Where Does LDA Sit for GitHub?

Xukun Wang
Department of Computer Science
University of British Columbia (Okanagan)
Kelowna, Canada
wxkace@hotmail.com

Matthias Lee
Department of Computer Science
University of British Columbia (Okanagan)
Kelowna, Canada
matthiaslmz@hotmail.com

Angie Pinchbeck
Department of Computer Science
University of British Columbia (Okanagan)
Kelowna, Canada
angie.pinchbeck@ubc.ca

Fatemeh H. Fard
Department of Computer Science
University of British Columbia (Okanagan)
Kelowna, Canada
fatemeh.fard@ubc.ca

***Abstract*—A lot of research in Software Engineering (SE) automatically extract topics of the text data and use the results directly or as a feature for a machine learning method. Research has shown that the majority of studies in SE use Latent Dirichlet Allocation (LDA) as the topic modeling approach. Similarly, there is a lot of work that apply LDA on GitHub data. However, there is no study that explores whether LDA is a good choice compared to other algorithms, nor is there any to investigate the effects of specific pre-processing steps on its performance. In this paper, we explore a large dataset of GitHub repositories and apply two main topic modeling algorithms, LDA (3 variants) and Non-Negative Matrix Factorization (NMF), in several experiments with different experimental settings. The results show that LDA results in a higher coherence score compared to NMF. However, care should be taken in the choice of LDA algorithm, setting its parameters, and the text pre-processing steps. The results of this paper benefit SE researchers who apply intelligent techniques using LDA.**

Keywords-Software engineering; Topic modeling; Software analysis

I. Introduction

One branch of Software Engineering (SE) research attempts to improve software development by analyzing software repositories [1]. SE data is mostly unstructured data and includes text. Many researchers use topic modeling to support SE tasks such as document clustering, feature location, and bug prediction [1]. Topic modeling is an unsupervised machine learning approach that automatically extracts the topics from a corpus of text [2]. It can benefit SE tasks as they require no training data to find the topics' distribution [1].

One of the main resources used by SE researchers in which topic modeling is applied is GitHub, the world's leading software development platform with more than 31 million developers and 100 million repositories [3-5]. Other than source code, commits, users, and issues, each project on GitHub has a readme file and a project description. The repositories' description on average contains of 15 words and is analyzed in many works as part of the SE research [6]. Many studies use Latent Dirichlet Allocation (LDA) to extract topics from GitHub repositories. Chen et al. mention that 66% of their 167 surveyed articles use LDA in SE research [1]. They report that the majority of their surveyed articles use LDA or LDA variants in their work. Although LDA is a widely used topic modeling algorithm [7], LDA is best suited to long text documents on large corpora and requires specific processing steps if applied on short text [8]. Research has shown that other topic models such as Non-Negative Matrix Factorization (NMF) are performing better on short text with an average length of 14 terms in each document in other domains [9, 10]. However, LDA is still widely used in the SE research community to extract the repositories' topics.

Considering the popularity of using LDA in SE research, there is no study that evaluates the results of LDA for describing GitHub repositories compared to other algorithms. Likewise, there is no study that evaluates the effects of text pre-processing steps when applying LDA on GitHub repositories' descriptions. The existing research applies topic modeling on a small dataset from GitHub, with 10,000 projects as the largest sample size [6] with purposes other than evaluating LDA or the effects of pre-processing on topics' coherence. The goal of this work is to experiment and evaluate the results of two main topic modeling algorithms, LDA and NMF [11], on GitHub repositories' descriptions (98 million before pre-processing) to characterize the following questions. We use topic coherence – a topic modeling evaluation metric – to compare the results.

- Question 1: Which of the LDA or NMF produces a better result for GitHub repositories' descriptions?
- Question 2: How does the text pre-processing steps affect the results of LDA?

Answer 1: LDA produces better coherence scores compared to NMF. This is confirmed by the manual inspection of the extracted topics and the results of Mann-Whitney u-test.

Answer 2: The choice of text pre-processing steps directly affects the coherence of the extracted topics. There are some regular tasks such as stemming in the text analysis that has an adverse effect on GitHub topic modeling using LDA. We provide suggestions based on our experiments to improve the outcomes when LDA is applied on GitHub descriptions.

The rest of the paper is organized as follows: related works are reviewed in Section II. Section III is dedicated to study set up and methodology, followed by results in Section IV. We will discuss the limitations and conclusions in sections V and VI.

II. Related Works

A. Topic Modeling

Topic modeling is used for the automatic extraction of dominant topics from a corpus of text documents [8]. In this approach, a number of topics are extracted automatically as a list of words. Some algorithms assign a weight to each word in the list. An important task in this modeling is setting the number of

978-1-7281-4136-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ASEW.2019.00037

topics – denoted by k – and the number of words required for each topic. To identify the number of topics, coherence score is calculated for a range of k and the number that produces a better score with less shared words among various topics is chosen. An expert can label each topic based on the topic's list of words.

One of the widely used topic modeling algorithms is LDA [7] which suffers from the sparsity of word co-occurrence in short text [2]. Researchers have developed other algorithms for short text, such as Biterm Topic Modeling (BTM) [2], Dirichlet Multinomial Mixture Model (DMM) [12], and algorithms based on Non Negative Matrix Factorization (NMF) [11]. NMF model learns the topics by decomposing the term-document matrix into two low rank factor matrices [9, 13]. NMF-based models have been applied to various datasets in other domains [14, 15].

B. *Topic Modeling on GitHub*

Topic modeling is used in SE research for various tasks such as data-driven requirement engineering [16], code quality [17], and surveying articles about software engineering [1, 18]. Ray et al. [17] apply the LDA algorithm on the text describing GitHub projects in order to identify the project domains for 729 projects as part of their code quality study. Mohammad et al. [19] use LDA to gain more insights about 8,517 GitHub pull requests made to 78 base projects. Sharma et al. [6] apply LDA-GA on readme files from 10,000 GitHub projects with more than 20 stars to categorize the repositories. Naoki [20] uses LDA to build a recommender system for GitHub, which is applied on 120,867 repositories that include only ID, name, and date.

Our work studies the effectiveness of LDA for GitHub. Unlike other works, we are interested to evaluate LDA rather than applying it to study GitHub. Therefore, we use a topic modeling evaluation metric i.e. coherence score to compare the outcomes [9]. Topic coherence evaluates how much the top terms in a topic are semantically related to each other [21, 22].

III. Study Setup And Methodology

A. *Data Collection*

Our dataset is a large open source dataset from GHTorrent [5]. GHTorrent monitors the GitHub public event timeline frequently, and collect data such as repositories and their features, users, and projects. Our collected data includes the information about 98,453,690 repositories (as of Nov. 2018).

B. *Data Preparation*

We merged different tables to obtain the users' and repositories' information. All the repositories that are not the main branch of a project are filtered out. In order to reduce the number of repositories that are created by inexperienced users, such as school work projects, we only include repositories that have more than 100 watchers. By applying this filter, the popular repositories are extracted [6]. As the authors only know English, the repositories that have description text in English are chosen. We also remove records with empty description and duplicates. After these steps, the number of remaining records is 104,315.

C. *Experiment Design*

Question 1: Comparing LDA with NMF. To answer this question, we apply LDA and NMF on data that is processed using the steps described in section III.D. We apply the algorithms on different number of topics varying from 1 to 100 and report the highest coherence score to compare the results.

In addition, we follow the guidelines in [9] and manually investigate the results to assess whether the list of words for each topic is meaningful and coherent. The manual inspection is done by one computer science expert with more than ten years' experience, and a graduate student pursuing a data science career. This inspection is also used to label each topic.

Question 2: Study the effect of text pre-processing on LDA algorithms. To answer this question, we empirically study the effect of various pre-processing steps, e.g. stemming the words and using bigrams and trigrams, on the coherence score. In addition, we compare the results of the original LDA, LDA with parallel implementation, and LDA with Gibbs Sampling. In each case, the effects of the changes in the alpha hyperparameter as 'symmetric', 'asymmetric', or 'auto' is recorded.

D. *Basic Text Pre-Processing*

We refer to the following steps as the basic text pre-processing. First, the text data is changed to lower case, split into single words, and the punctuations are removed. The bigrams (common phrases with two terms) are also added to the tokens. We use a list of common English stop words from NLTK package that should be removed from the text. Stop words are the common words that appear frequently and are useless in analyzing the text. Finally, we normalized the words using stemming (e.g. 'developed' to 'develop') and transformed them to their root word with lemmatization (e.g. 'saw' to 'see'). There are guidelines that using Term Frequency (TF) Inverse Document Frequency (IDF) penalizes the tokens in short texts too much [23], while higher quality results can be achieved using raw frequencies [24]. As a result, we only use TF.

E. *Topic Modeling*

Topic modeling requires preparing a document term matrix. In our study, the descriptions are the documents and the terms are the words in each description. A dictionary is created first by assigning a unique integer ID to each unique token with its TF. This dictionary is then converted to bag-of-words, which creates a document term matrix. This matrix is the input of the LDA and NMF algorithms. We use Gensim and sklearn packages with word embedding (word2vec). Following other studies in SE, the number of words is set to 20 for each topic [6].

IV. Results

A. *LDA versus NMF*

The highest coherence score achieved from LDA is approximately 0.37. The topic coherence increases slightly as the number of topics is increased from 1 to 100. We can observe this effect on the created list of topics. For example, when the algorithm produces 100 topics, it is noted that topics such as Swift, Game engines, and Deep learning appear in the list. When we apply NMF, topic coherence increases as the number of topics is increased. However, it stays around 0.34 as the highest. Our manual inspection of the results for 20, 40, 60, 80, and 100 topics show that a lot of words are shared among different topics produced by NMF, but, LDA generates more distinct topics. Our experiments show that all LDA versions outperform NMF. This is confirmed by applying the Mann-Whitney u-test [25] to compare the significance of the differences in coherence scores. We report an example of the u-test results for coherence scores of datasets with 16 programming languages and stemmed words: The coherence scores differ significantly (Mann–Whitney U = 14, n1 = n2 = 11, $P < 0.05$ two-tailed).

B. *Improving LDA Outcomes by Befining Text Pre-processing*

In this section, we report the results of performing more experiments with LDA to identify how we can improve its outcomes for GitHub. First, we calculate the coherence score when the number of topics is changing in the range of [100-1,000] with steps of 50. The dataset has more than 100,000 records, and we expect better representation of the corpus as the number of topics is increased. The score rises to 0.49 as the number of topics increases to 1,000. However, this number might not be suitable for applications such as text classification, which tries to discern only a few classes for the data.

Therefore, we investigate the effect of slightly improving the pre-processing on the topic coherence by applying the following steps in addition to the basic text pre-processing (described in section III.D). The numerical numbers are removed and only alphabetical numbers are kept in the description text. We also calculated the frequencies of the terms and bigrams in the corpus and extended the list of stop words. Moreover, the tokens that appear less than 10 times or in more than 20% of the documents are removed. In addition, the trigrams are appended to the projects' description. Fig. 1 shows a comparison of the coherence scores when the pre-processing is refined slightly. As it is shown, the topic coherence reaches to 0.45 in the best case.

Finally, we add more GitHub (SE) specific filtering criteria based on the repositories' programming languages. We investigate the effect of these steps and stemming the text for NMF and each of the three LDA variants with changes in the alpha hyper-parameter. We explore the number of repositories for each of the programming languages and filter the ones that are only associated with one repository, leaving 83,169 entries with 198 programming languages. Among them, only 14 programming languages have more than 2,000 repositories: JavaScript with 22,447 repositories is the highest and Swift with 2,133 repositories is the lowest one. The next ranks for the number of repositories are for Scala and Coffee Script with 872 and 792 repositories respectively. The other programming languages have less than 750 repositories. Based on the number of available repositories for each language, we perform studies with data that includes 16 (>=784 repositories), 84 (>=10 repositories), and 198 programming languages (>1 repositories). We run several experiments on stemmed versus not-stemmed text and with different settings for the alpha hyperparameter in the three versions of the LDA algorithm.

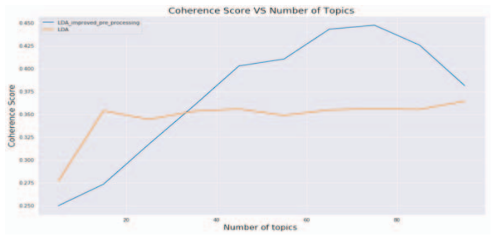

Fig. 1. Topic coherence of LDA for number of topics between 10-100 with steps of 10 applied on text with basic- and refined pre-processing steps

We also apply the NMF algorithm in each of the experiments. In each case, we calculate the optimal number of topics in the range of [1-200] based on the coherence score and report the highest score in the result. To the sake of brevity, we only report the results of the algorithms with highest scores in Table I. The row with bold font represents the highest coherence in each of the datasets filtered based on programming languages.

C. *Summary of Results and Discussions*

Answer 1: The result of our experiments show that LDA produces topics that have higher coherence compared to NMF. The manual inspection of the topics verifies this result. From the three variations of LDA, the original implementation of LDA has higher scores. In addition, LDA generates a probability that is associated with each term in the word list of each topic. The more relevant words are assigned a higher probability. This weight is beneficial in choosing the appropriate words that represent a topic, specifically when the refined pre-processing steps is applied. For example, LDA with 100 topics, results in 'Caffe' with weight 0.031 and 'Razor' with a weight of 0.0091 in the list of words for one topic that we labeled as deep learning.

Answer 2: The pre-processing is an important factor that can affect the results of LDA. When the list of terms is filtered by frequencies and bigrams and trigrams are used, the coherence scores increased. In our first experiment to answer Question 1, some words such as 'easy' appear in the list of topics' words and in many cases, they have high weights. Also, our results and inspections suggest that for GitHub, the regular list of stop words should be extended carefully in order to improve the coherence results. For example, the NLTK package includes 179 stop words for English language. However, when exploring the GitHub descriptions, this list should be extended to include more words. Moreover, stemming the data has a negative effect on the coherence score compared to when the terms are not stemmed.

The alpha parameter in the LDA algorithm is related to the Dirichlet Distribution and should be tuned. In the original implementation of LDA, in all cases, setting alpha to 'auto' results in higher coherence score. In the other two variants, the symmetric tuning produces higher coherence scores.

Finally, adding the programming languages to the repositories' descriptions is effective in producing results with higher coherence scores. The reason can be the appearance of the programming language in the list of words in each topic, and extraction of other words that are more related to each language. For example, one topic is related to Client-Server applications that are written in Java and the other topic consists of 'JavaScript', 'client', and 'script' keywords and is related to the client side applications with JavaScript.

In our experiments, we investigated the implementation of the coherence scores in different packages for the two algorithms. In some cases, the calculation of this score is simplified. We addressed this issue by using the same implementation of the coherence score in all of our studies. As a result, care should be taken when the coherence scores are compared for the same or different algorithms.

TABLE I. LDA AND NMF RESULTS

Model (# of languages, alpha)	**Stemmed/Unstemmed**	**Optimal # of topics**	**Coherence score**
ldaModel (16, 'symmetric')	Unstemmed	30	0.3725
ldaModel (16, 'auto')	**Unstemmed**	**30**	**0.3997**
ldaModel (16, 'asymmetric')	Unstemmed	30	0.3617
ldaMulticore(16, symmetric)	Unstemmed	20	0.3320
	Stemmed	20	0.3534
ldaMallet (16, symmetric)	Stemmed	16	0.3789
ldaModel (84, auto)	Stemmed	70	0.5054
	Unstemmed	**60**	**0.5137**
ldaMulticore (84, symmetric)	Stemmed	110	0.4108
	Unstemmed	140	0.4383
ldaMallet (84, symmetric)	Stemmed	200	0.3974
	Unstemmed	200	0.3890
NMF (84)	Stemmed	50	0.3471
	Unstemmed	120	0.3484
LdaModel (198, auto)	Stemmed	60	0.5029
	Unstemmed	**60**	**0.5059**
ldaMulticore (198, symmetric)	Stemmed	140	0.4081
	Unstemmed	110	0.4425
ldaMallet (198, symmetric)	Stemmed	200	0.3899
	Unstemmed	200	0.3902
NMF (198)	Stemmed	200	0.3633
	Unstemmed	200	0.3778

V. LIMITATIONS

Threats to validity: There are multiple implementations of the topic modeling algorithms, and various evaluation scores. To alleviate this problem, we use the popular packages used in many SE research [1, 6, 17, 18]. To mitigate the evaluation metric, we use the topic coherence score, which, according to [9], is one of the measurements that is consistent with experts' annotations and is used in many works.

The main limitation of our work is missing the comparison of other algorithms such as BTM and DMM. The main reason for leaving these algorithms out of this study is the rare usage of these algorithms in SE research and lack of a reliable implementation of the coherence results for these algorithms.

VI. CONCLUSION

LDA is used in many SE studies to extract the topics of the projects' descriptions on GitHub. However, its effectiveness and specific processing steps to improve LDA results have not been investigated. In this paper, we report the results of our experimental studies for applying LDA and NMF on a large dataset from GitHub. Our results show that LDA outperforms NMF in producing topics. The significance of the achieved coherence scores is confirmed by the results of Mann-Whitney u-test. Among the three LDA algorithms that we investigated, the original LDA generates higher coherent topics. We also argue that specific pre-processing steps should be applied to prepare the GitHub data for LDA algorithm rather than using the regular pre-processing steps for text analytics that are used in other domains. These steps have direct impact on the improvement of the LDA results.

In the future, we plan to expand our study to investigate more topic modeling algorithms for a wide range of tasks in SE research, such as software maintenance and evolution.

REFERENCES

[1] T. Chen, S. Thomas, and A. E. Hassan. "A survey on the use of topic models when mining software repositories." EMSE 21, no. 5 (2016): 1843-1919.

[2] C. Xueqi, X. Yan, Y. Lan, and J. Guo. "Btm: Topic modeling over short texts." *ITKDE* 26, no. 12 (2014): 2928-2941.

[3] GitHub. https://github.com/. Last Access: March 2019.

[4] GH Archive. https://www.gharchive.org. Last Access: March 2019.

[5] G. Gousios: The GHTorrent dataset and tool suite. MSR 2013: 233-236.

[6] Sh. Abhishek, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo. "Cataloging github repositories." In Proc. of the 21st Int. Conf. on Eval. and Ass. in Software Engineering, pp. 314-319. ACM, 2017.

[7] B. David M., A. Y. Ng, and M. I. Jordan. "Latent dirichlet allocation." *Journal of ML research* 3, no. Jan (2003): 993-1022.

[8] H. Liangjie, and B. D. Davison. "Empirical study of topic modeling in twitter." In *Proc. of the first workshop on SMA*, pp. 80-88. acm, 2010.

[9] C. Yong, H. Zhang, R. Liu, Z. Ye, and J. Lin. "Experimental explorations on short text topic mining between LDA and NMF based Schemes." *Knowledge-Based Systems*163 (2019): 1-13.

[10] Q. Xiaojun, Ch. Kit, Y. Ge, and S. J. Pan. "Short and sparse text topic modeling via self-aggregation." In *24th IJCAI*. 2015.

[11] L. Daniel D., and H. S. Seung. "Learning the parts of objects by non-negative matrix factorization." *Nature* 401, no. 6755 (1999): 788.

[12] Y. Jianhua, and J. Wang. "A dirichlet multinomial mixture model-based approach for short text clustering." In *ACM SIGKDD*, pp. 233-242. ACM, 2014.

[13] S. Tian, K. Kang, J. Choo, and C. K. Reddy. "Short-text topic modeling via non-negative matrix factorization enriched with local word-context correlations." In *Proc. WWW*, pp. 1105-1114, 2018.

[14] K. Hannah, J. Choo, J. Kim, C. K. Reddy, and H. Park. "Simultaneous discovery of common and discriminative topics via joint nonnegative matrix factorization." In *ACM SIGKDD*, pp. 567-576. ACM, 2015.

[15] K. Da, J. Choo, and H. Park. "Nonnegative matrix factorization for interactive topic modeling and document clustering." In *Partitional Clustering Algorithms*, pp. 215-243. Springer, Cham, 2015.

[16] M. Walid, M. Nayebi, T. Johann, and G. Ruhe. "Toward data-driven requirements engineering." *IEEE Software* 33, no. 1 (2016): 48-54.

[17] R. Baishakhi, D. Posnett, V. Filkov, and P. Devanbu. "A large scale study of programming languages and code quality in github." In *ESEC/FSE*, pp. 155-165. ACM, 2014.

[18] V. Garousi, and M. Mika. "Citations, research topics and active countries in software engineering: A bibliometrics study." *Computer Science Review* 19 (2016): 56-77.

[19] R. M. Masudur, and C. K. Roy. "An insight into the pull requests of github." In *MSR*, pp. 364-367. ACM, 2014.

[20] O. Naoki. "Collaborative topic modeling for recommending GitHub repositories." *Inf. Softw. Technol.* 83, no. 2 (2012): 110-121.

[21] D. Newman, L. J. Han, G. Karl, and T. Baldwin. "Automatic evaluation of topic coherence." In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 100-108. ACL, 2010.

[22] D. O'callaghan, D. Greene, J. Carthy, J. and P. Cunningham. 2015. An analysis of the coherence of descriptors in topic modeling. *Expert Systems with Applications*, *42*(13), pp.5645-5657.

[23] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, & H. Gall. "How can I improve my app? classifying user reviews for software maintenance and evolution." In *IEEE ICSME,* pp. 281-290, 2015.

[24] Ch. C. Aggarwal. Machine Learning for Text. 2018.

[25] J. D. Gibbons and S. Chakraborti. Nonparametric statistical inference. Springer, 2011.