Yeet Another Latent Dirichlet Allocation

Kehan Lyu *
Department of Computer Science
Duke University
kl320@duke.edu

Tiangang Chen *
Department of Computer Science
Duke University
tc260@duke.edu

May 1, 2019

Abstract

Latent Dirichlet Allocation is an essential algorithm in text processing tasks. In this final project of the course, we present our implementation of the LDA algorithm. We applied collapsed Gibbs sampling to achieve parameter inference. Our implementation was also optimized using Cython. The resulting document-topic distributions were treated as features to perform document classification on the 20 Newsgroups dataset. Two other language modeling algorithms, NMF and LSA, were evaluated against our implementation as well as readily-available LDA software packages. Finally, our code was packaged and published on GitHub and PyPI. [2].

1 Background

In this final project of the course, we explore the Latent Dirichlet Allocation paper.[1]. Latent Dirichlet Allocation is a model that provides full generative probability semantic of documents. The goal is to use short topics to describe large documents while preserving the essential statistical relationships. Latent Dirichlet Allocation is widely adopted in Natural Language Processing tasks like classification, novelty detection, summarization, and similarity and relevance judgments.

TF-IDF is one of the competing algorithm for topic modeling. TF-IDF has a good performance on identifying key words in documents.It did not summary the documents in small length and it could not provide per document topic and inter documents topic.

Compare to other algorithms, Latent Dirichlet Allocation has the flexibility of providing per document description and inter document comparison.

For our research, we will explore the topic modeling of LDA by using the generated topics as features to perform document classification and compare it with the competing algorithm like NMF and LDA. We

*equal contribution
[2]links https://github.com/cliburn/sta-663-2019

2 Description of Algorithm

Latent Dirichlet Allocation is a generative model for documents. LDA assumes each documents has latent topics that is generated by Dirichlet distribution. Also, for each topic, there is a distribution of words to describe the model.

The LDA generate process shows as follows:

1. Choose $N \sim \text{Poisson}(\xi)$.
2. Choose $\theta \sim \text{Dir}(\alpha)$.
3. For each of the N words w_n:
   (a) Choose a topic $z_n \sim \text{Multinomial}(\theta)$.
   (b) Choose a word w_n from $p(w_n | z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

Figure 1: The generate process for a document

where α and β are the hyperparameters, and θ is the document_topic distribution and topic z is the topic assignment for word z.

From the above generate process, we can have a full joint distribution as follow:

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

Figure 2: The full joint distribution for LDA generating process

The key problem of using LDA to implement topic modeling is the posterior inference. With the observed data, posterior inference is a method to reverse the generating process and solve for the posterior distribution of the latent variables. The posterior distribution shows as follows:

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

Figure 3: The posterior distribution of a document.

However, the posterior distribution can not be solved directly. We apply Gibbs Sampling as a inference method to estimate posterior distribution.

2.1 Gibbs Sampling

Gibbs Sampling is a MCMC sampling method that construct a Markov chain to approximate the posterior distribution showed in Figure 3. For LDA, Gibbs Sampling samples from the

Input: words $\mathbf{w} \in$ documents $\mathbf{d}$
Output: topic assignments $\mathbf{z}$ and counts $n_{d,k}, n_{k,w}$, and n_k
begin
  randomly initialize $\mathbf{z}$ and increment counters
  foreach *iteration* **do**
    for $i = 0 \to N-1$ **do**
      $word \leftarrow w[i]$
      $topic \leftarrow z[i]$
      $n_{d,topic}$-=1; $n_{word,topic}$-=1; n_{topic}-=1
      for $k = 0 \to K-1$ **do**
        $p(z = k|\cdot) = (n_{d,k} + \alpha_k)\frac{n_{k,w}+\beta_w}{n_k+\beta \times W}$
      end
      $topic \leftarrow$ sample from $p(z|\cdot)$
      $z[i] \leftarrow topic$
      $n_{d,topic}$+=1; $n_{word,topic}$+=1; n_{topic}+=1
    end
  end
  return $\mathbf{z}$, $n_{d,k}, n_{k,w}, n_k$
end

Figure 4: Pseudocode for Collapsed Gibbs Sampling

conditional probabilities of the latent variables.After a number of iteration, sampling from the distribution will converge and the sample will show similar distribution with the desired posterior distribution.

In detail, z_{iv} is the topic for word v in document i, π_i is the topic distribution for i-th document, and b_k is the distribution those words for topic k. For each iteration, sample z_{iv} with π_i and b_k, then sample π_i with $z_{iv} = k$ and b_k, and sample b_k with $z_{iv} = k$ and π_i.

The full conditional probability are as follows:

$$p(z_{iv} = k|\pi_i, b_k) \propto exp(log\pi_{ik} + logb_k, y_{iv})$$
$$p(\pi_i|z_{iv} = k, b_k) = Dir(\alpha + \sum_l I(z_{il} = k))$$
$$p(b_k|z_{iv} = k, \pi_k) = Dir(\beta + \sum_i \sum_l I(y_{il} = v, z_{il} = k))$$

2.2 Collapsed Gibbs Sampling

Collapsed Gibbs Sampling is an optimization to the original Gibbs Sampling algorithm. It simplifies the Gibbs Sampling algorithm by first leave out one word z_i, integrate out the multinomial parameters, and then resample z_i with the parameters. The pseudocode shows as Figure 4

3 Optimization for Performance

For performance optimization, we first considered implementing Gibbs sampling as the inference for Latent Dirichlet Allocation. We found that many previous research in Gibbs sampling showed that Gibbs sampling is over computational intensive. We implemented a prototype and run a few toy examples(5 documents with less than 5 words in each document). The performance is terrible so that we abandon Gibbs sampling immediately.

Then, we found a optimized algorithm for Gibbs sampling named as Collapsed Gibbs sampling. We located the only computational intense part in the algorithm, which is the inference function for Collapsed Gibbs sampling.

We optimized the inference function by rewriting the function with Cython package. Cython is an optimising static compiler for python. We first modified the data structure for the counters (such as topic_doc_word and p_z) from list of numpy array format to the numpy ndarray format to satisfied the Cython type conversion. By changing to static type declarations, cython can provide plain C performance. Also, we have a good amount of numpy array to be accessed and modified and Cython has good performance on numpy array manipulations.

We experimented the performance improvement gained from optimization by comparing the pure python version with the Cython optimized version. We tested both implementation on the whole 20 Newsgroup dataset (details in section 4.3). We measured the time of each iteration by setting parameters as k = 20, alpha=0.5, beta=0.5. We tested on the same machine to control the environment.

Pure Python Version		Cython Optimized Version	
Iteration: 0	Time: 125.75113201141357	Time:	29.670072078704834
Iteration: 1	Time: 122.77207207679749	Time:	29.654402017593384
Iteration: 2	Time: 122.44472098350525	Time:	30.754655122756958
Iteration: 3	Time: 113.75432014465332	Time:	31.316425323486328
Iteration: 4	Time: 112.43301200866699	Time:	30.885118007659912
Iteration: 5	Time: 120.74188685417175	Time:	29.850266218185425
Iteration: 6	Time: 113.52907609939575	Time:	30.068572998046875
Iteration: 7	Time: 113.32231903076172	Time:	31.804869174957275
Iteration: 8	Time: 113.32480883598328	Time:	33.027443170547485
Iteration: 9	Time: 113.02164793014526	Time:	33.01300406455994
Average: 117.10 seconds per iteration		Average: 31.05 seconds per iteration	

As a result, Cython optimization is 4 time faster than pure python implementation.

4 Evaluation

Rather than evaluating the produced topic models directly, we applied the produced features to document classification task. Since LDA is able to produce document-topic probability distributions, we use these distributions as features of the document and fed them to regular classification algorithms including SVM and MLP (multi-layer perceptron/fully connected neural network).

In this section, we first introduce the competing algorithms, the datasets we used. And finally we present our experiment setup details and results.

4.1 Competing Algorithms

Other than LDA, we explored two other algorithms that have been applied to topic modeling:

NMF Non-negative matrix factorization is a general decomposition technique. We have $X = WH$ where we'd like to decompose non-negative matrix X into a product of two non-negative matrices. In practice, we construct this as an optimization problem to minimize the distance between the original matrix and the product.

LSA Latent Semantic Analysis [2] introduces a SVD-based technique for text data analysis. We first process the documents to obtain TF-IDF design matrix, then apply truncated SVD to the design matrix to obtain topic and document representations.

In later sections we present experimental results using our implementation and these two competing algorithms on document classification tasks.

4.2 Simulated Dataset

First, we have a very tiny made-up toy example to check the correctness of our implementation. After replacing the words with vocabulary index, the corpus looks like this:

```
[0, 1, 2, 3, 4, 4],
[0, 0, 1, 2, 0, 1, 0, 1],
[0, 0, 1, 1, 1],
[0, 1, 2, 2, 2, 0, 1, 2],
[4, 3, 4, 3, 3, 4, 4, 4, 3, 3, 4, 4],
[3, 3, 4, 3, 3, 4],
[3, 4, 4, 4, 4, 4, 3, 4]
```

The word composition were chose such that there are two distinct topics, one consists of word 0, 1, 2 and the other consists of 3 and 4. Document 0 is an even mixture of both topics. Document 1-3 only have one topic while documents 4-6 only have the other topic. This construction should lead to a predictable topic model.

Applying our LDA implementation for 10 iterations results in the following distributions:

```
Variational parameters for topic word distribution:
Topic 0: [0.375     0.375     0.25      0.        0.       ]
Topic 1: [0.        0.        0.        0.4137931 0.5862069]]

Document-topic distributions for each document:
Topic 0   Topic 1
[0.5         0.5]
[1.          0. ]
[1.          0. ]
[1.          0. ]
[0.          1. ]
[0.          1. ]
[0.          1. ]
```

As a sanity check, we also applied the LDA algorithm in `sklearn` package and obtained the following results. `sklearn` LDA is implemented with variational Bayes rather than Gibbs sampling. This explains the discrepancy between this output and our implementation. Nevertheless, we can observe that the values are close and exhibit the same trends.

```
Variational parameters for topic word distribution:
Topic 0: [ 9.48361278,  9.48345025,  6.47634509,  0.51773044,  0.52432522]
Topic 1: [ 0.51638722,  0.51654975,  0.52365491, 12.48226956, 17.47567478]

Document-topic distributions for each document:
Topic 0      Topic 1
[0.49810604, 0.50189396]
[0.94420597, 0.05579403]
[0.91633138, 0.08366862]
[0.94415911, 0.05584089]
[0.03860094, 0.96139906]
[0.07170956, 0.92829044]
[0.05574671, 0.94425329]
```

4.3 Real Dataset

We used the 20 Newsgroup dataset for evaluating the model performance. The 20 Newsgroup dataset is a collection of newsgroup email documents. Figure 5 is an example of one of the samples.

The original dataset consists of documents from 20 topics distributed as shown in figure 6.

In order to get rid of interfering information, We used NLTK to apply the following preprocessing procedures:

- Remove email headers, footers, and quotes to other emails
- Remove punctuation and convert to lower case

```
From: v064mb9k@ubvmsd.cc.buffalo.edu (NEIL B. GANDLER)
Subject: Need info on 88-89 Bonneville
Organization: University at Buffalo
Lines: 10
News-Software: VAX/VMS VNEWS 1.41
Nntp-Posting-Host: ubvmsd.cc.buffalo.edu


 I am a little confused on all of the models of the 88-89 bonnevilles.
I have heard of the LE SE LSE SSE SSEI. Could someone tell me the
differences are far as features or performance. I am also curious to
know what the book value is for prefereably the 89 model. And how much
less than book value can you usually get them for. In other words how
much are they in demand this time of year. I have heard that the mid-spring
early summer is the best time to buy.

                                Neil Gandler
```

Figure 5: sample from 20 Newsgroup dataset

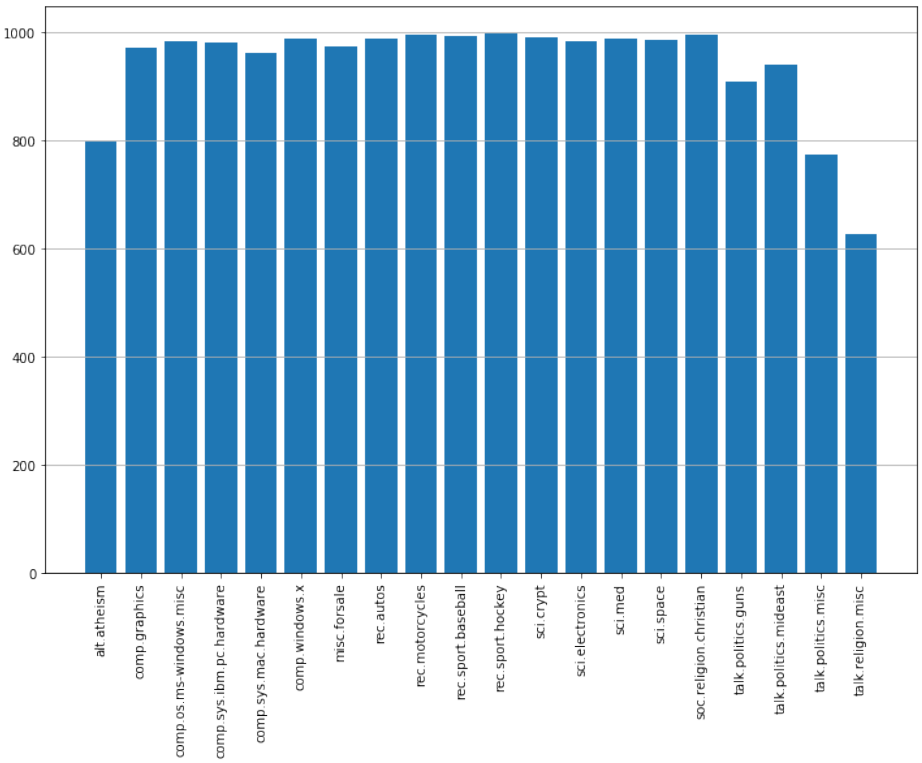


Figure 6: topic distribution of the 20 Newsgroup dataset

- Remove common English stop words
- Apply Porter stemmer to normalize English words

To make the classification task more straight-forward, we combined the computer related topics (topic index 1, 2, 3, 5) and recreation related topics (topic index 7, 8, 9, 10) respectively, making it a binary classification task: distinguishing between two large topics. The resulting dataset consists of 3928 computer related documents and 3979 recreation related documents.

4.4 Results

We performed experiments on our preprocessed computer-vs-recreaction documents which is a subset of the 20 Newsgroup dataset. For implementation of NMF and LSA algorithms, we utilized the sklearn library. Furthermore, in order to adapt to sklearn API input, we transformed the raw text into a matrix of word count. However, for our own LDA implementation, we still used equivalently the raw text since it is required by our Gibbs sampling algorithm.

For NMF, LSA, and LDA, we first run these topic modeling algorithms on the entire corpus to obtain the respective document features. Then we used the document features as input to classifiers. Their corresponding topics were used as class labels. The SVM and MLP classifiers in sklearn package were used to perform document classification. For fairness, all the algorithms were tested using the same train-test split of documents.

Using the following parameters for our models: alpha=1.0, beta=1.0, iterations=5, we present the accuracy on the test set of document classification:

	NMF	**LSA**	**LDA (sklearn)**	**LDA (ours)**
SVM	0.578	0.892	0.496	0.567
MLP	0.606	0.920	0.516	0.573

Table 1: testing accuracy with document word count as input

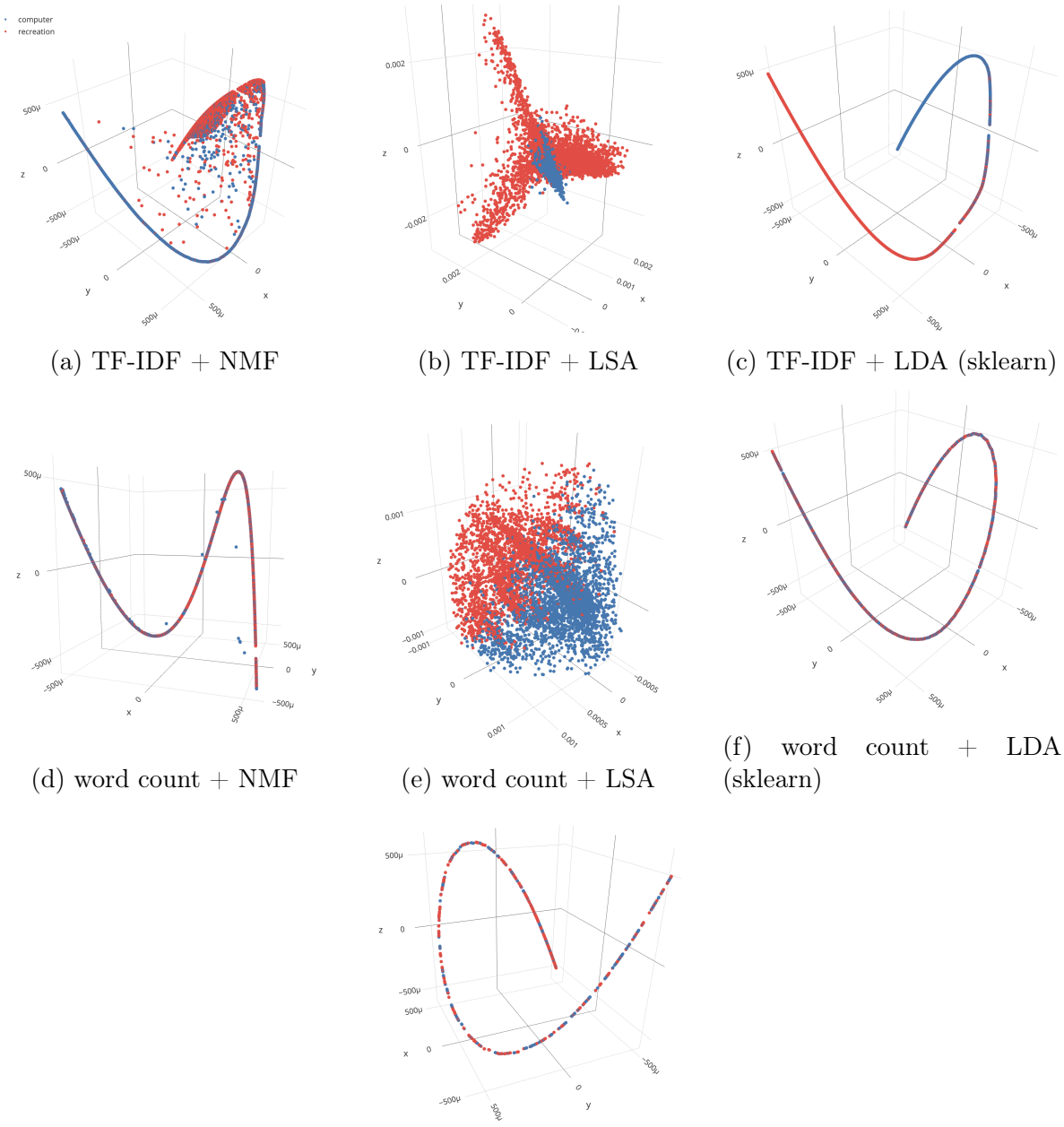
Additionally, we experimented with TF-IDF embedding that only preserves top 5000 most frequent words. However, our LDA implementation does not accept TF-IDF embedding, here we present accuracy results on the other models:

	NMF	**LSA**	**LDA (sklearn)**
SVM	0.877	0.905	0.917
MLP	0.881	0.938	0.919

Table 2: testing accuracy with TF-IDF document embedding

We also produced visualizations to help understand how the topic model representation separates the two class of documents. Spectral embedding was performed on the output of the three topic models. This maps the representation to 3 dimensions. This resulted in the plots shown in Figure 7. Each of the plots shows the spectral clustering result of the 7907 documents. Blue stands for computer related topic and red stands for recreation. These figures were created using Plotly.

As we can see from the figure, the same algorithm produces similar kind of shape. But using different vectorization (word count vs. TF-IDF) affects the separability greatly. But LSA is an exception as it produces distinctly separable clusters for both vectorization methods. We also observe that our implementation visualization shares the same characteristics as sklearn LDA.



(a) TF-IDF + NMF

(b) TF-IDF + LSA

(c) TF-IDF + LDA (sklearn)

(d) word count + NMF

(e) word count + LSA

(f) word count + LDA (sklearn)

(g) word count + LDA (ours)

Figure 7: Document features visualized using 3D spectral clustering

4.5 Result Discussion

Using word count generally leads to worse performance in classification accuracy. Transforming word count to TF-IDF can improve the classification accuracy across board.

We also noticed that in this experiment, LDA is not necessarily the best performing algorithm of the three. One possible reason could be that LDA is not the most suitable algorithm to extract document feature for classification. But rather LSA, as an SVD-based algorithm, can

produce very effective features for later classification tasks. Furthermore, we discovered that TF-IDF embedding is an effective technique for feature extraction. It significantly improved performance over word count vectorization.

All of these performances could be improved by parameter tuning. We have not performed this task in our experiment. All the parameters used were kept as default of the software package.

5 Conclusion

In this project, we created a Collapsed Gibbs Sampling based LDA implementation. Through experiments, we found out that our implementation have similar performance to sklearn implementation on document classification task, if given similar input.

LDA is a very specific model for text topic modeling. It poses many assumptions on how the text data is generated. Therefore, there is not much potential of generalizing it for other kinds of data.

Although LDA is one of the most widely used algorithm in text analysis, it has its limitations. Fundamentally, LDA assumes a bag-of-words model of documents. Therefore, the sequential text information is not utilized. Also, the number of documents is a predefined parameter in LDA, human users can not always have a good prediction on actual number of topics in documents.

5.1 Future Works

For future work, we want to further optimize the LDA implementation to pure C++ version. From the Cython optimization process, even though we can successfully static typed the parameters for the inference function and achieved 4 time performance improvement, we found that we could not apply the Cython nogil and parallel features to our Cython function. It is due to the dependency of variables for a iteration. We always need the document_topic distribution and the topic_word distribution from the previous iteration to generate new topic for words. To fully enjoy the efficiency of C++ language, we need to create all the data structures for counters in C++. We propose this will increase the performance significantly.

References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[2] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[3] W. M. Darling, "A theoretical and practical implementation tutorial on topic modeling and gibbs sampling," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 642–647, 2011.

[4] D. M. Blei, "Probabilistic topic models," *Commun. ACM*, vol. 55, pp. 77–84, Apr. 2012.

[5] I. Yildirim, "Bayesian inference: Gibbs sampling," *Technical Note, University of Rochester*, 2012.

[6] A. Kristiadi, "Gibbs sampler for lda."