On the Effectiveness of Labeled Latent Dirichlet Allocation in Automatic Bug-Report Categorization

Minhaz F. Zibran
University of New Orleans
2000 Lakeshore Drive, New Orleans, LA, USA
zibran@cs.uno.edu

ABSTRACT

Bug-reports are valuable sources of information. However, study of the bug-reports' content written in natural language demands tedious human efforts for manual interpretation. This difficulty limits the scale of empirical studies, which rely on interpretation and categorization of bug-reports. In this work, we investigate the effectiveness of Labeled Latent Dirichlet Allocation (LLDA) in *automatic* classification of bug-reports into a predefined set of categories.

CCS Concepts

•**Software and its engineering** → *Software libraries and repositories;*

Keywords

bug-report; automatic categorization; topic modelling; LLDA

1. INTRODUCTION AND MOTIVATION

Bug-repositories play a vital role in managing software defects over long term. Especially the long-lived software projects benefit from maintaining an accompanying bug-repository in activities such as bug triage, assignment of bug-fixing tasks, and keeping track of a history of known bugs. For purposes such as bug triage and bug-fixing task assignment, bug-reports in a bug-repository are typically kept characterized in several ways in terms of severity, priority, affected module, and open-close status of the bug-reports.

While these characterization of the bug-reports help in regular bug triage, additional means of characterizations are necessary for carrying out further analyses, given that bug-repositories are valuable sources of information used in software engineering research and development. For example, Zibran et al [19] studied bug-posts to determine the significances 22 factors, which affect the usability of published APIs (Application Programming Interfaces). They manually studied more than a thousand bug-posts, distinguished those bug-reports relevant to API usability issues, and characterized each of them based on what usability issues are reflected in the bug-posts. Due to unavailability of a reference-corpus, which could be used to train a NLP (Natural Language Processing) technique and automate the categorization process, they had to rely on their subjective analyses.

Indeed, manual investigation of thousands of bug-posts is a very tedious task. Devising a technique for automatic categorization of bug-reports (and documents written in natural languages) can save a lot of efforts and also allow large scale studies based on bug-report characterization. This work follows the study of Zibran et al [19]. In this paper, we present our ongoing work towards *automatic* bug-report categorization for large scale investigation of the significances of API usability factors reported earlier [18, 19]. We apply *Labeled Latent Dirichlet Allocation (LLDA)* [8], which is a probabilistic topic modelling technique emerged from the field of NLP. In particular, our work aims to make two contributions:

- We investigate the prospect and effectiveness of LLDA in automatically classifying bug-reports into a fixed set of predefined categories.
- We produce a curated data-set of bug-reports, which will benefit the research community in NLP tool evaluation and can be used as a training oracle in large scale studies, which rely on categorization bug-reports.

2. APPROACH

The set of 22 categories of API usability factors reported earlier [18, 19] is presented in Table 1. Further details about these 22 categories can be found elsewhere [18, 19]. In this work, we investigate the feasibility of LLDA in categorizing bug-reports into those closed set of 22 categories. Our study includes 1,138 bug-reports for three different projects namely `Eclipse`, `GNOME`, and `Python 3.1` (Table 2).

Table 1: Set of 22 categories of interest

Complexity	Concurrency
Naming	Conceptual correctness
Caller's perspective	Leftovers for client
Documentation	Multiple ways to do one thing
Consistency	Parameter and return
Error handling	Implementation vs. interface dependency
Reference chain	Memory management
API aging	Technical mismatch
API change	Factory pattern vs. constructor
Data types	Constructor parameter
Use of attributes	Code intelligibility

2.1 Phase 1: Oracle Development

The unavailability of a reference-corpus is a severe difficulty in applying LLDA for classifying bug-reports into those

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ICSE '16 May 14-22, 2016, Austin, TX, USA

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4205-6/16/05.

DOI: http://dx.doi.org/10.1145/2889160.2892646

Table 2: Bug-reports and selection criteria

Project name	Project component	Selection criteria: status	Selection criteria: Date until	# of posts: All	# of posts: U*
Eclipse	JDT Core	All *closed*	12 Oct'08	406	50
	JDT UI		22 Oct'08	260	37
GNOME	Libxml++	All	09 Jan'09	12	09
	glibmm	All	09 Jan'09	28	24
	gnomemm	All	09 Jan'09	16	08
	glade	All	09 Jan'09	05	01
	gnome-perl	All	09 Jan'09	09	06
	gnome-python	All	09 Jan'09	10	08
	gtkmm	All	09 Jan'09	44	33
	java-gnome	All	09 Jan'09	18	06
	libsigc++	All	09 Jan'09	19	12
	pyorbit	All	09 Jan'09	02	01
	pygobject	All	09 Jan'09	85	61
	pygtk	All	09 Jan'09	122	97
Python	All *closed* issues		15 Jan'09	102	75
(*here, U = API usability related)			**Total**	**1,138**	**428**

22 categories. Hence, we carefully go through each of these bug-reports and distinguish 428 of them as relevant to API usability. Then we further investigate each of these API-related 428 bug-reports. A bug-report is labeled with one or more of those 22 categories depending on whether the respective usability issues are found in the content and discussion of the bug-post. Thus, we develop an oracle of 428 bug-reports with known categorization.

2.2 Phase-2: Applying NLP

For applying LLDA, we use the `Stanford Topic Modeling Toolbox (TMT)` version 0.3.3 [1], which is a framework with core NLP algorithms including LLDA. We write Scala scripts making calls to appropriate APIs as required. Given a *training-corpus* of documents associated with multiple labels, LLDA is reported to be able to categorize documents by inferring the labels appropriate for each document in a seperate *inference-corpus* [8]. Using the oracle developed in phase-1, we measure accuracy of the categorization in terms of the precision, recall, and, F-score.

2.2.1 Document Preparation and Cleanup

Before applying LLDA, each of the bug-reports is transformed into a document containing only the title and concatenated comments from the bug-report. The documents are further cleaned up with the help of `TMT` at the time of applying LLDA. The *stop-words*, numbers, punctuation characters, and words having length of one or two characters are filtered out.

2.2.2 Invoking LLDA

A proper application of LLDA involves two stages: learning and inference [8]. At the learning stage, LLDA is applied on a training corpus (with documents having known labels) to develop a model, which is then used at the inference stage to infer labels of documents in inference-corpus.

We split the oracle of documents corresponding to the 428 bug-reports (distinguished in *phase-1*) into two disjoint sets. The 266 documents corresponding to the `GNOME` bug-reports constitute the *larger* corpus, and the remaining 162 documents corresponding to the `Eclipse` and `Python 3.1` bug-reports make the *smaller* corpus.

First, we train LLDA on the *smaller* and invoke it for inference on the *larger* corpus. Next, we use the *larger* corpus for training, and smaller one for inference. Then, we compare the results.

3. FINDINGS

Table 3 presents the accuracy of automated bug-report categorization when LLDA is trained on the *smaller* corpus and when the larger corpus is used for training. As we see, in Table 3, the accuracy in terms of precision, recall, and F-score substantially increased when the LLDA is trained on the *larger* corpus. The magnitude of increase in accuracy indicates that LLDA is potentially effective in automatic bug-report categorization if a sufficiently large corpus is used at the training stage.

Table 3: Accuracy of bug-report categorization

Training corpus	Accuracy of inference: Precision	Recall	F-score
Smaller	0.140	0.294	0.178
Larger	0.344	0.449	0.356

4. THREATS TO VALIDITY

Our study may be subject to human errors in the interpretation of the the bug-reports to produce the oracle. To minimize this threat, randomly picked 120 of the bug-report were verified for sanity check by two computer science practitioners other than the author. Exhaustive manual cross-check over all the bug-posts was not performed to minimize cost (time and effort) and that the random sanity check indeed confirmed correctness.

5. RELATED WORK

A number of earlier work [3, 7] attempted automatic binary classification to distinguish enhancement/feature requests from report addressing software defects. Bug-reports are also studied for detection of duplicate bug-reports [10, 12, 13], to facilitate bug triage [15], for predicting severity [5, 6], priority [14] of bug-reports, and estimating the time needed to fix a bug [2, 4, 16]. Attempts are also made for automatic summarization [9] of bug-reports and to develop hybrid techniques for bug-report classification [11, 17].

Although all of these work are more or less relevant to our work, ours significantly differs from them in terms of the objectives and approaches. For example, Somasundaram et al. [11] applied unsupervised LDA for automatic categorization of bug-reports, while we use *supervised labeled* LDA, which is an improved variant of LDA, particularly suitable in documents classification where a document can be classified to *more than one categories* in a closed set of categories [8], as such required for our work. Moreover, our work deals with an additional difficulty of not having an existing reference-corpus and we build one for our purpose.

6. CONCLUSION

In this work, we have investigated the prospect and effectiveness of LLDA-based topic modelling in automatic bug-report categorization. Based on a study on 1,138 bug-posts from bug-repositories for three different software projects, we derived that topic modelling can indeed be promising for such purposes when trained on large training corpus.

We, therefore, will extend this work by enlarging the oracle, which can be reused as a curated reference-corpus for tool evaluation and for conducting similar studies in larger scale. We also plan to carry out large scale studies to verify the results reported in earlier studies [18, 19], which relied on bug-report interpretation and categorization.

Acknowledgement: Thanks to Farjana Z. Eishita and Chanchal K. Roy for helping with random sanity check in the development of the oracle.

7. REFERENCES

[1] Stanford topic modelling toolbox, http://nlp.stanford.edu/software/tmt/tmt-0.3, last access: Jan 2016.

[2] P. Anbalagan and M. Vouk. On predicting the time taken to correct bug reports in open source projects. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 523–526, 2009.

[3] G. Antoniol, K. Ayari, M. Penta, F. Khomh, and Y. Guéhéneuc. Is it a bug or an enhancement? In *Proceedings of the Centre for Advanced Studies Conference (CASCON)*, pages 304–318, 2008.

[4] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of the International Workshop on Recommendation Systems for Software Engineering (WRSSE)*, pages 52–56, 2010.

[5] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals. Predicting the severity of a reported bug. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 1–10, 2010.

[6] T. Menzies and A. Marcus. Automated severity assessment of software defect reports. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 346–355, 2008.

[7] N. Pingclasai, H. Hata, and K. Matsumoto. Classifying bug reports to bugs and other requests using topic modeling. In *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)*, pages 13–18, 2013.

[8] D. Ramage, D. Hall, R. Nallapati, and C. Manning. Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 248–256, 2009.

[9] S. Rastkar, G. Murphy, and G. Murray. Summarizing software artifacts: A case study of bug reports. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 505–514, 2010.

[10] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the International Conference on Software Engineering*, pages 499–510, 2007.

[11] K. Somasundaram and G. Murphy. Automatic categorization of bug reports using latent dirichlet allocation. In *Proceedings of the India Software Engineering Conference (ISEC)*, pages 125–130, 2012.

[12] C. Sun, D. Lo amd S. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 253–262, 2011.

[13] C. Sun, D. Lo, X. Wang, J. Jiang, and S. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 45–54, 2010.

[14] Y. Tian, D. Lo, and C. Sun. Drone: Predicting priority of reported bugs by multi-factor analysis. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 200–209, 2013.

[15] D. Čubranić. Automatic bug triage using text categorization. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 92–97, 2004.

[16] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, page 1, 2007.

[17] Y. Zhou, Y. Tong, R. Gu, and H. Gall. Combining text mining and data mining for bug report classification. In *Proceedings of the International Conference on Software Maintenance and Evolution(ICSME)*, pages 311–320, 2014.

[18] M. Zibran. What makes APIs difficult to use? *J. Comp. Sci. Netw. Sec.*, 8(4):255–261, 2008.

[19] M. Zibran, F. Eishita, and C. Roy. Useful, but usable? factors affecting the usability of APIs. In *Proceedings of the International Working Conference on Reverse Engineering (WCRE)*, pages 151–155, 2011.