Automatic Categorization of Bug Reports Using Latent Dirichlet Allocation

Kalyanasundaram Somasundaram
Department of Computer Science and Eng.
Anna University
kalyanceg@gmail.com

Gail C. Murphy
Department of Computer Science
University of British Columbia
murphy@cs.ubc.ca

ABSTRACT

Software developers, particularly in open-source projects, rely on bug repositories to organize their work. On a bug report, the component field is used to indicate to which team of developers a bug should be routed. Researchers have shown that incorrect categorization of newly received bug reports to components can cause potential delays in the resolution of bug reports. Approaches have been developed that consider the use of machine learning approaches, specifically Support Vector Machines (SVM), to automatically categorize bug reports into the appropriate component to help streamline the process of solving a bug. One drawback of an SVM-based approach is that the results of categorization can be uneven across various components in the system if some components receive less reports than others. In this paper, we consider broadening the consistency of the recommendations produced by an automatic approach by investigating three approaches to automating bug report categorization: an approach similar to previous ones based on an SVM classifier and Term Frequency Inverse Document Frequency(SVM-TF-IDF), an approach using Latent Dirichlet Allocation (LDA) with SVM (SVM-LDA) and an approach using LDA and Kullback Leibler divergence (LDA-KL). We found that LDA-KL produced recalls similar to those found previously but with better consistency across all components for which bugs must be categorized.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Management

Keywords

software bug triage, component recommendation, recommendation system

ISEC '12, Feb 22-25, 2012, Kanpur, UP, India.

1. INTRODUCTION

Bug repositories help software developers communicate about a software development project amongst themselves, and in open-source projects, with users of the software. The bugs reported by developers or users may include defects associated with deployed software as well as desired enhancements to the system. For larger projects, it is common that bug reports are categorized to allow efficient handling by the appropriate subset of the development team. For example, in the Eclipse Platform software development project,[1] bugs are assigned to one of 15 *components*; each component represents a category.

Incorrect categorization of bug reports in a project's bug repository can cause problems in the development process. Guo et al. describe how changes in the categorization of bug reports to components is positively correlated with excessive reassignments of bugs [9]. Reassignment of a bug between components or between personnel associated with a component is not always bad, but it does typically slow the time needed to fix the bug or complete the enhancement request.

To help improve the correct categorization of bug reports early in the bug reporting process, researchers have considered automatic approaches for assigning a bug report to a category (or component) based on the text describing the report. These approaches are typically referred to as component recommenders. di Lucca et al. applied a number of machine learning approaches for producing a component recommender and found that they could achieve an accuracy rate of 84% in a recommendation list of size one when using the SVM approach [8]. This result was achieved on a bug repository for a proprietary system in which there were only eight very distinct components. Anvik showed that an SVM-based classifier could achieve results of between 72% and 92% for a recommendation list of size three on a broader range of open source systems where the components were not as distinct [1]. A limitation of these previous SVM-based approaches is that the automatic categorization will work less well for components that do not have a large number of reports in the repository on which the recommender can be trained. When we applied an SVM-based technique very similar to Anvik's on a per-component basis to reports in the Eclipse Platform, we found a wide range of results, with the recall for various components varying from 48% to 98%.

In this paper, we investigate whether the use of Latent Dirichlet Allocation (LDA) [4] in conjunction with a machine learning algorithm might allow for automatic bug report categorization that performs more consistently across the com-

[1] `www.eclipse.org`, verified 2011/09/06

ponents for which recommendations are needed, even if the number of reports for each category varies widely in the repository. Specifically, we report on experiments applying different classification approaches on the LDA topic cluster to three open-source projects. The three variations we consider are LDA combined with the Kullback Leibler divergence algorithm (LDA-KL), SVM with LDA as the feature vector (SVM-LDA) and SVM with the feature vector populated by a TF-IDF algorithm(SVM-TF-IDF) to provide a comparison similar to previous work. The three open-source projects we consider are the Eclipse platform, Eclipse Mylyn,[2] and Mozilla.[3] We demonstrate that the most consistent results can be achieved with the LDA-KL method, which achieves a recall of 70 to 95%, an improvement over existing approaches.

We begin by comparing our work to earlier efforts in bug repository automation (Section 2). We then provide background information about bug reports and LDA needed to understand the approaches (Section 3). We then describe the three approaches (Section 4) before presenting our evaluation of each (Section 5). Section 6 discusses aspects of the approaches considered and Section 7 summarizes the paper and presents future directions.

2. RELATED WORK

There have been several efforts directed at easing the process of bug triage through automated mechanisms.

Many of these efforts have focused on automating the process of assigning a bug report to a particular developer (e.g., [7], [5], and [2]). Based on an evaluation of five open-source projects, Anvik showed it was possible to achieve precision rates ranging from 70% to 98% for one recommendation [3]. When specific teams work on bugs associated with a particular component and developers work on only one team, a recommendation of which developer should fix a particular bug can be considered a component recommendation. However, on open-source projects, developers often work on reports associated with multiple components, meaning a recommendation of who should fix the bug does not indicate to which component a bug belongs. Instead, a separate recommendation is needed to support the bug triage process.

As mentioned in the introduction, there are two previous approaches that have considered the recommendation of a component for a bug report. di Lucca et al. looked at five approaches: a probability space IR model, a vector space IR model, a SVM-approach, classification and regression trees, and a k-nearest neighbour classification approach. They found that the SVM-based classifier produced the best results, achieving an 84% accuracy for a single recommendation for a bug report. The limitation of this finding is that the only system to which it was applied was a proprietary system for which there were only eight distinct components. Anvik applied the basic approach for determining who to fix a bug to the problem of component recommendation [1]. His approach achieved a recall rate of 79% on Firefox bugs and 92% on Eclipse bugs for a recommendation list of size three. Similar to di Lucca et al. approach, Anvik's approach found the best results with an SVM-based classifier. One problem with SVM-based classifiers is that they may not produce consistent results for each component for which recommendations might be needed if some components do not have as many reports appearing as other components.

In this paper, we consider the use of an approach, Latent Dirchlet Allocation (LDA) [4] that could help overcome this limitation, producing recall levels consistently across multiple kinds of systems, regardless of the number of bugs associated with each system component.

3. BACKGROUND

In this paper, we use approaches based on LDA to automatically categorize bugs. We briefly describe features of bug reports and provide an overview of LDA to provide the necessary background to comprehend the approaches we investigate in this paper.

3.1 Bug Reports

Across the many bug repository systems in use, a number of features are common, such as each bug report having a short free-form title, a longer free-form textual description of the bug, a status describing the point in the resolution process the bug has reached (e.g., new, assigned, fixed, won't fix), a means for developers to record comments about investigations into the bug and a number of pre-defined fields to describe various characteristics of the bug (e.g., platform and release).

In this paper, we report on experiments with bug reports stored in Bugzilla repositories.[4] One characteristic of Bugzilla-stored bugs is that the longer free-form description field is unchangeable once a bug is reported. As the bug triage process starts immediately after a bug is reported, this description is critical to any approach that attempts to automate bug triage. The unchanging nature of the description in a Bugzilla bug lets us easily go back in time to what information was present when the bug was submitted.

3.2 LDA

LDA is an unsupervised generative model that categorizes the words that appear in a corpus of documents into clusters, typically referred to as topics. According to Blei et al. [4], given the parameters α and β, the joint distribution of a topic mixture θ, a set of z topics and a set of words w is given by,

$$P(\theta, z|w, \alpha, \beta) = P(\theta, z, w|\alpha, \beta)/P(w|\alpha, \beta) \quad (1)$$

The numerator and denominator in Equation (1) is intractable and cannot be computed directly. According to Griffiths [11],[5] a suitable approximation is given by:

$$P(z_i = j|z\_i, w)\alpha \frac{n^{w_i}_{-i,j} + \beta}{n^{(.)}_{-i,j} + W\beta} . \frac{n^{(d_i)}_{-i,j} + \alpha}{n^{(d_i)}_{-i,.} + T\alpha} \quad (2)$$

where $n^{(.)}_{-i}$ is a count that does not include the current assignment of the topic z_i,W is the number of distinct and valid words, T is the number of topics, $n^{w_i}_{-i}$ is the number of times the topic z is associated with the word w_i and $n^{d_i}_{-i}$ is the number of times the topic z is associated with the document d_i. Equation (2) is a straightforward Markov Chain.

[2] `eclipse.org/mylyn`, verified 2011/09/06

[3] `http://bugs.mozilla.org`, verified 2011/09/06

[4] `www.bugzilla.org`, verified 11/09/11

[5] `www.pnas.org/content/101/suppl.1/5228.full.pdf` verified 13/09/11

In this paper, we use the longer free-form textual description of the bug report as the document provided to the LDA computations. Before applying the LDA computation, we filter stop-words from the documents using an Indo-European tokenizer factory and a stoplisting filter provided by the Lingpipe API.[6]

To assign words to topics as part of the computation, we use the Gibbs Sampler approach [11] that is based on counts of words which are computed from the subset of data seen thus far instead of all of the words in the corpus. We chose this approach because the approximation algorithm in Equation (2) is a Markovian chain. In essence, when a bag of words from the descriptions of bug reports are given as input to LDA, a topic is first assigned randomly to each word. Then, in subsequent iterations, the computation samples a topic to each word based on the number of times that the word is related to the topic in documents and the number of times the document is related to the topic. A word can be assigned to more than one topic. At each iteration of the computation, the conditional probability P(w|z) and P(D|z) where D is the document(here bug report) and z is the sampled topic can be determined. To use this approach, we must set the hyper-parameters α and β and the number of topics T.

We computed an estimate of P(W|T) assuming T to be $\{20, 70, 120, 170, 220, 270\}$ keeping the hyper-parameters α to be $50/T$ and β to be 0.1. As described by Griffiths et al. [11], the computed value attained a peak for a particular value of T which is assumed to be the number of topics T. We compare this approach for determining the number of topics with one suggested by Arun et. al. [10] in Section 5.

As an example of applying LDA to a bug report, consider the following bug #200004 from Eclipse Platform.

> Build ID: M20060629-1905
> Steps To Reproduce:
>
> 1. start Eclipse with a document to edit (in Java or CDT Perspective)
> 2. type something (e.g. "System.")and trigger the code-completion (CTRL-Space)
> 3. wait a sec till the yellow javadoc pops up
> 4. use your desktop-switching-hotkey and watch eclipse crash
>
> More information:
> This is Eclipse 3.2 running on an RedHat Enterprise 3.0 Linux System. Happens both on Gnome Desktop as well as on an Sun Solaris CDE Desktop, which is connected via ssh -X.
> Repeateable on both Systems, even after clean restart, in both Java and C++ Projects (with CDT). This should either be a problem of this specific configuration, or has allready[sic] been reported. I haven't found an existing Bug report with this behavior, so I have reportet [sic] it now.
> The output to the terminal is: The program 'Eclipse' received an X Window System error. This probably reflects a bug in the program. The error was '(invalid parameter attributes)'. Email me for more details on configuration.

[6] `http://alias-i.com/lingpipe/`, verified 11/9/11

When considered within the corpus of bug reports, the words of this bug report will be assigned to different topics, each topic appearing with a different probability. For example, the word *Build* is associated with topic #2 and topic #9 by the LDA computation. The other words predominant in topic #2 are commandline, problem, api, jdt, internal, pde, loader, and runtime, which are not obviously related to the SWT graphics and windowing component. The words dominant in topic #9 are button, menus, layout, wizard, selected, and adapter factory, which are related to SWT concepts.

The output of applying the LDA computation to a set of bug report documents is two matrices: a document topic matrix where the $[ij]$th element provides the probability that the ith document is associated with the jth topic and a word topic matrix where the $[ij]$th element gives the probability with which the ith word is associated with the jth topic.

4. CATEGORIZATION APPROACHES

To provide a bug categorization approach based on LDA, we need to combine the LDA information with a supervised machine learning approach that can learn appropriate components for bugs in a training set and that can then predict the appropriate component for a new bug in a test set. We consider three different approaches.

4.1 LDA-KL

Kullback-Leibler (KL) divergence is an approach to classify the bug reports by measuring the divergence between each topic's centroid's P(D|z) obtained from LDA and a test bug's P(D|z), where P(D|z) is the probability that the document D is related to the topic z [6]. For the bug reports in a training set, since the component finally assigned to each bug is known, we can compute the average document topic probability of all reports in a component category. This average document topic probability acts like the centroid of topics for each component. To categorize a new bug report amongst the n components for a system, we can compute the divergence of the new bug report's document topic probability from the average topic probabilities of each component in the training set using Equation (3), where i represents z_i, Each topic cluster created by applying the LDA computation and the terms $P(i)$ and $Q(i)$ are the conditional probability $P(D|Z_i)$ of the test and training data set.

$$D_{KL}(P||Q) = \sum_i P(i) log(\frac{P(i)}{Q(i)}) \quad (3)$$

With this approach, for a bug report in the test set, we recommend three components whose average document probability diverges the least from the document topic probability of a test (or new) bug report.

4.2 SVM-TF-IDF

As described earlier, previous approaches have found SVM to be the most appropriate machine learning algorithm for component recommendation. In a SVM approach, the input is mapped into a higher dimensional space and maximum margin hyperplanes are constructed for the space. The standard SVM solves a binary classification problem: if the feature vector of a test input falls into the hyperplane, the test input is classified as positive, otherwise it is classified as negative. Multiclass SVM trasnforms the single multiclass problem into multiple binary classification problems.

Table 1: Characterization of Test Systems

	Eclipse	Mylyn	Mozilla
Time Period	2001-2011	2005-2011	1999-2011
# of Components	15	13	26
Training Reports	2396	1970	6832
Testing Reports	927	696	2795

Previous approaches that have used SVM for bug classification problems have used a feature vector comprised of the TF-IDF values for each word that appeared in a bug report in the test set. The term TF(t,d) is the number of times a term t is present in a document d. The term IDF(t) is obtained by dividing the total number of documents by the number of documents containing the term t. The term TF-IDF of a term t and a document d is the product of TF(t,d) and IDF(t). In our experiments, we use SVM multiclass[7] for classification.

4.3 SVM-LDA

The approach taken in SVM-TF-IDF of considering each word as a feature yields a rich, but very large feature set. In the SVM-LDA approach, we use the document probability matrix from the LDA computation as the feature vector provided to SVM. This approach provides a smaller feature vector than SVM TF-IDF and was shown to produce good results for the Reuters-21587 dataset [4]. Specifically, in this approach, the feature vector for each document is populated by the P(D|z), which is a measure of the probability of document D given a topic z.

5. EVALUATION

We applied each of the three methods, LDA-KL, SVM-TF-IDF and SVM-LDA to bug reports for three open source systems: Eclipse Platform, Eclipse Mylyn and Mozilla. These systems were used by Anvik [1], allowing comparison to earlier approaches.

Table 1 outlines the time periods of bug reports we considered for each of these systems, the number of reports we considered in the training and test sets from each system and the number of components defined for each system.

For each system, we considered all bug reports with status fixed and wontfix. For each report in the test set, we applied the three approaches to produce three component recommendations. We computed the recall of each approach across a system as:

$$Recall = \frac{(\#\ of\ reports\ categorized\ correctly)}{(\#\ of\ reports\ in\ test\ set)} \quad (4)$$

Table 2 compares the values obtained for each approach. This data shows that SVM-TF-IDF produces the highest recall, similar to the findings of Anvik [1].

As we mentioned earlier, one disadvantage of SVM-based approaches can be inconsistency in recall, particularly if there is not sufficient data for a particular component for which recommendations are sought. To investigate whether

[7] `http://svmlight.joachims.org/svm_multiclass.html`,verified 13/09/11

Table 2: Recall Results for Each Approach

Method	Eclipse	Mylyn	Mozilla
LDA-KL	85.54	77.44	82.3
SVM-LDA	83.81	76.48	77.23
SVM-TF-IDF	84.68	84.57	79.92

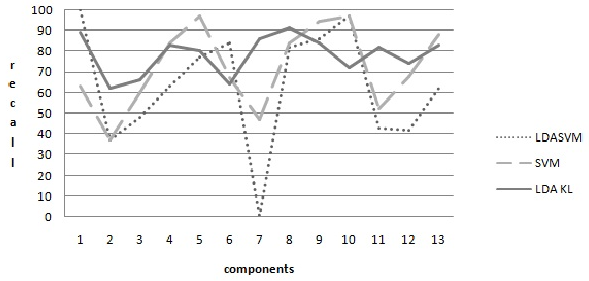

Figure 1: Recall for Mylyn Components.

LDA can add stability to the recommendations across components, we selected reports belonging to each component in each system and applied the three approaches. The graphs in Figures 1, 2 and 3 show the results for each component of a system. Though the SVM-TF-IDF and SVM-LDA have a recall as high as 98% for certain components, their range is very large (48% to 98%). We observe that the LDA-KL approach produces a consistent range of 70-95% and is the most consistent across all systems.

As we noted in Section 1, a limitation of SVM-based approaches it that they work less well for components that do not have a large number of reports in the repository on which the recommender can be trained. We argue in this paper that an LDA-based approach will be less sensitive in this regard. To provide evidence to support our claim, Figure 4 shows how the recall of the SVM-TFIDF and LDA-KL approaches vary with the number of positive examples used in training for the Mozilla system. This graph shows the higher degree of consistency provided overall by the LDA-KL approach.

Recall values for the two LDA-based approaches are affected by the initial assumption of the number of topics. For example, Figure 5 shows how the recall values achieved

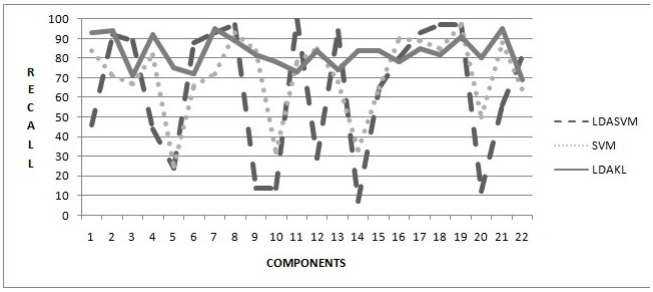

Figure 2: Recall for Mozilla Components.

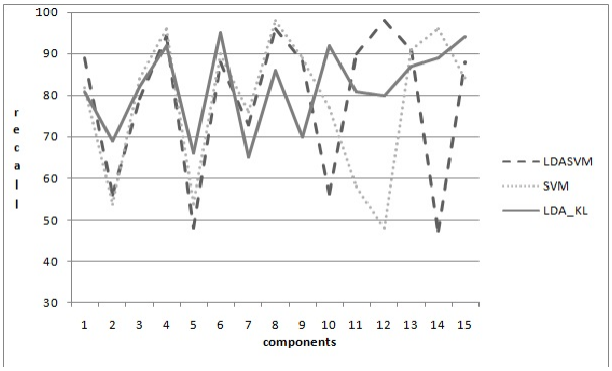

Figure 3: Recall for Eclipse Components.

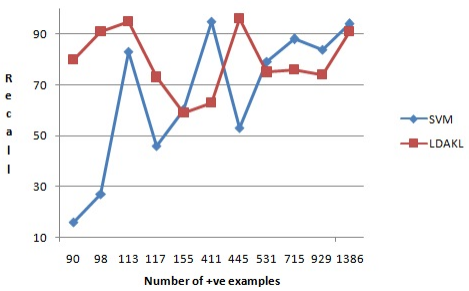

Figure 4: Recall Variation with Training Set Size.

for Eclipse Platform vary with the number of topics specified for the LDA-KL approach. In Section 3, we described how we tuned our choice of the number of topics for each system. We calculated the likelihood P(W|T) by taking the harmonic mean of P(w|z,T) where T is the number of topics and z is the sampled topic and looked for where the likelihood attains peaks. The likelihood attained maximum values when the T values were 50, 90 and 60 respectively for the Eclipse, Mozilla and Mylyn systems.

An alternate way of determining the number of topics has been described by Arun et. al. [10]. They describe that the number of topics T can be determined by running LDA on the corpus for different values of T with constant settings for the hyperparameters. For every value of T, their approach estimates the symmetric KL divergence between the singular values of the topic word matrix and the norm of the document topic matrix using Equation 5.

$$Measure = KL(CM1||CM2) + KL(CM2||CM1) \quad (5)$$

In Equation 5, $CM1$ is the distribution of singular values of the topic-word matrix, $CM2$ is the distribution obtained by normalizing the vector $L * M2$ where L is a one-dimensional vector of lengths of each document in the corpus and $M2$ is the document topic matrix. An appropriate value of T can then be determined by selecting the value for which the measure is at a minimum. If the value of T is chosen to be less than the minimum of the measure, than unrelated words cluster together reducing the accuracy of classification. T is chosen to be greater than the minimum of the measure, synonymous words will form different clusters, reducing performance overall.

Figures 6 and 7 plot the measure value from Equation 5 against a varying number of topics for the Mozilla and Eclipse systems respectively. The curve dips for T between 100 and 120 in Mozilla and for T between 40 and 60 in Eclipse. These values are consistent with those chosen using the other technique. The previous method assigned T to be 90 for Mozilla and 50 for Eclipse.

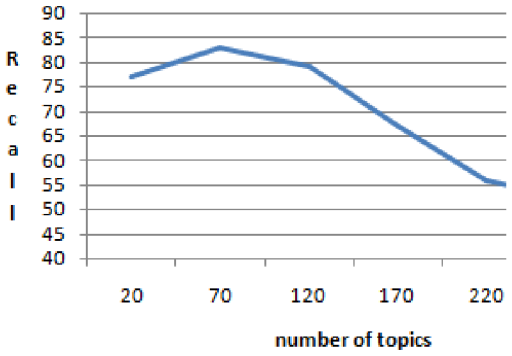

Figure 5: Recall Variation with Numbr of Topics for LDA-KL.

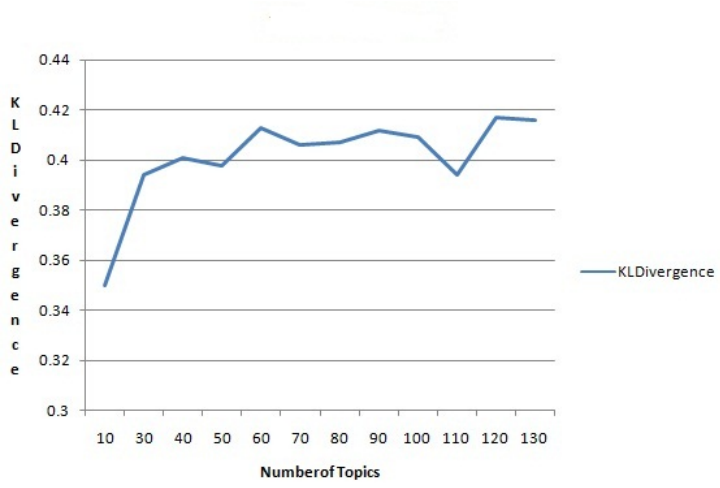

Figure 6: Variation of KL Divergence with Number of Topics in Mozilla.

6. DISCUSSION

We chose to focus on the use of the long free-form discussion portion of Bugzilla bugs for both the training of the classifers and when recommending components for a test bug report. As we mentioned earlier, this free-form discussion field in a Bugzilla bug report (the reports we used in

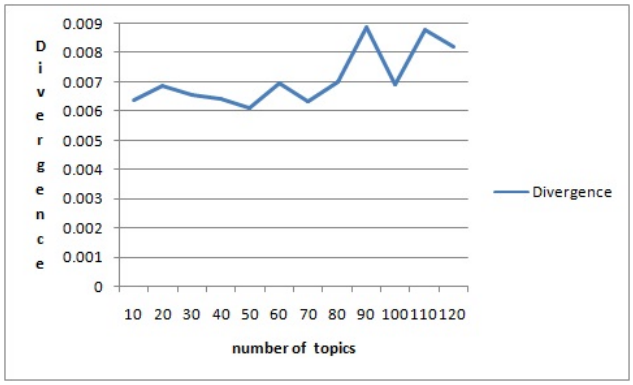

Figure 7: Variation of KL Divergence with Number of Topics in Eclipse.

our evaluation) cannot be changed after the information is first entered. For bug reports where the description if edited as more information is learned about a bug, it is possible that a classifier might achieve better recall given a better description on which to produce topics using an LDA-based approaches.

One might question our choice of not using the discussion software developers enter as comments in a bug report as input to LDA. In small experiments we performed that considered more information in a bug report, we found that sometimes the results can improve as the comments refine the naive description provided by a non-technical user. But in other reports, the comments distract from the description leading to wrong classification. Comments that have only stack traces, codes, patches and screen shots tend to reduce the accuracy of classification instead of improving it. Appropriate filtering to handle all of these cases would be needed to use comments effectively in the LDA classification approaches.

7. SUMMARY AND FUTURE WORK

We investigated whether combining LDA with a machine learning approach could improve the consistency with which component recommendation could be performed for bug reports. Through an experiment on three open source projects, we showed that an approach which combines LDA with Kullbach-Leibler Divergence (LDA-KL) can produce recommendations with more consistency in recall values across all components of a system than previous approaches that were based on SVM (SVM-TF-IDF) and approaches we considered that combine SVM and LDA (SVM-LDA).

In the future, we believe a similar approach could be used to determine the severity of the bug, the operating system of the bug or a code posted as a solution to a closely similar bug.

Acknowledgements

The authors acknowledge the support of MITACS-Globalink in making this project possible.

8. REFERENCES

[1] J. Anvik. *Assisting bug report triage through recommendation.* PhD thesis, University of British Columbia, 2007.

[2] J. Anvik, L. Hiew, and G.C. Murphy. Who should fix this bug? In *Proc. of the 28th Int'l Conf. on Soft. Eng.*, pages 361–370. ACM, 2006.

[3] J. Anvik and G.C. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. on Soft. Eng. and Methodology*, 20(3), 2011.

[4] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[5] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *Proc. of the 2006 ACM Symp. on Applied Computing (SAC)*, pages 1767–1772, 2006.

[6] B. Carpenter and B. Baldwin. *Text Analysis with LingPipe 4*. Lingpipe publishing, 2010.

[7] D. Cubranic and G.C. Murphy. Automatic bug triage using text categorization. In *Proc. of the 16th Int'l Conf. on Soft. Eng. & Knowledge Eng. (SEKE)*, pages 92–97, 2004.

[8] M. Di Penta G.A. Di Lucca and S. Gradara. An approach to classify software maintenance requests. In *Proc. of the Int'l Conf. on Soft. Maint. (ICSM'02)*, pages 93–102. IEEE Computer Society, 2002.

[9] P.J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. "Not my bug!" and other reasons for software bug report reassignments. In *Proc. of the ACM 2011 Conf. on Computer Supported Cooperative Work*, CSCW '11, pages 395–404. ACM, 2011.

[10] C.E. Veni Madhavan R. Arun, V. Suresh and M. Narasimha Murty. On finding the natural number of topics with latent dirichlet allocation: Some observations. *Advances in Knowledge Discovery and Data Mining*, pages 291–402, 2010.

[11] T.L.Griffiths and M.Steyvers. Finding scientific topics. In *Proc. of the National Academy of Sciences,101 (suppl. 1)*, pages 5228–5235, 2004.