entropy

MDPI

Article

En-LDA: An Novel Approach to Automatic Bug Report Assignment with Entropy Optimized Latent Dirichlet Allocation

Wen Zhang [1,*,†], Yangbo Cui [1,†] and Taketoshi Yoshida [2,†]

1 Research Center on Data Sciences, Beijing University of Chemical Technology, Beijing 100039, China; 2014500006@mail.buct.edu.cn
2 School of Knowledge Science, Japan Advanced Institute of Science and Technology, 1-1 Ashahidai, Nomi, Ishikawa 923-1292, Japan; yoshida@jaist.ac.jp
* Correspondence: zhangwen@mail.buct.edu.cn; Tel.: +86-10-6419-7040
† These authors contributed equally to this work.

Academic Editor: Kevin H. Knuth
Received: 6 February 2017; Accepted: 14 April 2017; Published: 25 April 2017

Abstract: With the increasing number of bug reports coming into the open bug repository, it is impossible to triage bug reports manually by software managers. This paper proposes a novel approach called En-LDA (Entropy optimized Latent Dirichlet Allocation (LDA)) for automatic bug report assignment. Specifically, we propose entropy to optimize the number of topics of the LDA model and further use the entropy optimized LDA to capture the expertise and interest of developers in bug resolution. A developer's interest in a topic is modeled by the number of the developer's comments on bug reports of the topic divided by the number of all the developer's comments. A developer's expertise in a topic is modeled by the number of the developer's comments on bug reports of the topic divided by the number of all developers' comments on the topic. Given a new bug report, En-LDA recommends a ranked list of developers who are potentially adequate to resolve the new bug. Experiments on Eclipse JDT and Mozilla Firefox projects show that En-LDA can achieve high recall up to 84% and 58%, and precision up to 28% and 41%, respectively, which indicates promising aspects of the proposed approach.

Keywords: automatic bug report assignment; bug resolution; entropy measure; Latent Dirichlet Allocation

1. Introduction

One of the most compelling aspects of OSS (Open Source Software) is that they are developed predominantly based on voluntary contribution from geographically-distributed software developers without rigorously-controlled planning and management as that found in proprietary software development [1]. To succeed in development, OSS developers need to collaborate with each other in the entire life cycle of OSS. Community-intensive development, which is explained as collective effort and group influences on software development, is becoming an important characteristic of OSS development. Members of an OSS community share a well-developed and repeatedly emphasized set of values and ideology with a sense of obligation, and they keep an intrinsic connection with other members in the community [2].

Open bug repository, which is often called an issue tracking system, is widely adopted for software projects to support software development. Open source software projects adopt open bug repositories to support its development and maintenance in managing bugs. With open bug repository, geographically-distributed software developers and users report bugs of the software by submitting

bug reports to the repository. In this way, open source software is iteratively developed and the quality of the produced software can be improved [3].

The large number of new bug reports submitted to open bug repository increase the burden of bug triagers. For instance, about 200 bugs are filed to the Eclipse bug repository per day near its release dates [4], and, for the Debian project, it is about 150 [5]. To make things worse, bug reports are always triaged manually, which is time consuming and error prone. About two person-hours per day have to be spent on this activity to triage Eclipse bugs [6]. Nearly 25% of Eclipse bug reports are reassigned [7] as inaccurate bug report assignment [8].

Although a bug was fixed by only one developer as recorded in the open bug repository, bug resolution is essentially a social process [9,10]. The collaborative action of developers invested in fixing the bug cannot be neglected. The bug tracking system is a platform facilitating bug resolution through developersąf coordination [8]. Taking the bug report #18994 (https://bugs.eclipse.org/bugs/show_bug.cgi?id=18994) from the Eclipse JDT project as an example, "Eric Gamma" submitted this bug report initially and 27 comments on how to fix the bug were posted by five persons. In order to build complicated software systems successfully, developers need to collaborate with each other to resolve bugs [11].

This paper proposes an entropy optimized topic model based recommendation approach, called En-LDA (Entropy optimized Latent Dirichlet Allocation) to recommend developers for bug resolution in collaborative behavior. This work is motivated and inspired by two observations obtained from Eclipse JDT and Mozilla Firefox projects. The first observation is that bug resolution is a collaborative activity among developers despite the fact that the bug was recorded as fixed by merely one developer. The contribution of those developers who posted comments following the bug report cannot be neglected. The second observation is that developers always participate in and contribute to bug resolution in the formation of technical clusters. That is to say, a software system usually involves multiple technical aspects and each of them is implemented by the developers in one or more artifacts [12]. Developers tend to participate in resolving different bugs of one or some of the technical aspects based on their own technical concerns.

Inspired by the above observation, En-LDA builds topic models of developers using their historical bug resolution records. Specifically, for each bug report, its natural language contents are extracted and preprocessed to train topic models. To decide the optimal number of topics inherent in the historical bug reports, we adopt entropy to measure the purity of the output of the LDA model. The basic idea is that we regard each bug report as a bag of words and measure entropy of each word given a bug report by using topics as the probabilistic labels of the word. Then, the entropy of words in all bug reports are aggregated to gauge the overall entropy of words given the distribution of words with respect to topics and the distribution of topics with respect to bug reports. With the entropy optimized LDA model, developers who actually participated in the bug resolution are extracted and mapped to the topics. Then, the association, which is a bilateral relationship between developers and trained topics, is established. When a new bug is coming into the open bug repository, En-LDA ranks the developers based on their interest and expertise with respect to the bug report. We evaluate the proposed approach with precision, recall and F_1 measure.

The remainder of this paper is organized as follows. Section 2 presents the background knowledge to understand the proposed approach. Section 3 proposes En-LDA with its details. Section 4 describes the experiments on Eclipse JDT and Mozilla Firefox bug data and explains the experimental results. Section 5 concludes this paper.

2. Background

2.1. Entropy

Entropy [13] is defined as the amount of information in a transmitted message in information theory (see Equation (1)), where p_i is the probability of the ith event in the transmitted message with m possible events:

$$Entropy = -\sum_{i=1}^{m} p_i log(p_i). \quad (1)$$

For instance, if the message is a sequence as {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, then the entropy of the message should be $-\frac{10}{10}log_2\frac{10}{10}$, that is, 0. However, if the message is a sequence as {1, 0, 0, 1, 0, 1, 1, 1, 1, 1}, then the entropy of the message should be $-\frac{3}{10}log_2\frac{3}{10} - \frac{7}{10}log_2\frac{7}{10}$, that is, 0.88. The larger the entropy value is, the more uncertainty contained in the message. This paper introduces entropy to measure the purity of information of the output of the LDA model. By setting a different number of topics for LDA, we measure the entropy of the output. The smaller the entropy is, the better is LDA to model the historical bug reports.

2.2. Open Bug Repository

When a developer or user encountered a bug when using open source software, he or she usually files the bug to the open bug repository in form of a bug report. Many open bug repositories (e.g., Bugzilla, JIRA, GNATS and Trac) have been adopted in open source projects. We explore the Eclipse JDT and Mozilla Firefox open bug repositories, which use Bugzilla for bug management. Since other repositories are similar to Bugzilla, we believe our approach can be generalized to them with minor changes.

2.2.1. Bug Report

In Bugzilla, bugs are stored in the form of bug reports, which consist of pre-defined fields, text description, attachments and dependencies. Pre-defined fields record basic attributes of a bug. Some attributes such as creation date and the reporter who files this bug are unchangeable. Other attributes may be changed over bug lifetime, such as product, component, priority and severity. Some attributes may be frequently modified by authorized persons, such as the assignee, the current state and the final resolution. In addition, for each bug, a list of persons who may be interested in it can be extracted from the cc (carbon copy) list.

The text description of a bug report refers to the natural language contents, including the title of this bug report, a full description of this bug, and comments posted by some developers. These textual contents provide us with abundant information through which we can gain a deep insight into the details of a bug. In this paper, we extract only the title and the full description contents of bugs to build topic models.

Besides the pre-defined categorical fields and textual contents, the bug reporters and developers may also upload attachments as non-textual information, such as a screenshot of erroneous behavior [6] or execution traces.

2.2.2. Bug Life-Cycle

During their lifetime, bugs go through a series of states. Figure 1 depicts the life-cycle of bugs in Bugzilla (see also: http://www.bugzilla.org/docs/tip/html/lifecycle.html).

When a new bug is filed, a bug report whose initial state is set to NEW is submitted to the repository. Once it has been triaged and assigned to a developer, its state is modified to ASSIGNED. If this bug has been closed, its state is set to RESOLVED, VERIFIED or CLOSED. In the meantime, the resolution to this bug is marked. If the resolution results in changing code base, this bug is marked as FIXED. When a bug is determined as duplicated to other bugs, it is set to DUPLICATE. If a bug will

not be fixed, or it is not an actual bug, it will be set to WONTFIX or INVALID, respectively. If a bug was once resolved but has been reopened, it is marked as REOPENED. In our study, we only consider the bugs whose final resolution is FIXED and final state is RESOLVED, VERIFIED or CLOSED. The state sequence tracking the activity of bugs can be extracted from bug repositories, such as the time a bug is filed, the time it is assigned to a developer, and the time it is resolved.

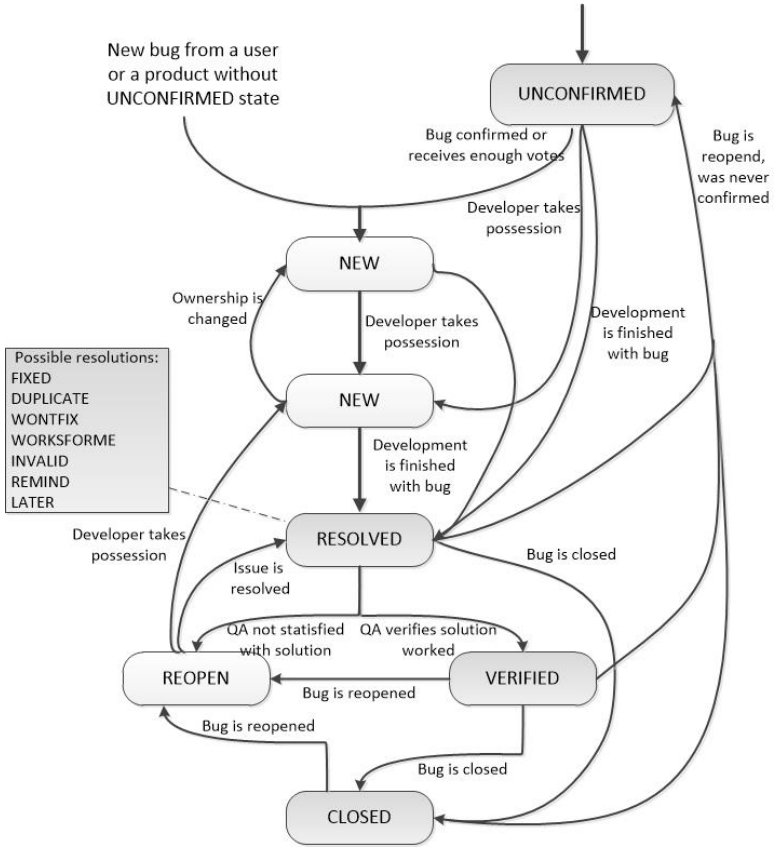

Figure 1. The life-cycle of a bug in Bugzilla (see also http://www.bugzilla.org/docs/tip/html/lifecycle.html).

3. The Proposed Approach

3.1. LDA Topic Model

Automatic topic extraction from documents is extensively studied in text mining and machine learning area [14]. The basic idea is to discover the latent structure of documents, which is inspired by words' co-occurrence in the corpus. For instance, LDA (Latent Dirichlet Allocation) [15], which is used in this study, models the documents in the same corpus as generated by some or all of the given K topics, while the words in each document come from the word distribution given by those topics. Using the discovered structure, documents can be linked through the topics to which they are assigned [16].

Topic models have several advantages in discovering structures from unstructured data. First, no training data are required for building topic models. Given unstructured data that we want to explore, it is very efficient to compute the LDA distributions of the data by using Gibbs sampling [17]. That is, we only need to properly set some parameters such as the number of topics, the number of iterations, and two hyper parameters, α and β, respectively. This makes topic models easy to use in practice. Second, topic models can operate on the raw, unstructured data without expensive data acquisition or preparation cost [18]. Third, topic models have been proven to be fast and scalable to large scale of data [19].

Because of these advantages, more and more fields such as social sciences [20] and computer vision [21] have benefited from topic models. In software engineering, the topic model has been successfully applied to solve many problems such as trend analysis in commit logs [22], bug localization [23], and code evolution [16,24,25].

In our study, the natural language contents of bug reports are the unstructured data we operated to discover topics, with which we used to group bugs together. Furthermore, we also group developers together based on their historical bug resolving activities on the grouped bugs.

3.2. Entropy Optimized LDA Model

One difficulty in using LDA for automatic bug report assignment is how to decide the optimal number of topics K of the historical bug reports to measure the expertise and interest of developers appropriately. In the prior work [15], the number of topics K is decided by heuristics or by trial and error. However, the performance of bug report assignment using the LDA approach can be further improved by optimizing the parameter K.

This paper proposes an entropy based measure to optimize the number of topics inherent in the historical bug reports as described in Equation (2). The basic idea is that we regard each bug report as a word bag and measure the entropy of each word given a bug report d_m using topics as the probabilistic labels of the word. Then, the entropy of words in all bug reports are aggregated to gauge the overall entropy of words given the distribution of words with respect to topics and the distribution of topics with respect to bug reports:

$$Entropy(K) = \sum_{m=1}^{M} \sum_{k=1}^{K} p(z_m = k|d = d_m) \left(\sum_{n=1}^{N_m} -p_{t,k,m} \ln p_{t,k,m} \right). \quad (2)$$

Here, M is the number of historical bug reports as the input for the LDA model. K is the number of topics inherent in the LDA model. N_m is the number of words in the bug report d_m. z_m is the topic for the bug report d_m, and z_n is the topic for the word w_n. Within a bug report, we normalize the sum of probabilities of word $w_n = t$ with respect to the topic $z_n = k$ being equal to one. That is to say, we regulate $p_{t,k,m} = \frac{p(w_n=t|z_n=k)}{\sum_{n=1}^{V_m} p(w_n=t|z_n=k),}$ where V_m is the size of word vocabulary in the bug report d_m. In other words, $p_{t,k,m}$ denotes the probability that the word w_n is t under the kth topic.

We can see from Equation (2) that when the number K is small, for instance, K is set as 1 in the extreme case, the probability $p(z_m = k|d = d_m)$ would be one since all of the bug reports have only one topic, and the probability $p(w_n = t|z_n = k)$ would be a very small number since all the words are under the same topic at this time. In this case, the overall entropy of the output of the LDA model would be very large. However, when the number of topics K becomes large, the probability $p(z_m = k|d = d_m)$ would be small since the number of topics z_m inherent in the bug report d_m would be large, and the probability $p(w_n = t|z_n = k)$ would be a very large number for a very small number of words; meanwhile, for most words, this probability would be very small since all the words would be distributed under different topics. In this case, the entropy of the output of the LDA model would also be very large. Thus, by using the entropy optimized LDA model, we first compute a series of entropy values with different parameters K and then select the optimal K with the minimum entropy measured by Equation (1). Based on the above analysis, it is necessary to tune an optimal number of topics K inherent in the historical bug reports to minimize the entropy of the output of LDA model as described in Equation (2).

3.3. Associating Developers and Topics

The association between developers and topics are based on the association between developers and bug reports. Similar to Anvik et al. [6], prior to associating developers and topics, we eliminated the developers who were not active or no longer available to participate in bug resolving activities.

The association between developers and bug reports is computed with a probabilistic model that calculates the probability of a developer being interesting and expertise in resolving the bugs. Given a new valid bug, the probability of a developer being a candidate to resolve can be expressed as the conditional probability $P(dev|d = d_m)$. It can be calculated by summing over probabilities that can be calculated by multiplying the probability of this bug belonging to a particular topic (i.e., $P(z_m = k|d = d_m)$) with the probability of this developer being a candidate for this topic (i.e., $P(dev|z_{dev} = k)$):

$$P(dev|d = d_m) = \sum_{k=1}^{K} P(z_m = k|d = d_m) \times P(dev|z_{dev} = k). \tag{3}$$

In Equation (3), $P(dev|z_{dev} = k)$, which is the probability of a developer being a candidate for resolving new bugs whose topic is k, is calculated in Equation (4). Given a topic, the probability of a developer being a candidate for the topic comprises two parts. The first part is the probability of the developer being active to participate in resolving bugs of this topic, i.e., the developer's interest in the bug report, denoted as $P(dev \rightarrow topic\ k)$. The second part is the probability of the developer being expertise in resolving bugs of this topic, denoted as $P(topic\ k \rightarrow dev)$. We set a weight denoted by θ to balance the interest and expertise a developer in the bug report:

$$\begin{aligned} P(dev|d_m = k) \quad &= \theta \times P(dev \rightarrow topic\ k) + \\ &(1 - \theta) \times P(topic\ k \rightarrow dev). \end{aligned} \tag{4}$$

Here, $P(dev \rightarrow topic\ k)$ is calculated as Equation (5):

$$P(dev \rightarrow topic\ k) = \frac{N_{dev,topic\ k}}{N_{dev}}. \tag{5}$$

$N_{dev,topic\ k}$ is the number of historical bugs that belong to the topic k and are resolved by this developer. Note that $N_{dev,topic\ k}$ is not an integer because one bug report may have more than one topic thus the proportion of each topic is a decimal fraction. N_{dev} is the number of historical bugs that are resolved by this developer. The probability of a developer having expertise in a given topic is calculated as Equation (6). $N_{topic\ k}$ is the number of historical bugs that belong to the particular topic k. Note that here $N_{topic\ k}$ is also not an integer:

$$P(topic\ k \rightarrow dev) = \frac{N_{dev,topic\ k}}{N_{topic\ k}}. \tag{6}$$

3.4. Recommendation

When a bug report is filed to the bug repository, a list of developers who are highly potential to participate in and contribute to resolving it are recommended according to the values calculated as Equation (3).

The values of the probabilities of a new bug belonging to all topics are calculated with the built topic models. Then, for each developer, the probability of he or she being a candidate to resolve the bug (i.e., $P(dev|d = d_m)$) is calculated. Next, according to the calculated probabilities of all developers, En-LDA ranks the developers in descending order. Finally, the Q developers with top probabilities are selected as the recommended developers.

4. Experiments

4.1. The Data

In this research, the historical bug reports are extract from the Eclipse JDT open bug repository (see: http://bugs.eclipse.org/bugs/) and Mozilla Firefox open bug repository (see: http://bugzilla.

mozilla.org/). We follow Guo et al. [26] to use 2.5 years as the time interval to collect bug reports in order to reduce possible changes in status and fields (such as products and components investigated in the paper) in the future. Thus, we extract the bug reports whose final resolution was FIXED and final state was the one of CLOSED, VERIFIED and RESOLVED. For the Eclipse JDT project, 2698 bug reports (from 10 October 2001 to 31 January 2009) are used in the empirical study. For the Mozilla Firefox project, 3005 bug reports (from 1 September 2008 to 31 July 2009) are used in the experiments.

We eliminate developers who participated in few bug resolution and retain those developers who contribute 90% bug resolution in the OSS projects. As a result, 31 developers participate 90% bug resolution in total in the Eclipse JDT project and in the Mozilla Firefox project, and 96 developers participate 90% bug resolution in total. Consequently, the numbers of developers used in Eclipse JDT and Mozilla Firefox projects decrease from 74 and 282 to 31 and 96, respectively.

Meanwhile, the bug reports, whose sizes of participants after the elimination decrease to 0, were removed as well. For the Eclipse JDT project, the size of training set (from 10 October 2001 to 31 January 2009) and testing set (from 1 February 2009 to 16 June 2010) decrease to 2448 and 110. For the Mozilla Firefox project, the size of the training set (from 1 September 2008 to 31 July 2009) decreases from 3005 to 3003 and the testing set (from 1 August 2009 to 31 August 2009) remain the same.

4.2. Experiments Setup

The preprocessing including tokenization, stop word elimination and stemming are conducted on the natural language contents. We obtain the stop-words from the USPTO (United States Patent and Trademark Office) patent full-text and image database (online: http://patft.uspto.gov/netahtml/PTO/help/stopword.htm). It includes about 100 usual words. The Porter stemming algorithm is used for English stemming processing which can be downloaded freely (online: http://tartarus.org/~martin/PorterStemmer/). Meanwhile, the participants of each bug report were extracted as well. The numbers of participants in resolving these bugs were 74 and 282 for Eclipse JDT and Mozilla Firefox projects, respectively. Given the activity logs of bug reports, we are able to find out the contributors of bug resolving. Then, affiliations between bug reports and participants are built as a graph.

Once the natural language contents of bug reports were extracted, En-LDA builds topic models for these bug reports. In order to train topic models with LDA, we the use Stanford Topic Modeling Toolbox (TMT, version 0.4.0) (see: http://nlp.stanford.edu/software/tmt/) to build topic models with setting the number of topics K, the number of iterations $R = 100$, and two hyper parameters $\alpha = 50/K$ and $\beta = 0.01$. The higher the α value, the higher the probability that a document (i.e., a bug report) will be associated with multiple topics. The higher the β value, the higher the probability that a topic will be associated with multiple words. Then, according to the bug-topic distributions, En-LDA maps each bug with multiple topics inherent within it with their probabilities.

For the hyper parameters and the number of iterations when using Gibbs Sampling for LDA model construction, we actually tune these parameters using exhaustive search for α from $30/K$ to $100/K$ with $10/K$ as the interval, and β from 0.01 to 0.1 with 0.01 as the interval and the number of iterations from 50 to 500 with 50 as the interval. With the hyper parameter composition α as as $50/K$, β as 0.01 and the number of iterations as 100, the proposed En-LDA approach has produced the best performance.

Figure 2 shows the entropy values computed using Equation (2) when we set different numbers of topics from 5 to 30 with increment as 1 for Eclipse JDT and Mozilla Firefox projects. We can see that when we set K as 20 for the Eclispe JDT project and as 23 for the Mozilla Firefox project, the entropy values become its minima. Moreover, the overall entropy values of the Eclipse JDT project are larger than that of the Mozilla Firefox project. We explain that the number of bug reports in the Eclipse JDT project are larger than in the Mozilla Firefox project. For this outcome, in the following experiments, we set the optimal K as for the Eclipse JDT project and the optimal K as 23 for the Mozilla Firefox project. For the outcome of the LDA model, we also tune a different number of K and find that the optimal K decided by entropy is the best choice.

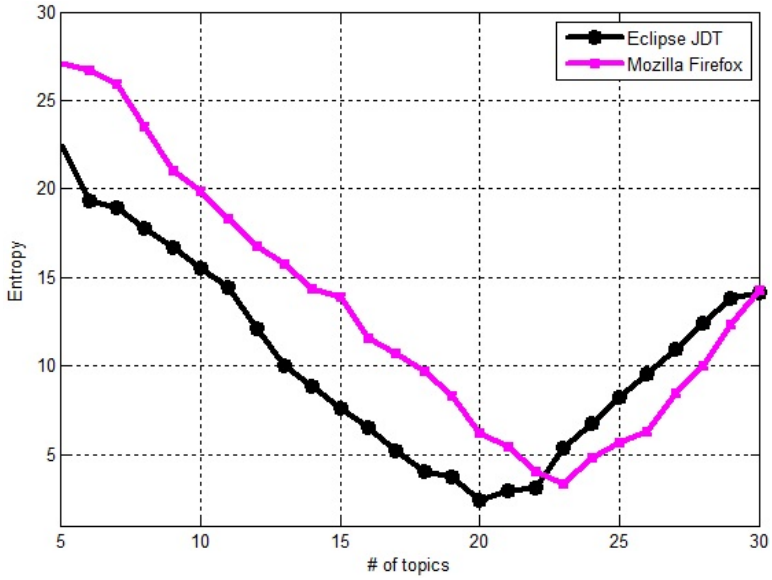

Figure 2. Entropy values in setting a different number of topics for Eclipse JDT and Mozilla Firefox projects.

Tables 1 and 2 show the top 10 words assigned to five of 20 topics for the Eclipse JDT and Mozilla Firefox projects, respectively. In LDA, topics are collections of words that co-occur frequently in the corpus. Table 1 shows four topics extracted from the Eclipse JDT bug report natural language contents. Each column entry presents a topic and its top 10 words that co-occur most frequently. We can see that topic 1 is roughly about the concern of *UI (User Interface) interactions*. Topic 2 is roughly about *project management*. Topic 3 is roughly about *debugging the dialog component*. Topic 4 is roughly about *errors on methods*.

Table 1. Top 10 Words Assigned to 4 of 20 Topics in Eclipse JDT.

Rank	Topic-1	Topic-2	Topic-3	Topic-4
1	menu	source	launch	error
2	action	package	debug	compiler
3	selection	folder	run	interface
4	view	jar	context	annotation
5	context	files	default	quick
6	show	create	config	warning
7	editor	src	resource	method
8	clean	explorer	dialog	override
9	open	path	remote	problem
10	add	copy	tab	missing

Table 2 shows four of the 23 topics extracted from the Mozilla Firefox bug report natural language contents. Each column entry presents a topic and its top 10 words that co-occur most frequently. We can see that topic 1 is roughly about the concern of the *method body*. Topic 2 is roughly about *error trace*. Topic 3 is roughly about *multimedia*. Topic 4 is roughly about *performance enhancement*.

Table 2. Top 10 Words Assigned to 4 of 23 Topics in the Mozilla Firefox project.

Rank	Topic-1	Topic-2	Topic-3	Topic-4
1	return	error	image	cache
2	const	fail	video	parser
3	null	log	background	time
4	string	unit	color	leak
5	static	dom	border	document
6	class	fix	media	content
7	type	check	frame	html
8	check	pass	element	event
9	fix	content	box	cycle
10	method	reply	canvas	thread

4.3. Evaluation

We used the historical bug resolution records in the open bug repository to calculate the precision and recall of En-LDA. For each historical bug report, En-LDA will recommend a number of developers to participate in resolving the bug. Notice that instead of finding out the actual bug fixers, we aim at recommending the developers who have a high potential to participate in and contribute to resolving newly filed bugs. Therefore, we evaluated our approach with the average value of recall other than that of precision. The values of precision and recall are calculated as Equations (7) and (8):

$$Precision = \frac{|\{dev_1, dev_2, ..., dev_K\} \bigcap \{Ground\ Truth\}|}{|\{dev_1, dev_2, ..., dev_K\}|}, \tag{7}$$

$$Recall = \frac{|\{dev_1, dev_2, ..., dev_K\} \bigcap \{Ground\ Truth\}|}{|\{Ground\ Truth\}|}. \tag{8}$$

Here, $\{dev_1, dev_2, ..., dev_K\}$ is the recommendation result for a new bug report, and $\{Ground\ Truth\}$ consists of real developers who participated in and contributed to resolving the new bug report.

4.4. Experimental Results

For the Eclipse JDT project, on average, two developers participate in each bug resolution in the testing set. For this reason, we vary the number of recommended developers, i.e., Q, from 1 to 5. For the Mozilla Firefox project, on average, five developers participate in each bug resolution. Thus, Q is varied from 1 to 7. Tables 3 and 4 present the average precision and recall of developer recommendation for the Eclipse JDT and Mozilla Firefox projects, respectively.

Table 3. The average precision and recall for the Eclipse JDT project. The numbers in bold indicate the best performances.

θ	Precision (%)/Recall (%)				
	Top 1	**Top 2**	**Top 3**	**Top 4**	**Top 5**
0	5/4	9/13	24/53	22/73	18/82
0.1	6/5	9/16	25/61	24/76	18/82
0.2	6/6	9/17	**28**/71	22/77	22/**84**
0.3	7/6	11/19	22/68	23/80	19/81
0.4	8/7	13/21	21/63	24/82	18/81
0.5	12/08	15/23	22/63	23/74	20/73
0.6	17/11	16/25	23/62	21/70	19/76
0.7	18/15	18/30	22/61	19/71	18/71
0.8	16/13	21/38	21/55	19/63	16/72
0.9	13/11	17/32	20/43	18/51	14/71
1	10/9	10/19	19/26	12/39	12/53

Table 4. The average precision and recall for Mozilla Firefox project. The numbers in bold indicate the best performances.

θ	Precision (%)/Recall (%)						
	Top 1	**Top 2**	**Top 3**	**Top 4**	**Top 5**	**Top 6**	**Top 7**
0	29/8	33/19	31/24	28/28	26/32	26/37	25/43
0.1	31/10	34/18	32/25	29/29	29/35	26/39	25/42
0.2	31/11	35/19	32/23	31/30	28/35	29/39	25/43
0.3	32/10	36/20	31/24	32/30	28/36	28/41	27/46
0.4	33/11	37/20	33/26	33/31	27/35	28/44	25/48
0.5	35/10	37/21	34/28	31/31	28/37	31/43	29/52
0.6	33/9	**41**/24	37/28	32/33	32/43	32/48	32/**58**
0.7	33/9	39/22	34/29	33/45	31/42	30/48	29/52
0.8	32/10	38/21	35/31	35/48	31/42	29/48	28/53
0.9	31/11	32/20	32/32	31/47	29/41	28/47	25/51
1	32/9	27/16	27/21	22/29	24/33	23/41	24/48

We can see from Table 3 that, for the Eclipse JDT project, the precision and recall peak at $\theta = 0.2$ (i.e., 28% and 84%, respectively, with the top 3 and top 5 recommended developers). For the Mozilla Firefox project (see Table 4), the average precision and recall peak at $\theta = 0.6$ (i.e., 41% and 58%, respectively, with the top 2 and top 7 recommended developers). Instead of finding out the actual bug fixers, we aim at retrieving developers who have a high possibility to participate in and contribute to resolving newly arrived bugs. This is similar to the interesting developers mentioned in [7]. A bug report's resolution is attributed to developers' collaborations other than a single developer's effort.

We can see from Tables 3 and 4 that, for the Eclipse JDT project, the parameter should be set as 0.2 and for the Mozilla Firefox project, the parameter θ should be set as 0.6 for the best performances in using En-LDA for bug report assignment. We explain the outcome as the number of bug reports and candidate developers for training and testing of the Eclipse JDT project being smaller than that of the Mozilla Firefox project. We draw the conclusion that if we have a large number of bug reports and candidate developers, the parameter θ should be set larger than 0.5. Otherwise, the parameter should be set as less than 0.5.

We hold that recall is a better measure than precision to gauge the performance of the proposed approach. The reason lies in the fact that the number of developers in ground truth is different from each other bugs. That is, different bug reports have a different number of developers participating in its resolution—for example, if a bug was resolved by a collaboration of two developers in the ground truth, and, in testing, we recommend five developers as potential resolvers. Then, the precision must be smaller than 40%. However, it is acceptable, for real practice, that the two developers were included in the five developers in the ground truth, especially in the case that more than 100 developers are involved in the whole OSS project. With this stand of point, we regard that, in terms of the average recall, our approach is promising in recommending potential bug resolvers. In fact, for the bug report assignment, what we are looking for is to find "enough" number of developers to make contributions rather than to find all correct developers as that in a traditional commodity recommendation. This is the reason why we use recall and precision for performance measure.

The different recommendation results between Eclipse JDT and Mozilla Firefox projects is due to two reasons. The first reason is that the average number of real developers is two for the former project, but it is five for the latter project. When the number of recommended developers is approximate to the number of real developers, the interest of developers should be considered. However, when the number of recommended developers is much larger than real developers, the developers' expertise is an important factor because, even if some recommended expert developers have little interest in the bug, there are still some expert developers who will be interested in resolving the bug.

The second reason is that the duration of Eclipse JDT is much shorter than that of the Mozilla Firefox project. When we use a data set with a short time period for training the topic models

in the proposed approach, it would be better to consider the developers' interest in a developer's recommendation. However, when the training data set lasts for a long period, the developers' expertise will be more helpful than interest in recommending the "right" developers to resolve the bug.

We admit that the proposed approach is sensitive to parameter θ. The way it affects the recommendation results varies for different bug repositories. We suggest that, prior to putting En-LDA into practice, some trials should be conducted to find the optimal θ. Once the optimal θ was produced, our approach to recommending potential bug resolvers is useful.

We also find some similar research to the paper as Anvik et al. [6], Xia et al. [27] and Canfora and Cerulo [28]. For Anvik et al. [6] and Xia et al. [27], what they take into account in their research is to predict the single fixer to fix the bug rather than a group of developers who may contribute to resolve the bug reports, and it is not surprising that the experiment results of Xia et al. [27] are much better than us with accuracy approximately 0.8, considering that bug report assignment is a more complicated problem than fixer prediction. Moreover, in our prior work [5], we find that multi-label classification is not good at bug report assignment although it succeeds in predicting the final fixer of the bug report. Canfora and Cerulo [28] use information retrieval to locate candidate developers for change requests. Their basic idea is that developers that have resolved similar change requests in the past are the best candidates to resolve the new one. We can see from Canfora and Cerulo [28] that, in the Mozilla project, their experimental results are comparable to us. However, they only consider one developer for recommendation because the bug tracking system assigns each change request to only one developer.

5. Conclusions

This paper proposes a novel approach called En-LDA to automatic bug report assignment by using entropy and the LDA model. The entropy measure is used to optimize the number of topics and the LDA model is used to capture the expertise and interest of developers on bug reports. The contribution of the paper lies in two aspects. First, we propose En-LDA for automatic bug report assignment. Second, we explore the Eclipse JDT and Mozilla Firefox open bug repositories to examine the performances of the proposed approach. The experimental results demonstrate that our topic models based approach can achieve high recall up to 84% and 58% with the top 5 and top 7 developers for Eclipse JDT and Mozilla Firefox, respectively. Moreover, the developers' interest and expertise on bug reports influence automatic bug report assignment in different ways. We argue that developers' interest can not be ignored in recommending developers for bug resolution.

Although experimental results have shown the promising aspects of our approach, we admit that this study is still on its initial stage. In the future, we will examine our approach on more open bug repositories to comprehensively evaluate our approach. It is obvious that when a new bug has been triaged and assigned to some developers, and if it is resolved by them, the topic models should be updated. We will consider this condition as well. In addition to these, we will take the relationships between topics into account. In view of the fact that topics may change over time [16,25], we will also take the evolution of topics to which developers pay attention into consideration.

Acknowledgments: This research was supported in part by the National Natural Science Foundation of China under Grants Nos. 71101138, 61379046, 91218301, 91318302, and 61432001; the Beijing Natural Science Fund under Grant No. 4122087; and the Fundamental Research Funds for the Central Universities (buctrc201504).

Author Contributions: Wen Zhang and Taketoshi Yoshida conceived and designed the experiments; Yangbo Cui performed the experiments; Yangbo Cui analyzed the data; and Wen Zhang wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Raymond, E.S. *The Cathedral & The Bazzar*; O'Reilly Media: Newton, MA, USA, 1999.
2. Bagozzi, R.P.; Dholakia, U.M. Open Source Software User Communities: A Study of Participation in Linux User Groups. *Manag. Sci.* **2006**, *52*, 1099–1115. [CrossRef]

3. Raymond, E.S. *The Cathedral and the Bazaar*, 1st ed.; O'Reilly & Associates, Inc.: Sebastopol, CA, USA, 1999.
4. Anvik, J.; Hiew, L.; Murphy, G.C. Coping with an open bug repository. In Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology Exchange (Eclipse '05), San Diego, CA, USA, 16–17 October 2005; pp. 35–39.
5. Wu, W.; Zhang, W.; Yang, Y.; Wang, Q. Time series analysis for bug number prediction. In Proceedings of the 2010 2nd International Conference on Software Engineering and Data Mining (SEDM), Chengdu, China, 23–25 June 2010; pp. 589–596.
6. Anvik, J.; Hiew, L.; Murphy, G.C. Who should fix this bug? In Proceedings of the 28th International Conference on Software Engineering (ICSE '06), Shanghai, China, 20–28 May 2006; pp. 361–370.
7. Anvik, J.; Murphy, G.C. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.* **2011**, *20*. [CrossRef]
8. Guo, P.J.; Zimmermann, T.; Nagappan, N.; Murphy, B. "Not my bug!" and other reasons for software bug report reassignments. In Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work (CSCW '11), Hangzhou, China, 19–23 March 2011; pp. 395–404.
9. Bertram, D.; Voida, A.; Greenberg, S.; Walker, R. Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. In Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work (CSCW '10), Savannah, GA, USA, 6–10 February 2010; pp. 291–300.
10. Wu, W.; Zhang, W.; Yang, Y.; Wang, Q. DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking. In Proceedings of the 18th Asian Pacific Software Engineering Conference, Ho Chi Minh, Vietnam, 5–8 December 2011; pp. 389–396.
11. Minto, S.; Murphy, G.C. Recommending Emergent Teams. In Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR '07), Minneapolis, MN, USA, 20–26 May 2007; p. 5.
12. Nguyen, T.T.; Nguyen, T.N.; Phuong, T.M. Topic-based defect prediction (NIER track). In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11), Honolulu, HI, USA, 21–28 May 2011; pp. 932–935.
13. Quinlan, J. Induction of decision trees. *Mach. Learn.* **1986**, *1*, 81–106. [CrossRef]
14. Griffiths, T.L.; Steyvers, M. Finding scientific topics. *Proc. Natl. Acad. Sci. USA* **2004**, *101*, 5228–5235. [CrossRef] [PubMed]
15. Blei, D.M.; Ng, A.Y.; Jordan, M.I. Latent dirichlet allocation. *J. Mach. Learn. Res.* **2003**, *3*, 993–1022.
16. Thomas, S.W.; Adams, B.; Hassan, A.E.; Blostein, D. Validating the Use of Topic Models for Software Evolution. In Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM '10), Timisoara, Rumania, 12–13 September 2010; pp. 55–64.
17. Phan, X.H.; Nguyen, C.T. GibbsLDA++: A C/C++ Implementation of Latent Dirichlet Allocation. 2007. Available online: http://gibbslda.sourceforge.net/ (accessed on 17 April 2017).
18. Thomas, S.W. Mining software repositories using topic models. In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11), Honolulu, HI, USA, 21–28 May 2011; pp. 1138–1139.
19. Porteous, I.; Newman, D.; Ihler, A.; Asuncion, A.; Smyth, P.; Welling, M. Fast collapsed gibbs sampling for latent dirichlet allocation. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08), Henderson, NV, USA, 24–27 August 2008; pp. 569–577.
20. Ramage, D.; Rosen, E.; Chuang, J.; Manning, C.D.; McFarland, D.A. Topic Modeling for the Social Sciences. In Proceedings of the NIPS 2009 Workshop on Applications for Topic Models: Text and Beyond, Vancouver, BC, Canada, 7–10 December 2009.
21. Barnard, K.; Duygulu, P.; Forsyth, D.; de Freitas, N.; Blei, D.M.; Jordan, M.I. Matching words and pictures. *J. Mach. Learn. Res.* **2003**, *3*, 1107–1135.
22. Hindle, A.; Godfrey, M.; Holt, R. What's hot and what's not: Windowed developer topic analysis. In Proceedings of the 2009 IEEE International Conference on Software Maintenance (ICSM 2009), Edmonton, AB, Canada, 20–26 September 2009; pp. 339–348.
23. Lukins, S.K.; Kraft, N.A.; Etzkorn, L.H. Bug localization using latent Dirichlet allocation. *Inf. Softw. Technol.* **2010**, *52*, 972–990. [CrossRef]
24. Linstead, E.; Lopes, C.; Baldi, P. An Application of Latent Dirichlet Allocation to Analyzing Software Evolution. In Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications, Kunming, China, 12–15 July 2008; pp. 813–818.

25. Thomas, S.W.; Adams, B.; Hassan, A.E.; Blostein, D. Modeling the evolution of topics in source code histories. In Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11), Honoluli, HI, USA, 21–22 May 2011; pp. 173–182.
26. Guo, P.J.; Zimmermann, T.; Nagappan, N.; Murphy, B. Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows. In Proceedings of the 32th International Conference on Software Engineering, Cape Town, South Africa, 2–8 May 2010; pp. 495–504.
27. Xia, X.; Lo, D.; Ding, Y.; Nguyen, T.N.; Wang, X. Improving Automated Bug Triaging with Specialized Topic Model. *IEEE Trans. Softw. Eng.* **2017**, *43*, 272–297. [CrossRef]
28. Canfora, G.; Cerulo, L. Supporting change request assignment in open source development. In Proceedings of the 2006 ACM Symposium on Applied Computing, Dijon, France, 23–27 April 2006; pp. 1767–1772.