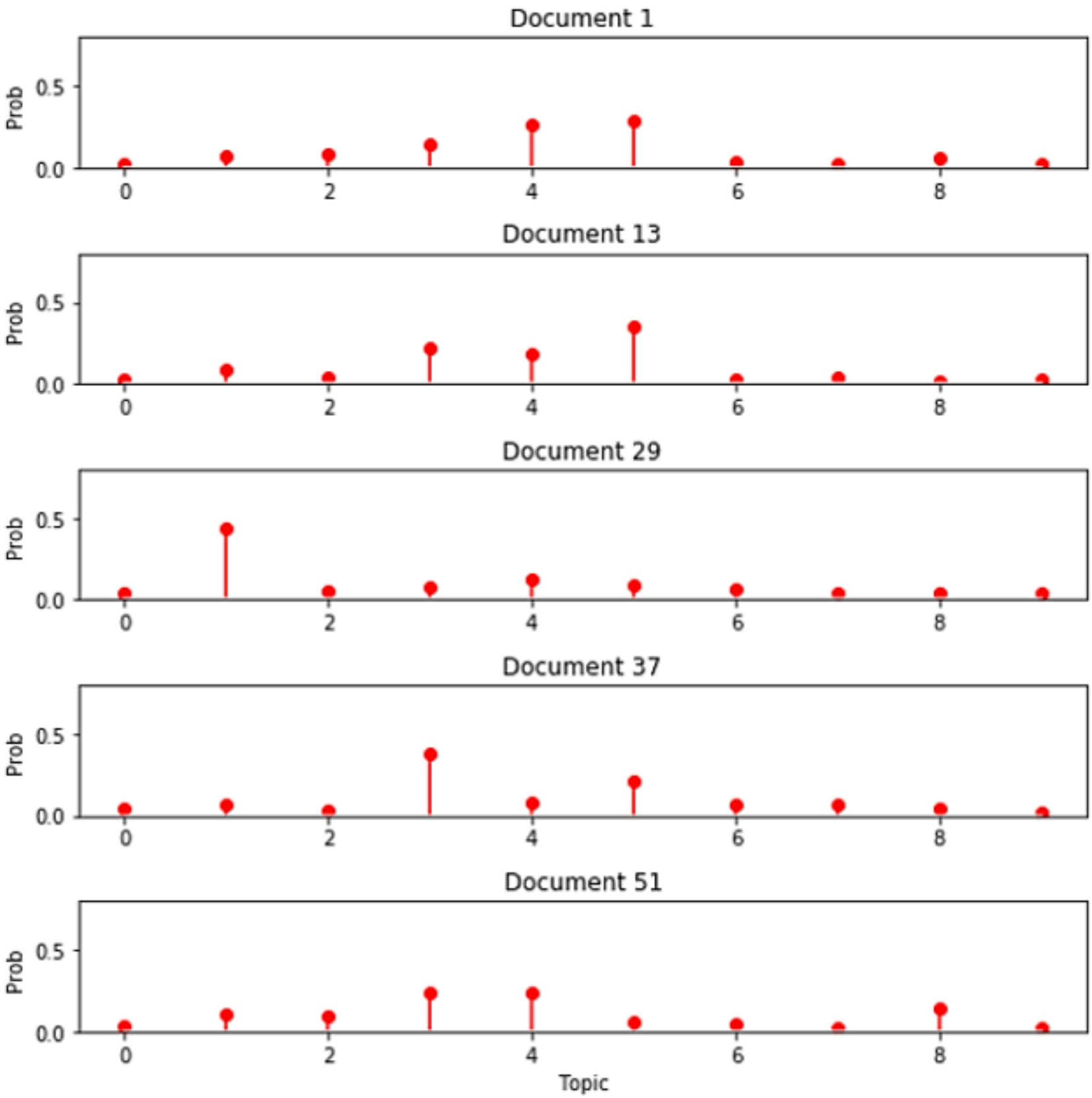

Figure 11: 6.2.2 doc-topic distribution

7 Comparative analysis

Except for latent dirichlet allocation, there are many other similar text topic model. For example, latent semantic indexing(LSI). In this section, we will briefly introduce this algorithm and compare it with our LDA model in performance.

7.1 LSI

Latent semantic analysis (LSA) is a technique in natural language processing, in particular distributional semantics, of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. Traditional semantics usually study the meaning of words and words, as well as the relationship between words and words, such as righteousness, near righteousness, antisense and so on.

The underlying semantic analysis explores a relationship hidden behind the words, which is not based on the definition of a dictionary, but the use of words as the most basic reference.The idea came from psycholinguists.They argue that hundreds of languages around the world should have a common, simple mechanism that allows anyone to grow up in a particular language.

Under the guidance of this idea, people found a simple mathematical model, the input of the model is composed of any language written document library, the output is the word of the language, a mathematical expression of the word (vector).The comparison between the word, the relationship between words, and even the meaning of any piece of text is generated by the operation of this vector.

LSI obtains text top based on singular value decomposition (SVD) method.

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}T^T_{n\times n}$$

Sometimes we need to reduce the dimension of matrix, then:

$$A_{m\times n} \approx U_{m\times k}\Sigma_{k\times k}T^T_{k\times n}$$

Assume we have m documents, each documents has n words. The matrix A is a represent of eigenvalues, A_{ij} is the eigenvalue of i^{th} document's j^{th} word. Here we always use normalized `TF-IDF` . k is the number of topics, which is usually smaller than m.

7.2 Comparison

After understanding these three algorithms thoroughly, it's not difficult to compare their performance by calculating the precision (accuracy) of document-topic distribution.

From the result below, we found for the "short" eassy, LSI showed even better performance than LDA. That is because LSI assumes multivariate normal distribution of the data. However, text data are count data. This may cause some problems in practice. Also, The real world data is usually a significantly large dataset so that the estimation of parameters is slow (because of the SVD step) and there is no guarantee of global optimism.

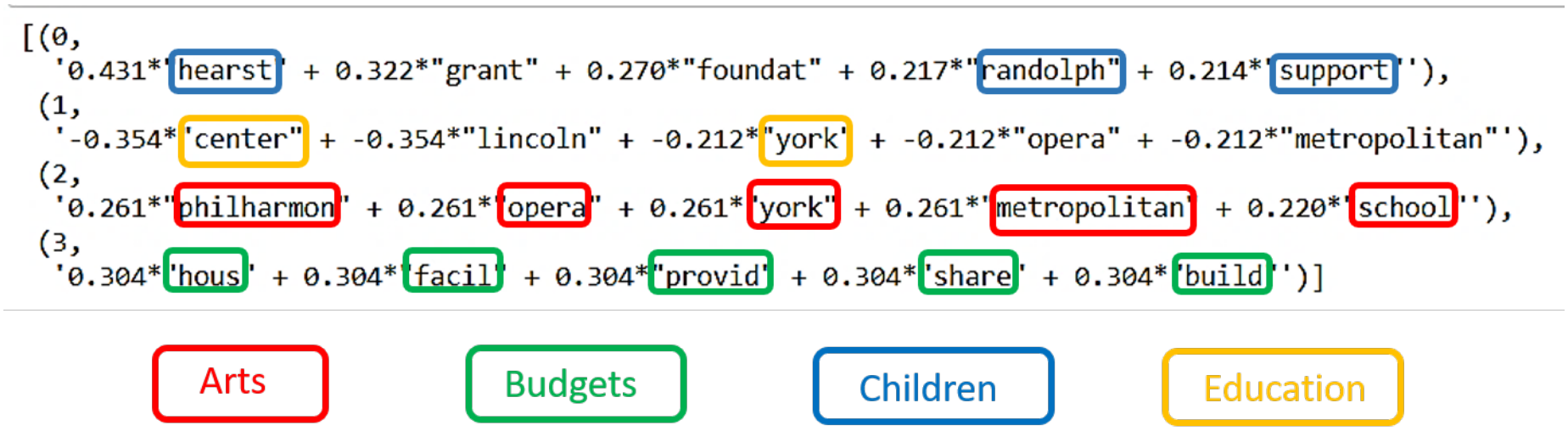

Figure 12: 7.2 LSI result

8 Conclusion/Discussion

In this report, we successfully achieve the LDA model using Python. And in order to test the model, we first use the simulated data, which we have already known the topic and the preferrable word in each topic. And we found though the test corpus is very short, we still achieve a great result. And we want to try a larger dataset to see if it has a better performance.We use the real-world data from AP Corpus, which have 13,500 documents and more than 25,000 different words. From the visulized result we have, we can definitely tell which topic has what kinds of preferrable words. Also, one document has what kinds of topics. For the optimization part. Besides using some high performance computing methods like JIT, we look back our algorithom, and optimize our algorithm both methematically and computationally. As for the part the LDA model could improve,the LDA model don't show very high performance in the short text(like Twitter data), due to the sparse relation of topics and words.

Reference

[1] Blei, David M, A. Y. Ng, and M. I. Jordan. "Latent dirichlet allocation." J Machine Learning Research Archive 3(2003):993-1022.

[2] Rosario, Barbara. "Latent Semantic Indexing: An overview." Techn.rep.infosys 25.4(2000):36-40.

[3] Zhao Zhou. "Variational Inference for LDA." The Hong Kong University of Science and Technology

[4] https://cs.stanford.edu/~ppasupat/a9online/1140.html (https://cs.stanford.edu/~ppasupat/a9online/1140.html)