Comparing Hierarchical Dirichlet Process with Latent Dirichlet Allocation in Bug Report Multiclass Classification

Nachai Limsettho
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
E-mail:nachai.limsettho.nz2@is.naist.jp

Hideaki Hata
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
E-mail: hata@is.naist.jp

Ken-ichi Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
E-mail: matsumoto@is.naist.jp

***Abstract*—Bug reports play essential roles in many software engineering tasks. Since validity and performance of these tasks definitely rely on the quality of bug reports, accurate information from bug reports is very important. However, as found in previous study, significant numbers of reports classified as bug are not really a bug. Recent studies proposed techniques to automatically classify bug reports into binary classes, yet there is still more to desire. These bug reports can be classified into multiple classes, which could help to identify what these reports are actually about. Moreover, previous study only looks into one possibility of topic modeling, that is, Latent Dirichlet Allocation (LDA). While LDA has its advantage, parameter tuning is required. In this paper, we propose a nonparametric approach to automatically classify bug reports with, another topic modeling method, Hierarchical Dirichlet Process (HDP). The result indicates that our nonparametric approach performance is comparable to the parametric one. We also examine various aspects of LDA to provide more thoroughly understanding of this process.**

Keywords—bug classification; topic modeling; bug reports; Hierarchical Dirichlet Process; Latent Drichlet Allocation; multiclass classification

I. INTRODUCTION

In data mining, good quality of data are a valuable asset. This also applies to empirical software engineering as well. Since nowadays, mining data from changes and bug databases had become common. As bug database is built from bug reports, quality of bug reports are crucial to data quality [1]. Correctly classified bug reports will greatly help in both research validity and modeling performance. More detail bug report will also contain more information which could help in understanding data. On the contrary, inadequate information and misclassified bug reports leads to misleading research and misrepresenting model. However, research by Antoniol et al found that significant number of bug reports are incorrectly classified [2]. Large number of reports classified as bug are not actually bug. They are, in fact, referring to other things such as a request for new feature, an improvement, or an update to documentation. These errors happen mostly due to reporter's misunderstanding and the complicating nature of Bug Tracking System (BTS) used for reporting number of other requests beside bug [2].

In order to correct these miss classification, large amount of effort is required, especially for manual inspection [2], [3], [4]. For example, Herzig et al. spent totaling 725 hours, 90 days, to classify over 7,000 bug reports. For this very reason, a technique to automatically classify bug reports is desired.

Several studies have been proposed to tackle this problem. A word-based automatic classification technique [2], by Antoniol et al, creates classification model base on word corpus and got a decent classification result. A recent study proposed binary classification based on topic modeling approach [5]. This method tries to improve bug report classification process by substitute word-level document corpus with Latent Drichlet Allocation (LDA) [6] topic membership vectors. The experiments shown that topic-based model outperforms word-based in almost of the evaluated cases.

Nevertheless, some problems still remain. First is that the LDA approach [5] requires a parameter tuning in order to work optimally. This means certain amount of effort is needed. Second, only one dimension of topic modeling, LDA, has been explored. Other dimensions of topic modeling are left uncharted. Third, it only works on binary classification this abandons some useful information, which otherwise would be obtainable with multiclass classification.

In this paper, we propose a nonparametric approach to automatically classify bug reports with Hierarchical Dirichlet Process (HDP) [7] as well as a method for handle multiple classes bug reports. In addition, various LDA parameters are experimented on in order to optimize this process. All experiments are evaluated in term of accuracy, F-measure and Roc.

The experiments in this paper are done on the combined dataset of three software projects based on the previous study published datasets [3]. These datasets are HTTPClient, Jackrabbit and Lucene. All of them are open-source. Each document in these data is categorized in to five classes which are bug, improvement, request for enchantment (RFE), task and test.

The contributions of this paper can be summarized as follows:

- We propose a nonparametric technique to automatically classify bug reports base on their textual information.
- Performance between different types of topic modeling is compared.
- We provide a guideline about how to optimize the previous LDA method.
- Multiclass classification method base on topic modeling is proposed.

This paper is organized as follows. In Section II, we discuss motivation behind our technique. Section III formalizes our method, we aim to solve. Section IV describes our methodology. Section V describes our experimental design. Experiment Results are reported in Section VI. Section VII discusses threats to validity in our research; followed by Section VIII that describes our related works. Finally, Section IX concludes our research and specifies future work.

II. Motivation

A. *Quality of Data*

For most of statistical and data mining tasks, good quality of data is essential. Mistake in data, aka noise, can leads to misleading and poor performance model. More specifically in bug classification task, decision boundary between each class can be significantly effect, thus make the classification model to be unsuitable for real world data [1].

Another aspect of data quality, is how detail it is. For bug report data, while binary classes data can contain a good amount of information and suitable for many tasks. The absence of some important information that otherwise obtainable with multiple classes data could be a problem. Using multiple classes data allows exact pinpoints of bug report purpose. It also allows research into many directions which would be impossible with binary one.

B. *Topic Modeling Approachs*

Using topic modeling to preprocess documents for bug report classification is advantageous in many ways. Compared with manual classification, it definitely saves a huge amount of effort and time. Its performance is also better than bag of words approach [5]. There are two reasons for this. First, since words that frequently occur together in document corpus are grouped into a topic, problems of synonymy and polysemy are diminished. Second, it projects a very sparse vector space model into a more compact and meaningful form. This generally helps classification in both computation time and classifying accuracy.

Topic modeling can be done in many different ways; in this paper two of popular methods are experimented. Latent Dirichlet Allocation (LDA) [6] is a Bayesian approach of topic modeling bug reports. This method views each document as mixture of various topics and assumes that topic distribution has a Dirichlet prior. Another way is Hierarchical Dirichlet Process (HDP) [7]. It is a nonparametric Bayesian model which assumes the number of topics from dirichlet process and allows mixture components to be shared between groups. Both approaches have its merit. While LDA is easier to apply since many tools and libraries implement it, the nonparametric nature of HDP is also very appealing. Experiments comparing these approaches are presented in Section VI.

III. Bug Report Multiclass Classification

We formalize our bug report multiclass classification in this section.

The input for training our method are bug reports and corrective dataset that indicates the actual bug report types [2]. After training is complete, the output will be classification model capable of automatically distinguish bug reports into many types.

Types of bug reports, aka classes, are descripted in Table I.

TABLE I. Bug Report Types (Classes)

Bug Report types (Classes)	**Description**
Bug	Report concerning corrective maintenance tasks to resolve incorrect or unexpected result of software system
Request For Enchantment (RFE)	Report concerning a call for a system adjustment which involve implement of new functionality (request for enhancement, feature request).
Improvement (IMPR)	Report concerning an improving task that upon resolution, would improve the overall performance of existing functionality
Task	Report concerning a general development tasks
Test	Report concerning test cases

IV. Methodology

Our bug report classification process is divided into two main phases. First is Topic Modeling phase, this phase converts bug reports into topics membership vectors. In second phase, Classification, data from previous phase are combined with its classes then pre-processed and used to build a classification models. More detail will be described in following subsections.

A. *Topic Modeling phase*

This phase is consist of five steps

1) Parsing: Our bug reports come in XML format. In order to get a more meaningful data from these bug reports, we extract three textual sections: title, description and comments. These sections are combined into a single text file per bug report.

2) Tokenization: After parsed, stream of text from bug reports are tokenized, broken into terms, and unnecessary punctuations are removed.

3) Stemming: In this step, the tokenized terms are mapped and converted to their root form. Porter Stemming algorithm [8] is used for experiments in this paper.

4) Removing stop words: Some words in English hold little to no meaning alone. As such, they are removed. We use stopwords from mallet 2.0.7 stoplist. The examples of these words are a, both, but, by, can, and the.

5) Topic modeling: Topic modeling is applied in this step in order to automatically extract topics from a text corpus. Two of well known topic modeling methods are applied in this research as we want to experiment on which approach is more suitable for topic modeling bug reports.

First is LDA, which are commonly used and implemented. This method is probabilistic generative model and it is required for user to specifiy number of topics (N). Since the best perform N is depend on the dataset, parameter tuning on topic's number is needed. Therefore, for any experiment that uses LDA topic, several numbers of topics are examined.

Second is HDP, which can assumed the number of topic by itself, hence reducing tuning effort. However, as second level of HDP drawn samples from already drawn subset of first level (assuming it is 2-levels HDP), topics drawn from HDP are overlap. This is differrent from LDA since LDA's topics are drawn separately from base distribution, thus making it less likely to overlap. Though the overlap nature of HDP can be advantageous in many situations, it can also make data to be harder to separate.

After topic modeling process, the output of both approaches is a set of topic membership vectors. Each vector represents a bug report and consists of a set of topics with its proportion. These topics comprise of co-occurring words throughout the bug report textual corpus and their proportion indicates what topics such bug report are related.

Figure 1 summarizes these steps in topic modeling phase.

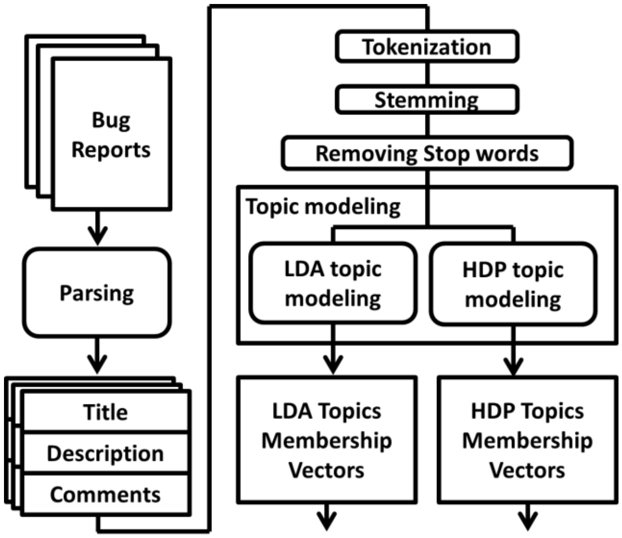

Figure 1 Diagram of our Topic modeling phase

B. *Classification Phase*

In this phase, HDP and LDA data are processed separately. Each topic membership vectors dataset from previous phase is combined with corrective dataset containing bug reports actual classes. This process is done to prepare these data for classification task. Data from LDA and HDP then proceed independently to the following steps.

Figure 2 summarizes these steps in Classification phase.

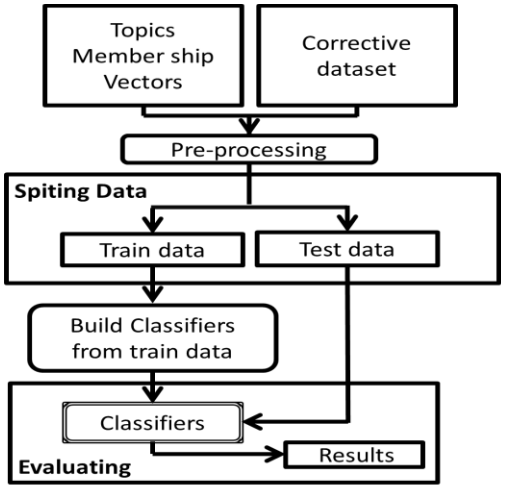

Figure 2 Diagram of our Classification phase

1) Pre-processing: This step will be described later in each experiment results subsection as the process is different for each experiment.

2) Spilting Data: Pre-processed data are then spilt randomly into two datasets: train and test dataset. The test data are reserved for evaulation, while train data are passes to the next step. To validate our result, we employ 10-fold cross-validation to all experiments in this paper and report average value of 10 runs as our result.

3) Build classifiers: Classification models, aka classifiers, are built from train data. All classifiers built in this research utilyze multiclass classifer all-against-all, a wrapper based type classifier. The wrapped classifier depends on the experiment. For Experiment 1, the base classifier is logistic regression. While in Experiment 2, three types of classification technique are wrapped: Alternating Decision Tree (ADTree), Naive Bayes and Logistic Regression. These classification techniques are chosen based on previous researches [5], [9]. Their details are described below.

a) All-against-All: This is a wrapper based type classification technique that enables binary classifiers to handling multiclass datasets. The wrapper based nature of this classifier means it cannot work by itself and needs a base classifier to wrap on. Specifically this classifier transforms a K classes classification problem into K(K-1)/2 binary classification problems of separating between each pair of classes, while ignoring the rest of them. Results from these classifiers then are combined via voting. We choose this method to handle multiclass problem since it is intuitive and has a good performance.

b) Alternating Decision Tree (ADTree): It is a set of generalized decision trees that employ boosting algorithm. Number of boosting iterations is user specific. For each iteration, the weight for each instance will be given differently according to the previous iteration results. The correctly classified instances are given reduced weight while the misclassified are given a larger weight.

c) Naive Bayes: This classifier applying Bayes' theorem with an assumption that each other features aside from class are independent.

d) Logistic Regression: A regression analysis that uses probability scores to measure relationship between class and features.

V. Experimental Design

Our study use combined data from three datasets in previous study [3]. The bug report from three open-source software projects from Apache: HTTPClient, Jackrabbit and Lucene, are combined. The reason we used combined dataset is because we want to evaluate each method in cross-project learning environment since it can greatly reduce amount of effort required in preparing training data in real world implementation. The Class distribution of combined dataset is shown in Table II.

TABLE II. Class distribution of combined dataset

Bug Report types (Classes)	**Number of Reports in each class**	**Percentage of Reports in each class**
Bug	2,718	49.96%
Improvement (IMPR)	2,092	38.46%
Request For Enchantment (RFE)	337	6.19%
Task	214	3.93%
Test	79	1.45%
Total	5,440	100.00%

Our experiments are divided into two parts: LDA-HDP comparison in Experiment 1, then optimization of LDA in Experiment 2. All experiments in this paper are evaluated by accuracy, F-measure and receiver operating characteristic (ROC).

VI. Experiment Results

A. *Experiment 1: Comparing LDA and HDP performance*

We want to compare LDA and HDP performance in this experiment, so aside from using LDA and HDP, all other parameters here are identical.

LDA and HDP topic membership vectors from Topic Modeling phase are independently processed. After combined with corrective dataset, both datasets are pre-processed. Here, the values in topic dimensions of each instance are occurrence of words from that topic divided by the sum of word occurrences from all topics of that instance. Note that this sum is not the same as a total word occurrence count for that instance as two different topics may have some identical words.

The pre-processed datasets are then spilt into train and test dataset. Then train datasets are used to build a multiclass classification model; in this experiment, All-against-all classifiers are built with Logistic Regression as base.

The result for Experiment 1 is shown in Table III where overall performance is presented in upper table. The first column of the upper table is evaluation measures. Other consecutive columns are grouped into results from LDA and HDP. For LDA, the three columns represent different numbers of topics which are 25, 50 and 100 respectively. As for HDP, since the number of topics for each run is not identical, we report the result from 3 runs of HDP in the following columns.

For the lower table, F-measure in each class is shown here. Bug report types or classes are in the first column while the other columns are identical to the one in the upper table.

TABLE III. Performance comparison between LDA and HDP

	Overall Performance					
	LDA			HDP		
	25 topics	50 topics	100 topics	1 run	2 run	3 run
Accuracy	62.94%	64.38%	67.04%	63.57%	63.99%	63.62%
F-measure	0.591	0.611	0.641	0.599	0.604	0.601
Roc	0.766	0.785	0.807	0.764	0.767	0.768
Number of Topics	25	50	100	42	46	47
	F-measure for each Type of Bug Report					
BUG	0.729	0.743	0.767	0.733	0.734	0.732
IMPR	0.584	0.605	0.639	0.598	0.607	0.603
RFE	0.006	0.034	0.09	0.006	0	0.006
TASK	0.026	0.1	0.091	0.052	0.077	0.066
TEST	0.024	0.1	0.229	0.065	0.068	0.022

As we see in Table III, LDA with 50 and 100 topics perform better than HDP and the trend seems to go up as the number of topics is increased. This can be interpreted in two ways. First, performance of classifier built from LDA topics will increase more and more as the number of topics increased; or second, the performance will increase until the number of topics reach a certain point then it will start to drop. This question is answered in Experiment 2, which indicates that second interpretation is right. Therefore we can summarize that with proper number of topics tuning, LDA performance is better than HDP. While this make HDP seems unsuitable for this task, it still has its use, as its performance is still comparable with classifier built from LDA and it requires no parameter.

The F-measure in each class from Table III demonstrates that for this dataset both LDA and HDP suffer from lack of data and imbalance dataset problems. The F-measure for the three minority classes are terrible. So measure for handling this problem is needed, the interesting approaches are sampling and cost-sensitive technique.

B. Experiment 2: Characteristics of LDA and its optimization

As for Experiment 2, we want to search for an answer for two questions. First is whether LDA performance can increased unlimitedly with the increase number of topics or it will start to drop at some point. Second question is how to optimize LDA for the optimum performance. This is achieved by varying numbers of topics (N), pre-processing methods and classifiers.

The numbers of topics in this experiment start from 50 topics and increase by 50 until it reaches 200. After that, we examine on every 100 topics until it reaches 600.

Three pre-processing methods are experimented on. First, simple count of words occurrences in each topic is used as value in topic dimensions. Second, we use the existence of words in the topic, the value will be 1 if for that bug report there is an occurrence of words in that topic and will be 0 if a word from that topic is not found. Third, the pre-processing method used in Experiment 1 is used.

Three type classifiers used as base in this experiment. They are describes below.

The results for Experiment 2 are shown in Table IV, V and VI. The Table IV shows accuracy, Table V shows weight F-measure and Table VI show Roc. Each table aligns in similar way. First column, N is the Number of LDA topics; the rest of columns are grouped base on their preprocessing. Count is simple count of words occurrence, Exist use the existence of word and Ratio is the pre-processing method used in Experiment 1. Each pre-processing column consists of three sub-columns that indicate used classification techniques, ADTree for Alternating Decision Tree, NB for Naive Bayes and LR for Logistic Regression. The black cell means that experiment on that cell position take too long to finish, thus left blank. The best result for each sub-column is marked with grey color.

TABLE IV. ACCURACY OF THE LDA TOPIC-BASED CLASSIFIER

	Count [integer,0-N]			Exist [boolean,0/1]			Ratio [double,0-1]		
N	AD Tree	NB	LR	AD Tree	NB	LR	AD Tree	NB	LR
50	0.62	0.24	0.61	0.60	0.53	0.61	0.62	0.34	0.64
100	0.62	0.32	0.63	0.61	0.52	0.63	0.64	0.35	0.67
150	0.62	0.50	0.63	0.59	0.51	0.62	0.63	0.39	0.67
200	0.61	0.50	0.63	0.60	0.50	0.63	0.62	0.40	0.67
300	0.62	0.51	0.62	0.60	0.50	0.61	0.61	0.42	0.63
400	0.61	0.52	0.62	0.61	0.51	■	0.62	0.46	0.63
500	0.61	0.51	■	0.60	0.51	■	0.60	0.49	0.60
600	0.60	0.52	■	0.60	0.50	■	0.61	0.49	■
AVG	0.61	0.45	0.62	0.60	0.51	0.62	0.62	0.43	0.65

TABLE V. F-MEASURE OF THE LDA TOPIC-BASED CLASSIFIER

	Count [integer,0-N]			Exist [boolean,0/1]			Ratio [double,0-1]		
N	AD Tree	NB	LR	AD Tree	NB	LR	AD Tree	NB	LR
50	0.59	0.29	0.56	0.56	0.53	0.58	0.58	0.40	0.61
100	0.58	0.36	0.60	0.57	0.52	0.60	0.60	0.41	0.64
150	0.59	0.43	0.61	0.56	0.52	0.60	0.60	0.44	0.65
200	0.57	0.44	0.61	0.57	0.51	0.62	0.59	0.45	0.65
300	0.59	0.45	0.59	0.56	0.51	0.60	0.58	0.47	0.62
400	0.58	0.46	0.62	0.57	0.51	■	0.59	0.49	0.63
500	0.58	0.45	■	0.56	0.53	■	0.57	0.52	0.61
600	0.56	0.46	■	0.56	0.51	■	0.58	0.52	■
AVG	0.58	0.42	0.60	0.56	0.52	0.60	0.59	0.47	0.63

TABLE VI. ROC OF THE LDA TOPIC-BASED CLASSIFIER

	Count [integer,0-N]			Exist [boolean,0/1]			Ratio [double,0-1]		
N	AD Tree	NB	LR	AD Tree	NB	LR	AD Tree	NB	LR
50	0.74	0.59	0.75	0.72	0.69	0.74	0.74	0.68	0.79
100	0.75	0.59	0.78	0.73	0.70	0.76	0.76	0.66	0.81
150	0.75	0.59	0.77	0.72	0.70	0.77	0.76	0.69	0.81
200	0.74	0.59	0.77	0.72	0.71	0.78	0.75	0.66	0.81
300	0.74	0.59	0.74	0.72	0.71	0.76	0.73	0.67	0.78
400	0.74	0.59	0.76	0.73	0.71	■	0.75	0.66	0.79
500	0.73	0.60	■	0.72	0.73	■	0.73	0.68	0.78
600	0.72	0.60	■	0.72	0.72	■	0.73	0.68	■
AVG	0.74	0.59	0.76	0.72	0.71	0.76	0.74	0.67	0.80

From ADTree and Logistic Regression columns of these tables, we can see that the performance of LDA topic-based classification model does not increase with the number of LDA topics. Instead, the performance will increase until reach the certain point depending on dataset, then it starts to drop. This is because too much increase in number of topics will lead to too sparse dataset and a lot of uninformative feature. As for some Naive Bayes columns that the best perform number of topics are 500 and 600, this is due to slower increase trend of Naive Bayes—which means that they have not yet reach the optimum performance.

When comparing performance by varying pre-processing methods, the Ratio method is the most promising one. Both ADTree and Logistic Regression perform best with this pre-processing while Naive Bayes on the other hand, prefers Exist method.

From these three classification techniques, Logistic Regression achieves the best evaluation scores in all three measurements: accuracy, F-measure and Roc. Though when the number of topics is exceed 300, its run time increases tremendously. In this regard, ADTree runtime is doing a lot better. Its runtime is more reasonable with large number of topics and its classifying performance is still comparable to the best performs logistic regression. As for Naive Bayes, although it got the fastest run time, its classifying performances is significantly worse than the other two techniques.

Therefore, for these mentioned reasons. We recommend using Ratio for pre-processing, using all-against-all with Logistic Regression as base for classification when the size of dataset is small while using ADTree with bigger dataset. As for the appropriate number of topics, it depends on the dataset but start from smaller N value is generally better for both Logistic Regression and ADTree.

VII. THREATS TO VALIDITY

This research experiments on published dataset from previous study. Although data we use are manual inspected with a fixed set of rules, some errors might still occur. The rules for manual inspection is also depend on individual perspective which could be different for each person. These might cause data to change thus cause our classifier to produce different results.

Some of processes in our research involve random value. For example, HDP and 10-fold cross-validation are both random process. Thus, although we try to repeat our experiment as much as possible to ensure the validity of our results, we cannot guarantee that our results are optimal.

Experiments are done on limited research subject. All bug reports in our combined dataset come from projects written in Java and using JIRA bug tracker which might not be representative for other programming language or bug tracker system.

VIII. RELATED WORKS

Many software engineering tasks use data mining to mine important knowledge from bug reports. The task, such as bug prediction [10], [11] uses bug reports, machine-learning and statistical analysis to identify pieces of code likely to contain bugs. Triaging reported bugs [12], [13] is another important task using bug reports. Since quality of data definitely affect the performance of data mining task, quality of bug report is crucial. However, significant number of bug reports is actually misclassified which will affect validity of research that works on these reports. Several researches [2], [3], [5] are try to address this misclassification problem. Some of them use manual inspection to identify misclassified reports [2], [3]. While the most recent one [5] employs topic modeling to extract textual information from bug reports, build classification model from correctly classified reports then uses this model to automatically classify uncorrected or unclassified reports. While this topic modeling method could work on binary classification, we notice that actual report from bug tracking system contain more than one types of report; thus, some information is lost when reports are transformed to binary class. Moreover, even though the process is automatic, some parameter tuning is required for number of topics. Our paper addresses these problems and tries to give a guideline for optimizing the process.

IX. CONCLUSION

In this paper, we proposes a method for automatically classify bug reports base on its textual information without the need to do a parameter tuning. This further reduces time and effort needs to process these bug report. The result from our experiment demonstrates that this nonparametric method performance is comparable, though lowers, to the parametric one. We also experiment on how to optimize the bug report classification process that use parametric method to topic modeling bug reports. The experiment are done on vary topic numbers, pre-processing methods and classification technique. The result could serve as a guideline to efficiently employ this bug report classification process. For future work, we plan to tackle lack of data and imbalanced dataset, the problems found in multiclass bug report corpus. We also want to improve the nonparametric method classification method performance. Last, we aim generalized our result by experiment on other project written in other programming language and different bug tracking systems.

REFERENCES

[1] N. Bettenburg, S. Just, A. Schr¨oter, C. Weiss, R. Premraj, and T. Zimmermann, “What makes a good bug report?” in Proc. SIGSOFT ’08/FSE-16, 2008, pp. 308–318.

[2] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Gu´eh´eneuc, “Is it a bug or an enhancement?: a text-based approach to classify change requests,” in Proc. CASCON ’08, 2008, pp. 23:304–23:318.

[3] K. Herzig, S. Just, and A. Zeller, “It’s not a bug, it’s a feature: how misclassification impacts bug prediction,” in Proc. ICSE ’13, 2013, pp. 392–401.

[4] F. Thung, S. Wang, D. Lo, and L. Jiang, “An empirical study of bugs in machine learning systems,” in Proc. ISSRE, 2012, pp. 271–280.

[5] N. Pingclasai, H. Hata, and K. Matsumoto, “Classifying Bug Reports to Bugs and Other Requests Using Topic Modeling,” in Proc. APSEC’ 20, 2013 pp. 13–18.

[6] D. M. Blei and J. D. Lafferty, ”In Text Mining: Classification, Clustering, and Applications”, Topic Models, 2009, pages 71-94.

[7] Y. W. Teh, M. I. Jordan, M. J. Beal,and D. M. Blei, “Hierarchical dirichlet processes,” Journal of the American Statistical Association, 2004.

[8] M. F. Porter, “Readings in information retrieval,”ch. An algorithm for suffix stripping, 1997, pp. 313–316.Report 653, Department Of Statistics, UC Berkeley, 2003.

[9] C. Hsu and C. Lin, “A comparison of methods for multiclass support vector machines,”in IEEE TRANSACTIONS ON NEURAL NETWORKS, volume 13, 2002. pp 415–425.

[10] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, “High-impact defects: a study of breakage and surprise defects,” in Proc. ESEC/FSE ’11, 2011, pp. 300–310.

[11] H. Hata, O. Mizuno, and T. Kikuno, “Bug prediction based on fine-grained module histories,” in Proc. ICSE ’12, 2012, pp. 200–210.

[12] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. Nguyen, “Fuzzy set-based automatic bug triaging: Nier track,” in Proc. ICSE’ 11, 2011, pp. 884–887.

[13] D. Cubranic and G. C. Murphy, “Automatic bug triage using text categorization,” in Proc. SEKE, 2004, pp. 92–97.