Automatic Unsupervised Bug Report Categorization

Nachai Limsettho, Hideaki Hata, Akito Monden and Kenichi Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology, Nara, Japan
Email: {nachai.limsettho.nz2, hata, akito-m, matumoto}@is.naist.jp

Abstract—Background: Information in bug reports is implicit and therefore difficult to comprehend. To extract its meaning, some processes are required. Categorizing bug reports is a technique that can help in this regard. It can be used to help in the bug reports management or to understand the underlying structure of the desired project. However, most researches in this area are focusing on a supervised learning approach that still requires a lot of human afford to prepare a training data. ***Aims***: Our aim is to develop an automated framework than can categorize bug reports, according to their hidden characteristics and structures, without the needed for training data. ***Method***: We solve this problem using clustering, unsupervised learning approach. It can automatically group bug reports together based on their textual similarity. We also propose a novel method to label each group with meaningful and representative names. ***Results***: Experiment results show that our framework can achieve performance comparable to the supervised learning approaches. We also show that our labeling process can label each cluster with representative names according to its characteristic. ***Conclusion***: Our framework could be used as an automated categorization system that can be applied without prior knowledge or as an automated labeling suggestion system.

Keywords— automated bug report categorization; topic modeling; clustering; cluster labeling

I. INTRODUCTION

Bug report often contains a great deal of information [1]. However, this information is usually implicit and some processes are required in order to understand its meaning. One way to do this is using manual inspection approach [2]. This approach has good accuracy but the process is very time consuming. In Herzig et al. [2], categorizing a bug report requires researcher to spend 6 minutes on average and since bug reports corpus is usually very large, this approach is not practical in many areas. It is also prone to human bias which human perspective could affect the process outcome.

The better solution would be using a supervised learning method to help in categorizing these numerous bug reports. Several approaches base on this method have been proposed [3, 4]. Antoniol et al. [3] proposed a classification technique that works over the word-level of the documents. However, recent approach base on topic modeling shows much better performance [4].

Despite its advantages, this supervised approach got some shortcomings; it still requires a great deal of human effort. Since preparing a training data needs human inspections and in order for supervised learning to work properly, a large amount of data is initially required. While the long existing project that already manually process its data, could bypass this problem easily; the same could not be said for many other projects.

Using unsupervised learning approach such as clustering can help reducing the amount of effort needed [5]. However, clustering also comes with its own problem; its result is hard to interpret. Thus, it usually requires experts to label these clusters with more understandable name.

The automatic cluster labeling algorithms [6, 7] thus far focus on a boarder range of documents. David Carmel et al. [7] proposes using Wikipedia to enchant labeling performance, such approach is advantageous when applied to labeling clusters of general documents. However, when applied to specific fields such as labeling clusters of bug reports, it is likely to fail since many words in bug report are usually too specific for Wikipedia to capture.

In this paper, we propose an almost fully automated approach to categorize bug reports with Hierarchical Dirichlet Process (HDP) [8] and clustering. We also provide a technique to automatically labeling cluster using NLP chunking and top words from relevant topics of that cluster. Our approach can infer many important parameters by itself, thus minimize the human effort needed in our process. The result indicates that our framework can distinguish different bug reports, then categorize them into different groups and label them with more representative names.

This paper is organized as follows. We discuss the Preliminaries in Section II. Section III describes our method. Section IV explains our experimental design. Experiment results are reported in Section V and threats to validity in Section VI; then related works in Section VII. Section VIII concludes our research and future work.

II. PRELIMINARIES

A. Topic Modeling

Projecting bug reports into topic vector space can be advantageous in many ways. When comparing to bag of words approach, its performance is definitely better [4]. This is mainly due to two reasons. First, by projecting documents into topic vector space, we can greatly reduce the effect of data sparseness which is one of the main problems of word-level approach. Second, by grouping words that frequently co-occur in document corpus into a single topic we can also reduce the problem of synonymy and polysemy. This generally makes documents easier to distinguish as well as reduce the computation time.

B. Supervised and Unsupervised Learning Approach

Supervised learning is widely used in many tasks of software engineering. However, while this approach has its own advantage, its major limitation is that it required training dataset in order to work. The luxury that not many projects can actually afford, since the human effort required in preparation of a training dataset is definitely not low.

Cross-projects supervised learning approach also tries to solve this problem. However, using bag of word or even topic modeling approach is likely to run into the same problem; the classification mode built from another bug report corpus will likely to not be able to cover many different words and different writing styles of a new corpus.

For unsupervised learning, the main advantage of this approach is that there is no need for a training dataset. Topic modeling is one of unsupervised learning methods. However, using topic modeling alone is often not enough to understand the dataset structure; even with topics proportion demonstrated, comprehending the similarity of each bug report in high dimensional data space is far from easy.

III. METHODOLOGY

We present our methodology in this section. Our framework can be divided into three main phases; Topic Modeling, Clustering and Cluster Labeling.

A. Topic Modeling Phase

This phase aims to project bug report into topic vector space. The input of this phase is bug reports and the outputs are topic membership vectors of entire bug report corpus and top word list of each topic. This list also contains the proportion of each top word in each topic. Our topic modeling phase consists of four steps.

1) Parsing: Incoming bug reports come in XML format; as such some unnecessary text such as tag, attribute and declaration are included. In order to transform these bug reports into more informative form, three sections: title, description and comments (if any) are extracted and combined into a single text file for each bug report.

2) Tokenization: In this step, we tokenize stream of text from previous step. The parsed stream of text is broken into terms and unnecessary punctuations are removed.

3) Removing stop words: Since many words in English hold little to no meaning when alone, they will not provide useful information when transformed into topic vector space that disregard their position. As such, these words are removed. Stop list from mallet 2.0.7 is used in this step.

4) Topic modeling: Topic modeling is applied in this step in order to project bug reports into topic vector space. Hierarchical Dirichlet Process (HDP) is chosen as our topic modeler. Mostly due to its ability to infer the number of topics automatically, hence reducing tuning effort and make our process much more automatic.

Note that even if we only use HDP in our experiment to improve the automation of our process, other topic modeler that needed parameter tuning, such as Latent Dirichlet Allocation (LDA), can also be applied easily. With proper parameters tuning, such topic modeler could achieve better categorization performance at the cost of increasing effort.

The outputs of this process are bug reports in topic vector space and top word list of each topic. Bug reports are represented by a set of topic membership vectors; each vector represents a bug report and consists of a set of topics with its proportion. Frequently co-occurring words are grouped into a topic and top words from each topic are output in the top word list.

B. Clustering Phase

In this phase, bug reports in topic vector space are clustered to group similar ones together. We use this to categorize bug reports without any prior knowledge of the data. The clustering methods we use in this step are Expectation Maximization (EM) and X-means algorithm [9]; both are a method that can estimate the number of clusters on its own. All experiments using X-means in this paper set minimum number of clusters to 2 and maximum to 10. Both of clustering algorithm employed using Weka 3.6. The output of this process is cluster assignment. Similar to our previous phase, this step could also employ prior knowledge such as number of cluster, given that they are known beforehand. This allows user to use boarder range of cluster algorithm as well as improve categorizing performance.

C. Cluster Labeling Phase

Clusters from Clustering Phase are labeled in this phase. Fig. 1. summarizes our Cluster Labeling phase.

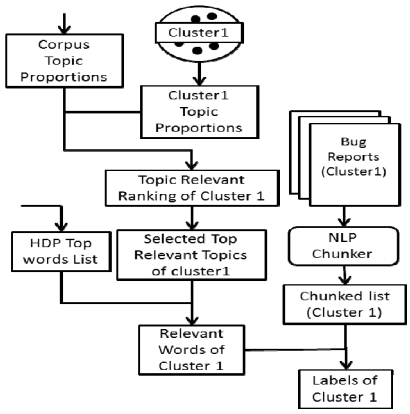

Fig. 1. Diagram of our Clustering Labeling phase

1) Calculating Average Topic Proportion: In this step, average topic proportions of each topic are calculated for each cluster and also for the entire corpus. The entire corpus proportion is calculated using all bug reports in topic vector space while each cluster proportion only uses a proportion of bug reports from that cluster.

For example, average topic proportion of topic *i* in cluster *j* will be the sum of topic *i* proportion from all documents in cluster *j* divide by number of document in cluster *j*. This equation is presented in Eq. (1).

$$AVG\ Topic_i Cluster_j = \frac{1}{n_j}\sum_{x=1}^{n_j} Topic_{i,x} \quad (1)$$

2) Ranking Topics of by their Relevant and Relevant Topics selection: This step aims to ranking and selecting relevant topics base on their ranking.

a) Finding Ratio: First is finding ratio of each topic in each cluster. This ratio is computed by dividing the average topic proportion of each cluster with average proportion of entire document corpus. Average proportion of topic *i* in cluster *j* will be divided by average proportion of topic *i* from entire corpus and so on. This is shown in Eq. (2).

$$RatioTopic_i Cluster_j = \frac{AVG\ Topic_i Cluster_j}{AVGTopic_i\ Corpus} \quad (2)$$

b) Ranking Topics by their Relevant: The goal is to sort topic based on their relevant. The relevant of each topic in the cluster is determined by the ratio, its occurrence in the cluster comparing to their occurrence in the corpus. This is from our assumption that topics that appear a lot more in specific cluster than in the corpus are likely to be best representative for that cluster.

c) Selected Top Relevant Topics: Sorted topics are then selected based on their ranking. We select top 20% of topics as relevant topic for each cluster. The output for each cluster is a set of topics that deemed as relevant for that cluster and their ratio.

3) Create List of Relevant Words: In this step we use the top word list, output of Topic Modeling phase, and selected top relevant topics from previous step as our input. Output of this step is a list of relevant words and their score for each cluster. This step consists of following sub-steps.

a) Prune Words: Top word list from Topic modeling phase is processed to remove words that are too short or too long from the word list. Since our topic modeling is done on bug report documents, some top words are hard to interpret or too specific terms: for example a word like "br" or "DistributedMultiQueryParser" is most likely unsuitable for using as cluster label.

b) Create Relevant Word list: Pruned top word list and selected top relevant topic are input of this sub-step. We compute the score of each word a from pruned top word list. The word *k* score in cluster *j* is equal to ratio of topic *i* in cluster *j* multiplied by word *k* frequency in topic *i* and divided by the total frequency of all top words in topic *i*. If same word is found in multiple relevant topics the score of that word is the sum of score from all relevant topics.

The idea is that if word *k* is occurring a lot more in the specific cluster j than in the entire corpus then such word k is likely to be a good candidate for a word in a phrase that representing cluster *j*. The equation is shown in Eq. (3).

$$Score_{k,j} = \sum_{i=1}^{Topics\ Number} \frac{Ratio_{i,j} \times \text{Frequency}_{k,i}}{\sum_{y=1}^{Words\ Number} \text{Frequency}_{y,i}} \quad (3)$$

4) Chunking: We use Natural Language Processing (NLP) chunker to extract meaningful phrases from bug reports of each cluster in this phase. The output of this phase is a list of noun phrases (NP) from the NLP chunked list and their frequency. We also limit length of phrase in chunked list; specifically, only phrase having length between two to four words is allowed in our chunked list. As the phrase with only one word is likely to be too board and phrase with five or more words is likely to be too specific. The chunking process in this paper is employed by using OpenNLP 1.5.3.

5) Create Label List: We use relevant word list and chunked list to generate cluster labels. Each cluster is labeled with five labels that have the most label score in that cluster. Labels are chunked list and score of each label is computed after selecting a combination of two words: word *k1* and word *k2*, from relevant word list. Each phrase in chunked list that contain these two words is moved to label list and have their score calculated. The phrase *m* score in cluster *j* is equal to frequency of phrase *m* in cluster *j* multiplied by average score of word *k1* and word *k2* and divided by phrase length squared. After check and compute a score for every item in chunked list, new words combination is selected; this process is repeated until every combination is used. In case that phrase contains more than one combination, its score is the sum of all combination existing in that phrase. This equation is presented in Eq. (4).

$$\forall k1, \forall k2 \in Candidate\ Word\ List: k1 \neq k2.$$

$$Score\ of Pharse_{m,j} = \sum \frac{Frequency\ of\ Phrase_{m,j}}{(Length\ of\ Phrase_m)^2} \times \frac{(Score_{k1,j} + Score_{k2,j})}{2} \quad (4)$$

The idea of this scoring method is that phrase that occurs regularly and consist of high score relevant words should be a good cluster label, while the phrase with longer length should have some penalty.

The output of this step is top five label list; these labels are used as cluster label. We use five labels per one cluster instead of one since it can improve coverage as well as giving more insight about the bug report in the cluster.

IV. Experimental Design

We describe our experimental design in this section. There are two primary goals for our experiments. First is to evaluate our framework's ability to understand a meaningful structure of the bug report corpus. Second is to demonstrate that our labeling algorithm can provide good labels that are both human interpretable and representative for their cluster.

A. Measurements

Measurements used in our experiments are cluster purity classification accuracy (for comparison) and F-measure.

1) Cluster Purity: Cluster purity indicates overall purity of all clusters. If cluster result is capable of distinguish different classes, its purity will be high. In order to calculate purity, cluster will be assign classes based on the most frequent class in that cluster. However, each class will have at least one cluster representing it; in case that there is a class that does not get labeled to any cluster, cluster with

minimal difference in number of instance between the two classes will be chosen. After labeling, we sum number of instances in each cluster that match its cluster class and divided it by its total number of instances.

$$Purity(\alpha, C) = \frac{1}{n}\sum_{j} max_i|Cluster_j \cap Class_i| \quad (5)$$

Where n is the total number of instance, α is the set of clusters and C is the set of classes.

2) Accuracy: This measure is similar to cluster purity but it used for evaluating classification instead of clustering.

$$Accuracy = \frac{Number\ of\ correctly\ classified\ instance}{Total\ number\ of\ instance} \quad (6)$$

3) F-measure: We use F-measure to evaluate each class and overall performance. In order to compute this score, we need to calculate precision and recall first.

Precision is calculated by equation 7. True Positive (TP) is the number of instances categorized as interested class and actually that class, while False Positive (FP) is the number of instance that falsely categorized as interested class.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \quad (7)$$

Equation 8 is Recall calculation. False Negative (FN) refers to the number of instance in the interested class that incorrectly assign to other class.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \quad (8)$$

The F-measure itself is the harmonic mean of precision and recall; it giving equal weight to both measures and can be calculated by equation 9.

$$F\text{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (9)$$

This measure is used to evaluate the performance of each class on both classification and clustering. In case of clustering, each cluster is labeled with its most prevalent class in the same manner used in cluster purity. Multiple clusters can also be assigned to the same class. Weight Average summarizes F-measure by taking each class number of instances into account.

B. Experimented Data

We use three bug report datasets from [2] in our experiments. In each dataset, there is only two classes BUG and Other Requests. Other Requests class is actually a combination of four classes Request for Enchantment (RFE), Improvement, Task and Test classes. Each project class distribution of is shown in Table I.

TABLE I: Class distribution of experimented datasets

Project	Total Number of Instances	Num of Bugs Class	Num of Other Requests Class
Lucene	2382	1037	1345
Jackrabbit (JCR)	2328	1213	1115
HTTPClient	731	469	262

C. Design for each Experiment

We explain our experiments in this section. There are a total of two experiments in this research. First is to measure the capability of distinguishing between two groups of bug reports that are not closely related. In order to do this, we combine two datasets, Lucene and JCR, and try separating them using our framework. In this experiment, we compare our clustering result with two well-known classification methods; J48, an implementation of C4.5 decision tree, and Logistic Regression. For both methods, training and testing data are created from bug reports in HDP topic vector space by using 10-fold cross-validation.

Second experiment tries to evaluate distinguishability between more closely related groups. Specifically, this experiment tries to separating two different types of bug report in the same project. In this experiment we use bug report data from three projects: Lucene, JCR and HTTPClient. Other settings are similar to experiment 1.

Cluster labeling results of second experiment are shown in section 5.3.

V. EXPERIMENT RESULTS

A. Experiment1: Categorizing bug reports from different projects

The goal here is to evaluate each method performance in capturing the structural difference of bug report from two software engineering projects.

Experiment results are presented in Table II. The result in this table is the average values from 3 Runs. These Runs represent different HDP results; since HDP process involves random value, the result of each run will be slightly different. As such, results from three HDP runs are averaged in order to reduce the results variance. Number of topics and number of clusters which could be different in each run are also averaged. The lower part of the table represents F-measure: per class and Weight Average which summarizes the overall performance from both classes.

TABLE II: Categorizing bug reports from different projects

	Lucene+JCR			
	Average result from 3 Runs			
	X-means	EM	J48	Logistic
Accuracy/purity	0.867	0.793	0.916	0.965
Number of Topic	57.333	57.333	57.333	57.333
Number of Cluster	4	12.667	2	2
F-measure				
JCR	0.872	0.789	0.915	0.964
Lucene	0.866	0.791	0.916	0.965
Weight AVG	0.869	0.787	0.916	0.965

From Table II, we can see that both approaches can separate bug reports from different projects quite well, their accuracy/purity and F-measure are high. This means the textual structure of bug reports from two different projects are indeed quite different and can easily be separated.

When comparing performance of X-means approach and supervised learning, Table II shows that both J48 and Logistic Regression performance is slightly better. For J48 the different in weight average F-measure is around -0.026 to -0.064 which accounts for 2.9 - 6.9 percent divergent. While for Logistic Regression, the gap is larger; their different is around -0.088 to -0.105 accounting for 9.1 - 10.9 percent. This demonstrates that, while our performance is lower, it still comparable to the supervised approaches which having the advantage of the prior knowledge.

For X-means and EM, our result shown that, in this task, X-means is overall better; it overcomes EM in both purity and F-measure.

B. *Experiment2: Categorizing bug reports in each project into Bug and Other Requests*

We evaluate performance in separating bug reports of the same project in this experiment. Each approach tries to categorizing bug reports into two classes: BUG and Other Requests. Experiment2 results are shown in Table III. The other setups for this table are the same as Table II.

In this experiment, the F-measure differences between our X-means approach and J48 is in the range of +0.010 and -0.068. This shows that even in environment that instances are closely related our framework performance is comparable to J48, the well-known supervised learner.

For Logistic Regression, difference between our and Logistic Regression is around -0.066 to -0.171; the difference here is indeed significant. This mean using Logistic Regression is recommended when training data is available. However, in case that prior knowledge is unobtainable or too costly, our framework is much more appealing. It can also be used as a suggestion system in the training data manual labeling process, which supervised learner such as Logistic Regression required.

When comparing results between our two clustering approaches, we can see that the EM approach does a lot better here than in the previous experiment; its results become more akin to X-means approach and even better in some runs. This is likely due to three reasons: smaller dataset, more implicit data structure and X-means tendency to produce equal size clusters. First is that EM has a tendency to create a lot more cluster than X-means. When dataset get smaller the advantage of more clusters number became more apparent especially in the purity measure. Second, as data structure aimed to capture in this experiment is much more complex, it becomes more representable through higher clusters number than fewer one. Last is that cluster produced by X-means approach is likely to be in the same size. However, with all that said, X-means and EM performances are still close to each other. All in all we recommend the EM approach when it is known beforehand that dataset structure is likely to be an imbalance and/or very complex; otherwise the X-means approach is better due to much faster runtime and comparable result.

C. *Experiment 1 and 2 Clusters Naming Results*

We provide top five labels from our X-means approach in Table IV. Samples from X-means clusters are presented for visualize purpose, since presenting labels from EM will take too much space due to a large number of clusters. This table showed our labeling result from Lucene dataset. The columns in this table are the original cluster name while each row is our cluster labeling result: Label 1 is label that has the highest label score followed by Label 2, Label 3 and so on. These scores come from Equation 4, the higher the score the more likely for that label to be a good label for this cluster. Below these label rows, are rows that represent the number of instances in each class: BUG and Other Requests. The class of each cluster, which decides by a majority of each class instance, is marked by grey color.

TABLE IV: Top 5 Labels of Lucene project, X-means 1 Run

	Lucene X-means 1 RUN			
	cluster 0	cluster 1	cluster 2	cluster 3
Label 1	term query	non lucene files	merge segments	junit test
Label 2	relevant queries filters	ant file	merge deletes	junit bad file descriptor
Label 3	query scorer	build xml file	flush to merge	fields junit test
Label 4	query collector having	jar file	merge thread	junit note
Label 5	searcher query internals	file src java	merge policy	error junit note
BUG	297	266	372	102
Other Request	486	471	367	21

TABLE III: Categorizing bug reports in each project into Bug and Other Requests

	Average result from 3 Runs											
	Lucene				JCR				HTTPClient			
	Xmeans	EM	J48	Logistic	Xmeans	EM	J48	Logistic	Xmeans	EM	J48	Logistic
Accuracy/purity	0.602	0.608	0.633	0.710	0.616	0.609	0.636	0.745	0.600	0.667	0.626	0.710
Number of Topic	47.67	47.67	47.67	47.67	55.67	55.67	55.67	55.67	52.67	52.67	52.67	52.67
Number of Cluster	4	11.33	2	2	4	14.67	2	2	2	10	2	2
F-measure												
BUG	0.445	0.428	0.565	0.634	0.656	0.687	0.632	0.732	0.673	0.678	0.717	0.787
Other Requests	0.688	0.694	0.682	0.760	0.549	0.525	0.639	0.756	0.470	0.435	0.448	0.550
Weight AVG	0.582	0.556	0.631	0.705	0.605	0.610	0.636	0.745	0.600	0.591	0.621	0.702

From clusters of Lucene bug reports; Label 1, 3, 4 and 5 are very good candidates for cluster 3 name. Since this cluster contains a lot reports about BUG, labels containing words such as “junit”, “test”, and “error” are certainly a good representative for the structure of this cluster. On the other hand in cluster 2 that number of each class are more similar, we got labels that mostly contain a word “merge” which seem much less bias to either class. As for cluster 0 and 1, our labels suggest that words such as “query” is likely to be more associate with Other Requests more than BUG; same goes for the word “file” in bug reports.

VI. Threats to Validity

A. Experiments are done on published dataset

The categories of bug reports are from other published study. Even though these datasets are manually inspected and cross validated, it is still possible that some error might still occur. This change will likely cause our experiment to produce different results.

B. Research subjects of our experiment are limited.

Experiments in this research are done on bug reports of projects written in Java and using JIRA bug tracker. This data might not be representative for other programming language or bug tracker system.

VII. Related Works

Using data mining to mine important knowledge out of bug reports is common practice in many software engineering tasks. Most of them [3, 4, 10, 11] used supervised learning approach to extract hidden information within bug reports. In the task of bug prediction [10, 11], classification and statistical analysis are employed to identify faulty code. Another task using bug report is triaging reported bugs [12, 13] which try to determine urgency of each bug.

There are many researches that try to categorize bug reports into predetermine categories [2, 3, 4]. Some of them use manual inspection [2]. While this method have the advantage of flexibility and accuracy; it is also very time consuming and prone to human bias. Using supervised learning to automate this process [3, 4] can reduce amount of time and human effort needed. However, the training dataset is still required. In most cases, obtain this dataset itself is not an easy task, usually required a lot of human effort to gather and correct the data. Using predetermined categories also could limit the knowledge obtain from the data. For example, one single predetermine category might be better represented by two for a certain dataset.

In categorizing bug reports study, not many researches try using unsupervised learning approach. Clustering, one of unsupervised learning is mostly only used for bug triaging. Furthermore the current methods of labeling cluster [6, 7] are more focus on labeling general documents that are from different fields; this make them unsuitable for labeling documents clusters that are closely related and embedded in a specific environment like bug report. Our paper addresses this problem by using an unsupervised learning approach and automated cluster labeling method.

VIII. Conclusion

In this paper, we propose a framework to automatically cluster bug reports and label these clusters based on their textual information. Our framework is also capable of infer many important parameters on its own, minimizing amount of human effort required in parameters tuning, as well as finding hidden knowledge in bug reports structure as our categories are not predetermined.

The results from our experiments demonstrate that our almost fully automatic approach could achieve comparable performance, even if lower, to supervised learning approach that has the advantage of prior knowledge. We also show that our labeling method can provide labels that are representative for each cluster. Our work could be used as automated bug report categorization system or as a label suggestion system helping in the manual data inspection.

As for future work, we aim to improve our categorization framework increasing its performance while still retain its nonparametric and none requirement for prior knowledge property. We also aim to improve our labeling method making our label easier to understand.

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] K. Herzig, S. Just, and A. Zeller, “It’s not a bug, it’s a feature: how misclassfication impacts bug prediction,” in Proc. ICSE ’13, 2013.

[3] 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.

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

[5] S. Mani, R. Catherine, V. Singhal, and A. Dubey, “AUSUM: approach for unsupervised bug report summarization,” in Proc. ACM SIGSOFT ’20, 2012.

[6] F. Geraci, M. Pellegrini, M. Maggini, and F. Sebastiani, “Cluster generation and labeling for web snippets: A fast, accurate hierarchical solution,” Internet Mathematics, 2007, pp 413-443.

[7] D. Carmel, H. Roitman, and N. Zwerdling, “Enhancing cluster labeling using Wikipedia,” in SIGIR, 2009, pp 139-146.

[8] Y. W. Teh, M. I. Jordan, M. J. Beal,and D. M. Blei, “Hierarchical dirichlet processes,” J Am Statist Assoc, 2004.

[9] D. Pelleg, and Moore, A. 2000, “X-means: Extending K-means with Efficient Estimation of the Number of Clusters,” in Proc Machine Learning ’17, 2000, pp.727-734.

[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, “Fault-prone module detection using large-sclae text features based on spam filtering,” in Proc. ESE, 2010.

[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.

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