International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

Unsupervised Bug Report Categorization using Clustering and Labeling Algorithm

Nachai Limsettho, Hideaki Hata, Akito Monden* and Kenichi Matsumoto

Graduate School of Information Science,
Nara Institute of Science and Technology
nachai.limsettho.nz2, hata,matumoto@is.naist.jp
**Graduate School of Natural Science and Technology*
Okayama University
monden@okayama-u.ac.jp

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Background: Bug reports are one of the most crucial information sources for software engineering offering answers to many questions. Yet, getting these answers is not always easy; the information in bug reports is often implicit and some processes are required to extract the meaning of these reports. Most research in this area employ a supervised learning approach to classify bug reports so that required types of reports could be identified. However, this approach often requires an immense amount of time and effort, the resources that already too scarce in many projects.

Aims: We aim to develop an automated framework that can categorize bug reports, according to their grammatical structure without the need for labeled data.

Method: Our framework categorizes bug reports according to their text similarity using topic modeling and a clustering algorithm. Each group of bug reports are labeled with our new clustering labeling algorithm specifically made for clusters in the topic space. Our framework is highly customizable with a modular approach and options to incorporate available background knowledge to improve its performance, while our cluster labeling approach make use of natural language process (NLP) chunking to create the representative labels.

Results: Our experiment results demonstrate that the performance of our unsupervised framework is comparable to a supervised learning one. We also show that our labeling process is capable of labeling each cluster with phrases that are representative for that cluster's characteristics.

Conclusion: Our framework can be used to automatically categorize the incoming bug reports without any prior knowledge, as an automated labeling suggestion system or as a tool for obtaining knowledge about the structure of the bug report repository.

Keywords: Automated bug report categorization; topic modeling; clustering; cluster labeling.

1. Introduction

Bug reports offer important insight into the status of the software project; they can be used to estimating the bug-fixing time [1, 2], deciding which bug should be fixed [3], or analyzing the bug type distribution [4, 5]. Categorization is one way to extract meaningful information from bug reports. Traditionally, these reports are inspected and categorized by humans; this approach has good accuracy and flexibility. However, the time it takes to understand each individual report combined with the numerous numbers of reports make manually reading through them impractical or even impossible in many situations [5]. It is clear that in order to use categorization in a practical environment, an automated system is needed.

Many approaches using supervised learning [6–8] have been proposed to automate the process of obtaining information from bug reports. These approaches construct a classification model from the training data of the labeled bug reports that can later be used to automatically categorize new incoming data into predetermined labels. The advantage of this approach is that it can greatly reduce the amount of human effort required after a classification model is built. However, building that said model is, sometimes, quite difficult. The supervised learning approach requires a lot of labeled bug reports to construct its model [7], but these reports are often unavailable in many software engineering projects.

There are mainly two options that can be used to obtain labeled data for the project without historical data. The first approach is manual inspection [5], which as mentioned previously requires a great amount of human effort. The second is a cross-project classification [9–11], which builds a classification model from the labeled dataset of another project. While this approach has its own advantages, there are also some limitations. Depending on the characteristics between the target and training projects, the same class might not represent the same concept and would be better to differently represent. To illustrate, while both Debian and HttpClient projects have bug reports in a network category, their impacts are starkly different. In an operating system project such as Debian, network bug, while still important, is not as critical as in HttpClient that mainly focus on communicating. Instead of grouping every network bugs in just one class, it would make much more sense for HttpClient to divide this class into several categories.

Another way to obtain knowledge from bug reports is unsupervised learning which extracts information from the underlying structure of unlabeled bug reports. The major advantage of this method is its ability to categorize the bug reports without the need for pre-labeled data. Also, since knowledge obtained by this method is not limited by the pre-determined categories, using this method may allow information to be discovered that might otherwise be easily overlooked by supervised learning.

While using unsupervised learning offers many advantages, this approach has not been applied to classify bug reports. Moreover, this approach still requires some human effort to understand the obtained categories. In other application domains, there have been many attempts to automate a cluster labeling process [7, 12–14] in order to make them easier to comprehend by labeling these clusters with more representative names. Most approaches do this by using either the most prevalent words in that cluster or bigrams of words in the cluster. However, while using these methods makes the cluster result easier to understand to some degree, using the n-gram does not take into account the grammatical meaning of sentences; its prerequisite is just adjacent terms in a string. Therefore, in many cases, the cluster that is labeled by the verb phrase will be quite counter-intuitive for a human inspector since we normally label a group of things using a noun phrase. For example, between wont connect and connection timeout, the second one is more likely to be a better label candidate.

This paper extends our workshop paper and gives improvements and further investigation on the performance of our previous categorization framework [15]. In that paper, an almost fully automatic framework was proposed to categorize bug reports, according to their textual contents . That paper also provided a new technique for automatically labeling a cluster using NLP chunking [16] and top words from relevant topics of that cluster.

While the previous paper shows a potential of the proposed framework, there are still many questions left unanswered: Is using topic modeling actually help improve the categorization performance? Is it stable? How well does it perform compared to cross-project classification? These questions are answered in this paper.

Another problem is that the previous labeling algorithm lacks the variety in its suggesting labels; a few high ranking words dominates almost its entire suggesting labels. For example a word JUnit, although it is representative for its BUG related cluster, appears in 9 out of 10 of the suggesting phrases. This lowers the coverage of the suggesting labels, which is not a good thing. To counter the lack of term variation, this paper presents a new weighted reduction algorithm to increase the variety of terms in the suggesting labels.

Our result shows that our method can distinguish between different types of bug reports and can categorize them into different groups with a performance result comparable to the supervised learning approach. It also shows that our cluster labeling method can generate representative labels for clusters built in topic vector space.

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, threats to validity in Section VI, and related works in Section VII. Section VIII concludes our research and future work.

2. Preliminaries

2.1. *Supervised Learning Approach*

Supervised learning is widely used in the area of software engineering; the most prevalent method is a classification that trains a classification model with training dataset to later be used to classified new incoming data [7]. Each instance in the train dataset is labeled with its actual class; these classes are pre-determined and act as prior knowledge which the model will try to learn.

While this approach is widely used and clearly has its own advantages, its major problem lies in its absolute requirement for the prior knowledge; without this information the classification model simply can-not be built and obtaining this knowledge is far from easy. Since to obtain a good classification model, a large amount of training data is needed, human inspection is required in order to prepare this dataset. In Herzig et.al [5], a large amount of time and effort are spent to reclassify bug report categories.

One way for supervised learning to mitigate this problem is cross-project classification [9–11]. This method builds a classification model from a dataset from another project instead of using its own. This generally makes obtaining training data become easier, and it also makes the concept that the model learns become much more general. However this generality, sometimes, becomes its own downfall. When the classification model is intended to be used only in one project, the project manager will definitely want a model that best performs on that said project, not one that works best in general. Technically, the decision boundary that the model used to categorize each class will be distorted causing parts of the data to be misclassified.

2.2. *Unsupervised Learning Approach*

Some research in this area uses unsupervised learning to find hidden structures within their data. It is commonly used in bug triaging [17], duplicate bug report detection [18] and in topic modeling [6, 19]. The main advantage of this approach is the non-requirement of a training dataset. This greatly helps reduce the amount of effort required to obtain and process prior knowledge which would otherwise be needed for the supervised learning.

Aside from this, the amount of knowledge obtainable from supervised learning is also limited by its prior knowledge; supervised learning cannot comprehend anything beyond which it is specifically taught. This means that supervised learning will always categorize bug report to the predetermined class which, sometimes, is not the best approach since a certain class might be better represented as two or more in certain situations.

2.3. *Topic Modeling*

Topic modeling is an unsupervised learning technique that captures the underlying structure of the document repository by grouping co-occurrence words into the same topic [20, 21]. The result is a set of topics, a cluster of words that likely to share the same meaning. A document can be associated with topics using a topic proportion vector that indicates what topics that document is associated with. The more the document relates to the topic, the more proportion is assigned to that topic. Compared to the bag-of-words [8, 22], topic modeling can greatly reduce the effect of data sparseness, which is one of the main problems of the word-level approach. In addition, this approach also help reduces synonymy and polysemy problems by grouping the co-occurrence words together. This generally makes documents much easier to distinguish and it reduces the computation time for both supervised and unsupervised learning.

In the software engineering field, there are several research areas that use topic modeling. However most of them are used in bug triaging [17] and duplicate detection [18], while only a few of them are applied to the categorization and knowledge discovery of bug reports. Plingclasai et.al [8] has shown that this area could benefit from topic modeling and could significantly improve classification performance of bug reports by just adopting it instead of using a word-level model [23]. The problem is that using this approach alone is often not enough to understand the underlying structure of bug reports; even with topics proportion demonstrated, comprehending the similarity of each bug report in high dimensional data space is far from easy.

3. Methodology

We present our methodology in this section. Our framework can be divided into three main phases: Topic Modeling, Clustering and Cluster Labeling. The first phase, Topic Modeling, is to preprocessing raw bug reports textual content. These bug reports are then projected into topic document vectors, a format which can easily be utilized by machine learning algorithms. In the second phase, projected bug reports are categorized according to its textual similarity using clustering algorithm. Finally, in the Cluster Labeling phase, each group of bug reports is labeled by phrases that portray its characteristics.

3.1. *Topic Modeling Phase*

Bug reports are projected onto topic vector space in this phase. This projection processes and transforms bug reports into a more manageable form, and topic modeling is employed instead of the bag-of-words to mitigate data sparseness as well as the effect of word ambiguity. The input of this phase is bug reports in the XML format, and the outputs are topic membership vectors of all the bug reports in the corpus and the 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, presented in Figure 1.

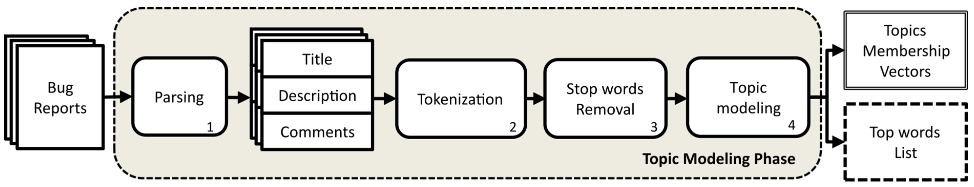

Fig. 1. Diagram of our Topic Modeling phase

1. **Parsing** Raw incoming bug reports are in XML format; therefore, some non-textual information such as tags, attributes and declarations are also included. For each bug report, three sections: title, description and comments are extracted and combined into a single text file.
2. **Tokenization** In this step, in order to transform these bug reports into a more processive form, we a tokenize stream of text from the previous step. The parsed stream of text is broken into words and unnecessary punctuation is removed.
3. **Stop Words Removal** Since many words in English hold little to no meaning when alone, they will not provide useful information when transformed into a topic vector space that disregards their position. Therefore, these words are removed. The 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 onto the topic vector space. The Hierarchical Dirichlet Process (HDP) [20] is chosen as our topic modeler due to its ability to infer the number of topics automatically, hence reducing tuning effort and making our framework become much more automatic.

   We also provide results from Latent Dirichlet Allocation (LDA) topic modeling [21]. While this process is less automatic and some amount of tuning is required, it could provide a significant performance improvement when properly tuned. Note that even if we only use HDP and LDA in our experiments, our framework is modular. As such, other topic modelers can also be easily applied. This allows our framework to be adjusted according to data structure and the actual situation in which it will be employed.

The outputs of this process are bug reports in topic vector space and the top words 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 words list.

3.2. *Clustering Phase*

To categorize bug reports without the need for any prior knowledge of the data, a clustering technique is used. In this phase, topic membership vectors representing bug reports from the previous phase are grouped and categorized according to their textual similarity. Any clustering algorithm can be used in this phase, depending on preferred categorization criteria and available information about dataset structure.

This available information can easily be utilized, which allows a boarder range of algorithm to be used and improves the categorizing performance. For example, when the number of categories existing in the dataset is known, a clustering algorithm that can specify the number of clusters should be employed. On the other hand, if there is no known information about the dataset structure, an algorithm that can automatically infer the number of clusters on its own should be used instead.

The clustering methods used in our experiments are Expectation Maximization (EM) and the X-means algorithm [24]; both are methods that can automatically estimate the number of clusters. All experiments using X-means in this paper set a minimum number of clusters to 2 and maximum to 10. Both of the clustering algorithms are employed using Weka 3.6.3. [25] The output of this process is the cluster assignment, which indicates which bug report belongs to which cluster. Figure 2 summarizes our Clustering phase.

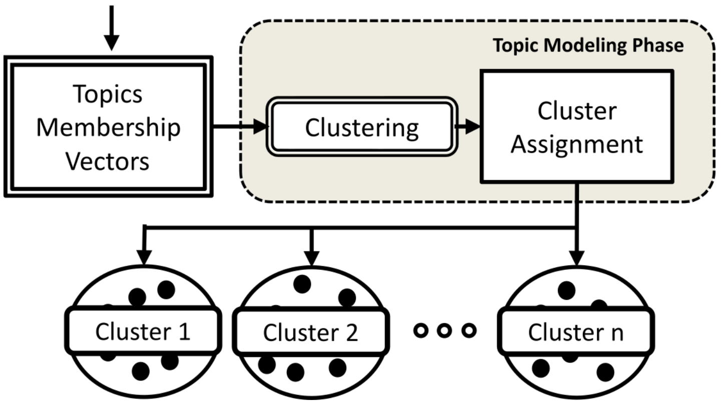

Fig. 2. Diagram of our Clustering phase

3.3. *Cluster Labeling Phase*

Understanding the result of clustering is difficult because normal clustering techniques do not provide adequate descriptions to understand the meaning behind their results. This means that the task of interpreting the meaning of each cluster is often left to its user.

While there are many algorithms already proposed for cluster labeling [12–14] and topic naming [26, 27], they are not directly applicable for our work, which performs clustering in the topic space. Most cluster labeling algorithms are designed for word-level clustering, not for topic-level. Similarly, just finding a set of important topics and then using topic naming algorithms will either limit the labeling result to a set of terms or disregard the term co-occurrence between topics. With this in mind, we suggest three algorithms for labeling clusters in topic vector space.

Title of the closest instance: Cluster Labeling: One of the simplest methods for labeling a cluster of documents, this method sorts the instances (bug reports) in each cluster according to their distance from the cluster center, and then uses the title of the closest instance to label the cluster.

While this method is both straightforward and easy to implement, its result can be misleading; largely because the result depends on how informative the report's title is in that particular bug report.

Adjusted Jensen-Shannon Divergence: Cluster Labeling: Labeling according to the adjusted Jensen-Shannon Divergence, makes slight modifications to the important word extraction method proposed by Carmelet et al [13]. This method selects a set of top ranking topics for each cluster according to two criteria: First, the Jensen-Shannon Divergence, $\mathrm{D_{JS}}$, between the target cluster and the other clusters, has to be as high as possible. The Jensen-Shannon Divergence is measured for each dimension (topic), from the centroid of the target cluster to the centroid of other clusters. Second, in addition to having the highest possible Jensen-Shannon Divergence, the selected topic has to have an averaged topic distribution higher than the averaged topic distributions of that topic in other clusters. The $\mathrm{MaxD_{JS}}$ can be calculated by Eq.1.

$$MaxD_{JS} = \max_{i=1}^{TopicNum} \left(\sum_{j=1}^{J} P(i) log \frac{P(i)}{M(i,j)} + \sum_{j=1}^{J} Q(i,j) log \frac{Q(i,j)}{M(i,j)} \right) \quad (1)$$

P is the centroid of the target cluster; P(i) is the location of P in $\mathrm{Topic_i}$ dimension. Q is centroid of other cluster; Q(i, j) is the location of other $\mathrm{cluster_j}$ in $\mathrm{Topic_i}$. J is the number of clusters. M(i,j) is computed by Eq.2.

$$M(i,j) = \frac{1}{2 \times (P(i) + Q(i,j))} \quad (2)$$

After obtaining a set of top ranking topics for each cluster, the top words of these clusters are then assigned as the clusters' labels.

Figure 3 shows a diagram of this process.

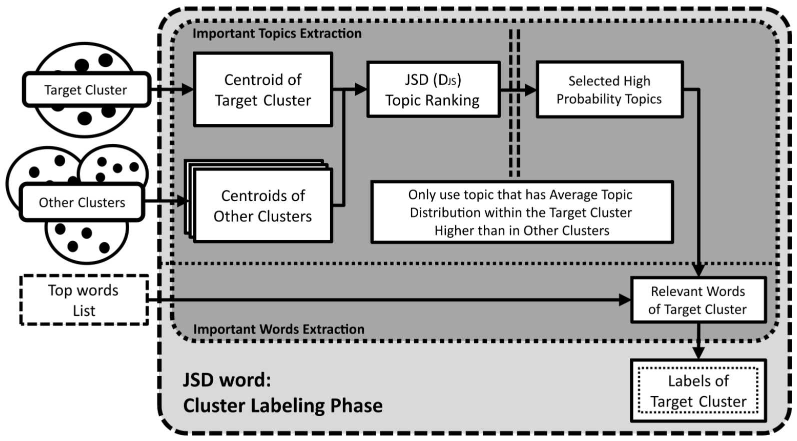
Important Topics Extraction
Target Cluster
Centroid of Target Cluster
JSD (D_{JS}) Topic Ranking
Selected High Probability Topics
Other Clusters
Centroids of Other Clusters
Only use topic that has Average Topic Distribution within the Target Cluster Higher than in Other Clusters
Top words List
Relevant Words of Target Cluster
Important Words Extraction
JSD word: Cluster Labeling Phase
Labels of Target Cluster

Fig. 3. Diagram of Adjusted Jensen-Shannon Divergence: Cluster Labeling

NLP Chunk: Cluster Labeling: This novel method takes term co-occurrence between topics into account, labeling clusters with noun phrases extracted via a NLP chunker. This method consists of four main steps: Important Topics Extraction, Important Words Extraction, Noun Phrase Extraction, and NLP Chunk labeling. These steps are summarized in Figure 4.

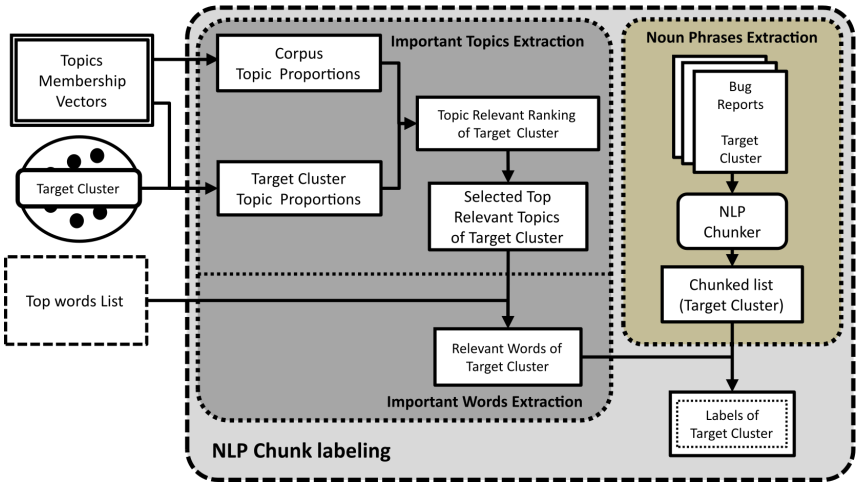
Topics Membership Vectors
Target Cluster
Top words List
Corpus Topic Proportions
Important Topics Extraction
Target Cluster Topic Proportions
Topic Relevant Ranking of Target Cluster
Selected Top Relevant Topics of Target Cluster
Relevant Words of Target Cluster
Important Words Extraction
NLP Chunk labeling
Noun Phrases Extraction
Bug Reports Target Cluster
NLP Chunker
Chunked list (Target Cluster)
Labels of Target Cluster

Fig. 4. Diagram of NLP Chunk: Cluster Labeling

(1) **Important Topics Extraction:** The inputs to this step are topic membership vectors, the output of 3.1, and the assignment of the interested cluster from 3.2. A set of important topics for each cluster is acquired by ranking topics according to the ratio between each cluster's average probability distribution in that cluster and the average probability distribution of the entire dataset. The idea is that the topics with the highest probability distribution in the specific cluster when compared to the corpus average are likely to be the best representatives for that cluster. This phrase, however, does not directly take each topic word distribution into account, but only selecting a top important topic for each cluster.

The average probability distribution of $\mathrm{topic_i}$ in $\mathrm{cluster_j}$ is equal to the sum of topic i proportion from all documents in cluster j divided by the number of documents in cluster j. The equation is show in Eq. 3.

$$AVGTopicDistribution(Topic_i, Cluster_j) = \frac{1}{n_j} \sum_{Report_x \in Cluster_j} TopicDistribution(Topic_i,\, Report_x) \quad (3)$$

The variable $\mathrm{n_j}$ is the number of instances (bug reports) that are assigned to cluster$_j$. For the entire dataset average distribution, simply replace cluster$_j$ with the set of all instances in the bug report corpus.

The ratio of each Topic$_i$ in cluster$_j$ can then be calculated by dividing the average Topic$_i$ proportion of cluster$_j$ with the average Topic$_i$ proportion of the entire document corpus. The equation is shown in Eq. 4.

$$Ratio(Topic_i, Cluster_j) = \frac{AVGTopicDistribution(Topic_i, Cluster_j)}{AVGTopicDistribution(Topic_i, EntireCorpus)} \tag{4}$$

All topics are ranked according to their ratio, and then output as a set of the top relevant topics. These topics can also be used to generate the label list of the cluster, by selecting the top word of each topic in the set. This relevant topics set is used as input for the following step.

(2) **Important Words Extraction:** The previous step does not take the word distribution into account, this means it will neglects the second and later ranking words in the topic even though they might have probability distribution closed to the top one. Since same word can appear multiple times in the different topics, it is entirely possible that using only the topic distribution ratio will result in discarding potential representative words. To solve this problem, we propose using Eq. 5 to ranking words obtain from the set of top relevant topics.

$$WordScore(word_k, Cluster_j) = \sum_{i=1}^{TopicNum} Ratio(Topic_i, Cluster_j) \times P(word_k|i) \tag{5}$$

P(word$_k$|i) represents the probability distribution of word$_k$ in topic$_i$, and this combined with the summation of all relevant topics makes it possible for subsequent words to obtain a higher TermScore than a top word if it appears multiple times in several topics. Each word is then sorted according to their WordScore into a relevant word list, which can also be used as a label list. This relevant word list is used as input for the last step, NLP Chunks labeling.

(3) **Noun Phrases Extraction** In this step, a set of noun phrases are extracted from the interested cluster. The inputs for this step are raw bug reports from the interested cluster. A Natural Language Processing (NLP) chunker from Apache OpenNLP [16] Library version 1.5.3 is used to extract a set of noun phrases from these bug reports. A noun phrase is used instead of other phrase types, as it is more informative and more prevalent [28]. The output of this phrase, the chunked list, is used as input for the next step.

(4) **NLP Chunks Labeling** This step labels each cluster with its most representative noun phrases. The inputs for this step are a relevant word list and a chunked list. Each noun phrase in the chunked list is scored according to Eq. 6.

$$PhraseScore(Phrase_m, Cluster_j) = \sum_{k \in Phrase_m} WordScore(word_k, Cluster_j) \tag{6}$$

The $\mathrm{Phrase_m}$ score in $\mathrm{Cluster_j}$ is the summation of the WordScore of all words in $\mathrm{Phrase_m}$. After the score calculations are complete, all phrases are then sorted using their score.

However, we found that using this scoring method alone will usually result in only a few of the most important words dominating the label list. To give the label list more variety, the score of words that already appeared several times in the phrase list are adjusted to have less and less impact on the PhraseScore. This is done by reducing the score of the sorted phrase list, from Eq.6, using Eq.7.

$$ReductionScore(Phrase_m, Cluster_j) = \sum_{k \in Phrase_m} WordScore(word_k, Cluster_j) \times \frac{X_k^2}{(X_k^2 + 1)} \quad (7)$$

X is the number of times $\mathrm{word_k}$ already appears in the higher ranking phrases. After this score reduction, the phrase list is then sorted again to represent its new ranking. The output of this process is a noun phrase list representing the interested cluster.

4. Experimental-Design

We describe the measurements used, the experimented datasets, and the experimental-design in this section. The goal of these experiments is to demonstrate the performance of the proposed framework in various perspectives, and the specific goals for each experiment are given in the experimental-design subsection.

4.1. *Measurements*

In this section, we describe five measurements used in our experiments: Cluster's Purity, Accuracy, Precision, Recall and F-measure (F1). All clustering results are evaluated via external evaluation, by using known class labels.

Cluster's Purity Cluster purity is a simple measure to evaluate how pure the cluster's result is according to the predetermine categories, classes. This measure can be calculated using Eq.8.

$$Purity(ClusterResult, Categories) = \frac{1}{n}\sum_j max_i|Cluster_j \cap Class_i| \quad (8)$$

The ClusterResult is a set of Clusters, $\{\mathrm{Cluster_1}, \mathrm{Cluster_2}, \ldots, \mathrm{Cluster_j}\}$ while Categories represent the set of categories (classes) $\{\mathrm{Categories_1}, \mathrm{Categories_2}, \ldots, \mathrm{Categories_i}\}$. The n variable is the total number of instances in the entire data corpus.

To summarize, each cluster will be assigned to its predominant class. However, in the case where there is a class not assigned to any cluster, a cluster containing the highest ratio between the number of unassigned class and the number of instances in that cluster, will be reassigned.

Accuracy Accuracy is similar to the cluster's purity, but it is used on the classification result instead of clustering. It grants a basic idea of how many instances are correctly classified in relation to the total number of instances. The arithmetic equation is shown in Eq. 9.

$$Accuracy = \frac{|Correctly\, Classified\, Instance|}{Total\, number\, of\, instance} \tag{9}$$

F-measure F-measure, or F1, is used to evaluate the results of both clustering and classification. In the case of clustering, the same method as in cluster's purity is used to assign a class to each cluster. This measure is a harmonic means of precision and recall, which can be computed via Eq. 10 and Eq. 11, respectively. The calculation is applied to each class separately, then combined using a weight average.

A true positive indicates the number of instances in the interested class, a positive class, that are correctly classified while a false positive represents the number of instances from other classes mis-assigned into the positive class. A false negative is the number of positive classes that are incorrectly classified to any other classes.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{10}$$

This precision provides us with the insight as to how accurate the prediction is, in relation to the number of the predictions made.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{11}$$

Recall indicates the completeness of the prediction in the interested class. In other words, recall shows how many instances in the interested class that the classifier misses, in relation to the actual number of instances in the interested class.

When the two measures above are calculated, the f-measure can be computed using Eq. 12.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{12}$$

After obtaining the f-measure for each class, the final overall result is summarized using the weight average which takes the number of instances from each class into account.

4.2. *Datasets*

The experimented bug reports are from three datasets of our previous research [5], and which are manually classified by experts using a fixed set of rules. To represent our method performance by two possible situations, we use two sets of binary classes to evaluate its performance. First is to distinguish between the different projects, to represent this problem; namely, the bug reports from the Lucene and Jackrabbit (JCR) projects are combined. We then try to categorize the merged dataset back into its original categories: Lucene and JCR. This dataset is used in Experiment 2.

Second, we classify bug reports into their corresponding types. Originally, these bug reports were categorized into six categories. However, as the increasing number of categories make it harder to accurately classify bug reports [6–8], the number of categories is reduced to preserve the process performance. The used categories are BUG and Other Requests types; the Other Requests type is a composite type consisting of four non-bug report types: Improvement, Request for Enchantment (RFE), Task and Test. These datasets are used in Experiment 1,3,4 and 5. Each project class distribution is shown in Table 1.

Table 1. Class distribution of the experimented datasets

Project	Total Number of Instances	Number of "Bug" Type	Number of "Other Requests" Type
Lucene	2382	1037	1345
Jackrabbit (JCR)	2328	1213	1115
HTTPClient	731	469	262

4.3. *Design For Each Experiment*

There are a total of five experiments in this research, and each is designed to evaluate the different aspects of our method. The detail and idea behind each experiment are given below.

Experiment 1: Comparison between Bag-of-Words and Topic Modeling

The first experiment is designed to evaluate the performance of different ways to represent a document corpus. We experiment on a well known word-level approach, the bag-of-words model, using both term frequencies and term weighting. Then we compare their results with the topic modeling approaches.

First, the document corpus is transformed into term frequencies; each instance is a bug report and the value in each dimension represents the frequency of the associated term appearing in that particular bug report. After the transformation, the dataset is further processed to eliminate out the terms with an insignificant appearance, so that dimensions with a frequency below twenty are filtered out.

Tf-idf stands for term frequencyinverse document frequency, and it is a term weighting method regularly applied to the bag-of-words model. The tf-idf scores for each term in each document are calculated via the following equations.

$$tf(term_{k,}Doc_p) = freq(term_{k,}Doc_p) \tag{13}$$

Tf can be computed using Eq. 13. It represents how many times $\mathrm{term_k}$ appears in $\mathrm{document_p}$.

$$idf(term_{k,}Doc_p) = log\frac{Number\,of\,Documents\,in\,the\,Corpus}{1 + Number\,of\,Documents\,Containing\,term_k} \tag{14}$$

Idf is calculated by Eq. 14. This measure indicates the significant of the $\mathrm{term_k}$, according to its appearance in the entire data corpus. The less the term is found in other documents, the better its score.

$$tf - idf(term_{k,}Doc_p) = tf(term_{k,}Doc_p) \times idf(term_{k,}Doc_p) \tag{15}$$

After both tf and idf are obtained, the tf-idf can be calculated using Eq. 15. This tf-idf represents the importance of $\mathrm{term_k}$ in $\mathrm{document_p}$.

Topic modeling is a natural language processing (NLP) process, and in this research, it is used to transform the bug report corpus into a vector matrix of documents and topics. Each topic modeling's topic is a group of words assumed to represent the same or similar meaning. There are two topic modelers used in this experiment, Latent Dirichlet Allocation (LDA) [21] and the Hierarchical Dirichlet Process (HDP) [20]. LDA is a method commonly used in research [8]; however, the method requires the user to specify the number of topics. This makes the method rather inconvenient when there is no prior knowledge about the dataset.

The second topic modeler, HDP, has the advantage that it can infer the number of topics automatically. The number of LDA's topics in this experiment are set to be similar to the HDP's topics. Since the output of each run from these topic modelers can be slightly different, due to their probabilistic property, the experiments are done on three separate runs for each topic modeler. These results are then averaged and shown in section 5.1.

These two topic modeling methods are experimented on to decide which method is more suitable for our categorization method.

Four document vector corpora are used as an input for X-means clustering [24]. The number of minimum clusters is two and the maximum is ten; each cluster is labeled according to the method described in subsection 4.1. As X-means results can change according to the cluster seeds used, we perform an experiment on one thousand different cluster seeds for each corpus, then average the results. All approaches are evaluated by five measurements described in section 4.1. This experiment is done on HttpClient and JCR dataset, by categorizing bug reports into BUG and Other Requests.

Experiment 2: Categorizing bug reports from two different projects

This experiment measures the ability of our method when trying to distinguish between two groups of bug reports that are not closely related. To achieve this, the bug report from the projects Lucene and JCR, are combined. The HDP is then employed to the topic model to the combined dataset, according to the results in Experiment 1. We then try to categorized bug reports from this dataset into their actual projects using four different approaches.

The first two are our unsupervised method employed using two different clustering algorithms, X-means [24] and Expectation Maximization (EM) Clustering [29], both capable of inferring the appropriate number of clusters each on their own.

For comparison, we compared our unsupervised method results with two classification algorithms; J48, an implementation of the C4.5 decision tree [30] and Logistic Regression [31]. These two supervised methods are used as an upper bound of the unsupervised method, as they are created from the prior knowledge (labeled training dataset) in which we assume that it may not exist or is very costly to obtain. All methods are evaluated in term of the cluster's purity/accuracy and f-measure. The results shown are the average values obtained from three separated HDP's runs. Both classification methods are trained and tested using 10-fold cross-validation.

Experiment 3: Categorizing bug reports into Bug and Other Requests

The goal of this experiment is to evaluate our method performance in a more complex situation, and to distinguish between the different types of bug reports in the same project. Different from the previous experiment, each project is separately experimented on; the experimented on projects are HTTPClient, JCR, and Lucene. The results of the X-means clustering are from the average of one thousand runs. Aside from this, the settings of this experiment are similar to 4.3.

Experiment 4: Cross-Project Classification Comparison

In this experiment, we look into more competitive methods of handling the lack of prior knowledge. Generally, when we want to categorize a project in which no labeled training dataset is available, we usually employ cross-project classification [9]. Cross-project classification trains a classification model from the different but similar projects that already have training data readily available. While there are still some limitations, for many projects, this method is certainly more realistic than creating a whole new training dataset from scratch. Here, the cross-project classifiers are trained using training dataset from first one, and then two projects, and is then compared with our unsupervised approach.

Experiment 5: Cluster Labeling Results Comparison

This experiment shows the results of several cluster labeling algorithms. Most cluster labeling research is done on the word-document vectors [13], not the topic ones, so that some processes are required to make them capable of properly handling topic-level clusters. In this experiment, there is a total of three labeling methods, as described in section 3.3

5. Results

In this section, we present five experimental results as described in section 4.3. The experiments are performed on Lucene, Jackrabbit (JCR) and HTTPClient datasets.

5.1. *Experiment 1: Comparison between Bag-of-Words and Topic Modeling*

While previous studies have shown the advantages of topic modeling in classification [8] and information retrieval (IR) [18,19], its efficiency in bug report clustering has hardly been explored. The goal of our experiment is to verify the benefits of using topic modeling and selecting the best topic modeler that is most compatible with our framework. Four different methods for representing the document corpus are investigated: Tf, Tf-idf, LDA, and HDP. The set of document vectors from each method is then used as input for X-means clustering. The experiment is done on HTTPClient and JCR datasets.

Table 2 presents the result of this experiment. The lower part of the table shows F-measure: per class and Weight Average summarizing the overall performance from both classes

Table 2. Experiment1:Comparison between different Dataset Dimension

		HTTPClient				JCR			
		Bag-of-Words		Topic Model		Bag-of-Words		Topic Model	
		Tf	Tf-Idf	LDA	HDP	Count	Tf-Idf	LDA	HDP
Overall	Num Dim	1220	1220	52	52.67	1922	1922	52	55.67
	Accuracy	0.645	0.573	0.559	0.596	0.533	0.553	0.620	0.625
BUG	Precision	0.646	0.676	0.747	0.711	0.529	0.585	0.639	0.638
	Recall	0.991	0.725	0.490	0.641	0.938	0.653	0.661	0.649
	F-measure	0.782	0.629	0.539	0.656	0.676	0.576	0.595	0.625
Other	Precision	0.712	0.625	0.452	0.459	0.601	0.580	0.642	0.640
	Recall	0.025	0.302	0.684	0.516	0.091	0.443	0.575	0.600
	F-measure	0.048	0.224	0.510	0.467	0.155	0.436	0.564	0.605
WeightAVG	Precision	0.669	0.658	0.642	0.621	0.564	0.583	0.640	0.639
	Recall	0.645	0.573	0.559	0.596	0.533	0.553	0.620	0.625
	F-measure	0.519	0.484	0.529	0.588	0.426	0.509	0.580	0.616

From Table 2, we see that even with the removal of the terms with less frequency, the bag-of-words approaches still have a very high number of dimensions. This results in a very sparse high-dimensional space degrading the clustering efficiency. In both datasets, the topic model approaches perform better than the word-level approaches, and have higher weight average F-measures in all situations. While the

Tf method has better accuracy in HTTPClient dataset, its performance in Other class is clearly suffered due to a heavy bias toward the majority class (BUG). This confirms that using topic modeling can efficiently reduce the sparseness of the bug report dataset and project it into a space more suitable for clustering.

When compared between the two topic modelers, the HDP method is shown to be more compatible with our method. The results of using HDP document vectors are better than LDA in almost all aspects. Further more, HDP is also more automated than LDA,and is capable of assuming the number of topics by itself. With these two factors combined, we are confident that HDP topic modeling is more compatible with our method for representing a set of document vectors.

5.2. *Experiment 2: Categorizing bug reports from different projects*

This experiment evaluates the performance of our unsupervised framework in the simple task of capturing the structural differences of bug reports from two software engineering projects. In this experiment, our framework adopted two clustering methods, X-means and Expectation Maximization (EM), with both capable of assuming the number of cluster on their's own. The upper bounds are created using two, well-known classification algorithms: J48 and Logistic Regression. Both supervised and unsupervised learning are performed on the same HDP dimensions. The result is shown in Table 3.

Table 3. Experiment2: Categorizing bug reports from different projects

		Train 2 dataset/ Lucene + JCR			
Overall	Num Instance	4710			
	Num Topic	57.333			
	Method	Xmeans	EM	J48	Logistic
	Accuracy	0.8671	0.7939	0.9155	0.9646
F1	JCR	0.8675	0.7891	0.9147	0.9640
	Lucene	0.8666	0.7914	0.9163	0.9653
	WeightAVG	0.8670	0.7876	0.9157	0.9647

As shown in Table 3, the performance of our method, while lower, is still comparable to the supervised learning one. For the X-means approach, its average F-measure is 5.62 and 11.26 percent lower than its upper-bounds, J48 and Logistic Regression, respectively. On the other hand, the EM clustering performance is not as decent; its performance is 16.26 and 22.49 percent lower, despite having a higher number of clusters (12.667 while X-means only created 4).

5.3. *Experiment 3: Categorizing bug reports in one project into Bug and Other Requests (In-Project Classification)*

This experiment categorizes bug reports from the one project into two different classes: BUG and Other Requests. The experiments are done on three different datasets: HTTPClient, JCR, and Lucene. Bug reports in each project are transformed into three sets of topic document vectors, using three separated HDP runs with the same setting. In each set, we perform 1,000 runs with the X-means approach for a total of 9,000 runs in this experiment.

The result is shown in Table 4. The value in each cell is averaged from three HDP runs, except for the X-means, which is from 3,000 runs.

Table 4. Experiment3: Categorizing bug reports in one project into Bug and Other Requests

		HTTPClient			JCR			Lucene		
Overall	Instance	731			2328			2382		
	NumTopic	52.667			55.667			47.667		
	Methods	Xmeans	J48	Logistic	Xmeans	J48	Logistic	Xmeans	J48	Logistic
	Accuracy	0.596	0.626	0.710	0.625	0.636	0.745	0.601	0.633	0.710
F1	BUG	0.656	0.717	0.787	0.625	0.632	0.732	0.438	0.565	0.634
	OtherReq	0.467	0.448	0.550	0.605	0.639	0.756	0.688	0.682	0.760
	WeightAVG	0.588	0.621	0.702	0.616	0.636	0.745	0.579	0.631	0.705

Table 4 illustrates how this task of categorizing bug reports from the same project is a much harder task; all approaches perform worse here compared to the previous experiment.

As shown in Table 4, the F-measures of our X-means approach and J48 are still in close proximity, with a 5.90 percent difference in the average F-measure from three projects. This shows that even in a complex environment the performance of our framework is comparable to J48, the well-known classifier.

When compared to Logistic Regression, the difference here is significant; our method average F-measure is 0.123 (20.71 percent) lower. However, as mentioned previously, sometimes obtaining a training dataset in the first place can be quite challenging. In those situations, using our unsupervised framework can provide a good solution to the problem, as no training set is required.

5.4. *Experiment 4: Comparison between Our Method and Cross-Project Classification*

While Experiments 5.2 and 5.3 are performed in a circumstance where the supervised approach has a very clear advantage, learning directly from the desired project, this experiment provides a more competitive situation.

Here, our unsupervised framework stays the same as in Experiment 5.3, while the classification models are trained using labeled datasets from other projects [9]. From the three datasets: HTTPClient, JCR, and Lucene, cross-project classification is performed using all possible combinations. Bug reports are transformed into three sets of topic vectors using three separated HDP runs, and their results are averaged.

Figure 5 presents this experimental result. The Y-axis is an average of the weighted F-measure while the X-axis shows the used methods, which are grouped according to the testing projects.

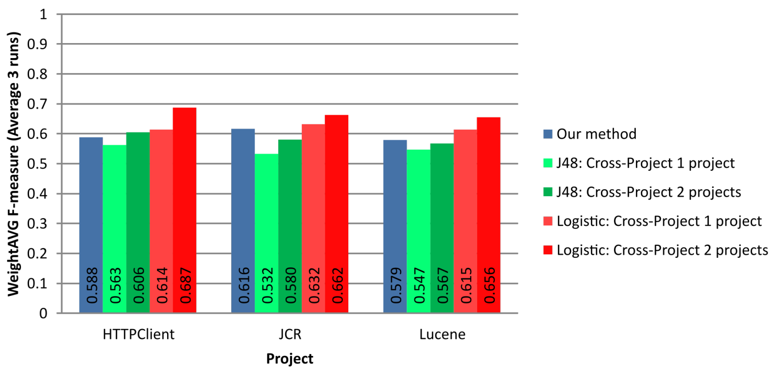

Fig. 5. F-measure comparison between Our Method and Cross-Project Classification

Figure 5 shows that, without any training data, the performance of our framework is better than the J48 by 5.16 percent. However, when compared to the Logistic Regression, its performance is 7.63 percent lower. While this indicates that if labeled datasets from similar projects are available, Logistic Regression can perform significantly better than our framework, but our framework can still provide a good alternative. As in many scenarios, even with cross-project classification, a labeled data matched with the desired categories can still be hard to acquire due to the numerous ways that bug reports can be categorized. For example: the types of package conflict [4], the types of bug report [5], the types of bugs, etc.

The four experiments above demonstrate that our framework is capable of classify bug reports automatically and reliably.

5.5. *Experiment 5: Cluster Labeling Results*

This experiment evaluates our cluster labeling process. Because there are only a few methods that can be directly applied to the cluster in topic space, we modify a cluster labeling method proposed by Carmelet et al [13]. This modified labeling method (JSD) and labeling using the title of the closest instance method (Section3.3) are employed as to provide comparison methods for cluster labeling.

Table 5 shows part of the labeling result, and this result is from one of the runs on the HTTPClient dataset from Experiment 2. The columns in this Table are the original cluster ID, while each row shows the cluster labels. The labeling methods are JSD, Labeling using the NLP Chunk, and the Title of the Closest Instance. These method are described in Section 3.3.

On the lower part of the table are rows that represent the number of instances in each class: BUG and Other Requests. Each cluster is labeled by its majority class; the detail method is described in Section 4.1.

Table 5. Experiment5: Cluster Labeling Results

	HttpClient		
	HDP 2 run		
	Cluster 1 BUG	Cluster 2 Others	Cluster 3 BUG
Jensen Shannon Divergence (JS-D)	**connection connection connection** abstracthttpclient.java **timeout** getmethod rulebasedcollator logs **exception debug**	cache patch serializable equals encoding httpclient mockito socket encoding configuration	**cookie** uri http **authentication** http guide file **authentication debug** http
Labeling using NLP Chunk	**http request connection timeout**, **multi threaded connection manager**, **closed socket**, **s catch stale connection**	chunked encoding, alive debug headers transfer encoding, transfer encoding header, cache responses headers	**cookie domain attribute**, **cookie domain**, **domain public credentials**, **domain com cookies**
Title of Closet Instance	**Connection is not released back to the pool if a runtime**, **An IOException or RuntimeException leaves the underlying socket in an undetermined state**	RequestEntity EntityEnclosingMethod have inconsistent Javadocs use deprecated variables, Do not consume the remaining response content if the connection is to be closed	Invalid redirects are not corrected, **DigestScheme.authenticate returns invalid authorization string when algorithm is null**
Num Class BUG	89	163	217
Num Class Others	30	135	97

From the example result from Table5, clusters 1 and 3 are labeled as BUG classes. Their results are reliable, having an internal cluster purity around 0.7. On the other hand, cluster 2 which is assigned to Other class has not quite as good a result. The bug reports within it are mixed, containing a high number of reports from both classes. The three labeling methods are present in Table5 label cluster with different granularities: word, phrase and sentence level. We only discuss the quality of labels from cluster 1 and 3, as their purity are quite high so that it is easier to identify the main subjects in each cluster.

For the word level, with labeling using Jensen Shannon Divergence (JSD), we can see that while some labels are representative and make sense; e.g. ”connection”, ”timeout”, ”exception” and ”debug” in cluster 1, the majority of its reports are related to ”connection problems”. However some of its labels, such as ”abstracthttpclient.java”, ”getmethod” or ”rulebasedcollator”, are clearly not that suitable to be a cluster label. Out of ten suggested labels, the numbers of usable labels are six and four, in cluster 1 and 3, respectively. This shows that using a word level approach for labeling a cluster of bug reports can introduce some problems since, unlike in a general document, many software engineering terms are too specific to be understood without their context.

For the sentence level, labeling is done using the title of the closest instance to the cluster center, and while it can provide a decent idea of what bug reports in that cluster are related to, there are a few problems. First, it largely depends on how informative the title is. As the report’s title is written by just one submitter, whose knowledge related to that particular report is not guaranteed, it is very possible that its title will be misleading or not contain enough information. For example, the title ”Invalid redirects are not corrected” is too short and can not be easily assigned to either of the two categories.

For the phrase level the generated labels are quite representative. In cluster 1, all of the top four labels are related to ”connection” which is the main topic of cluster 1. Moreover, three out of these labels are associated with ”connection problems”, topics strongly correlated with the BUG category to which the majority of reports in cluster 1 were classified. The same trend can be found in cluster 3, related to ”authentication”, which means that all the suggested phrases are representative as they all contain words like ”cookie” or ”credentials”. This indicates that this labeling method can provide a high quality set of labels that are both compact and representative.

6. Threats to Validity

This research is subject to threats to validity induced by the limitations of our approach. The most important threats are listed below.

6.1. *Measurements used*

The evaluation measurements used in this research are: cluster purity, accuracy, precision, recall and F-measure. While these measurements are well-known and commonly used in many past studies, we still can not guarantee that the result would be the same if other measurements are used instead.

6.2. *The categories of bug reports*

As mentioned previously, bug reports can be categorized in many ways. Our experiments are performed only on two sets of categories, and as such, the performance of our framework in other sets of categories still cannot be guaranteed.

6.3. *Experimented Datasets*

Our experiments are performed on the dataset of previous study. Even if these datasets are manually inspected and cross-validated, some errors might still remain, which could slightly change the result of our experiments. The research subjects of our experiment are also limited; all experiments are performed on bug reports of projects written in Java using the JIRA bug tracker. This data might not be representative of other programming languages or bug tracker systems.

6.4. *Heuristics parameters*

Some parameters used in our approach and evaluation are heuristics. These parameters are:

- The number of top words for topic modeling is set at 50 for both the HDP and LDA topic modelers
- The minimum and maximum number of clusters for the X-means algorithm (Set to two and ten receptively)
- Number of suggested labels for each label algorithm in Experiment 5.4

6.5. *Coverage of other possible parameters and approaches*

Due to the modular property of our framework, there are simply too many ways that it can be adjusted. While this property is intended and beneficial, there are just too many possible setting combinations for us to experiment on. For instance, instead of LDA or HDP, the topic modeling algorithm can easily be changed, the same could be said for the clustering algorithm as well.

7. Related Work

Nowadays, the bug report system is undeniably a fundamental part of any software engineering project. Aside from its usual benefits, it also provides a good quantity of information which can be extracted and analyzed. A large number of software engineering systems and research are based on this extracted information [1,32–34]. They are used in the bug triaging systems [33], in defects prediction [35] to predict software defects, and also in many other tasks [1,32,34]. However, as previous researches [5, 23] point out, a significant number of bug reports are actually misclassified..

There is some research related to bug reports categorization. Some [5] use an manual inspection to classify bug reports according to their interested categories. While using a traditional approach like this has its own benefits and is necessary in some situations, it usually requires more time and human resources than what most projects can actually afford.

To solve this problem, previous research proposes an automated system for categorizing bug reports. Most research uses classification [7], a supervised learning approach, to learn concepts from a labeled dataset. The previous research by Thung et al. [35] classified a type of defect, using code features and bag-of-word document vectors. Anvik et al. [36] use classification in their bug triaging systems. Some research created systems for detecting bug reports that are actually related to bug. Antoniol et al. [23] proposes using tf-idf word vectors to create a classification model, while some researches [8, 17] suggest that the LDA [21] topic vectors should be used instead.

While the supervised learning approach clearly has its advantages, there are also some limitations. Obtaining the historical data to use as a training dataset is, usually, not an easy task. Unless it is already readily available, a lot of human effort will be required to gather and correct that dataset [5]. This issue is worsened by the fact that many software companies won't release their data to the public, making a difficult problem become even more challenging. Furthermore, using classification can limit what we learn from the data. For instance, if two groups well separated are formed into one category, it might be better to split that category into two.

To solve the lack of the historical data problem, cross-project classification is usually employed. Some research studies the performance of cross-project defect prediction [10, 11, 37]. While this method can solve the lack of the data problem, it also comes with a trade-off. Most of the time, the classification performance will become significantly lower [10].

Although not as common as supervised learning, unsupervised learning has been used in software engineering areas. Previous study by Anvik [38] proposes using clustering in the bug triaging while Podgurski et al. [39] cluster software failure reports to determine similar errors.

However, using clustering alone introduces a new problem; its result can be hard to interpret. As such, it would be much more beneficial if each cluster is labeled by what it is most related to, and cluster labeling algorithms are created for this purpose. These algorithms come in several granularity levels: term, N-gram and sentence. In 2006, Carmel et al. [14] suggested the cluster labeling approach for labeling general documents using JSD terms extraction, which they further improved by utilizing Wikipedia in 2009 [13]. There is also a previous study for a bug report summarization which uses a word document vector [12].

8. Conclusion

In this paper we propose a framework for categorizing bug reports automatically by utilizing topic modeling and clustering algorithms. Compared to the traditional supervised learning method, our framework provides the definitive benefit of not requiring any training dataset, while still having comparable, albeit lower, performance to the supervised approach. Our framework could be deployed to automatically classify bug reports for a newly deployed project, to categorize bug reports in to several groups based on its text or to help as a label suggestion system for manual data inspection.

In addition, we also presented a new cluster labeling method that can be used in topic space. Different from previous approaches, it takes into account both topic and word distribution. Its labels are also created from a noun phrase, making them both compact and meaningful.

Future work is to further improve the categorization performance while still retaining its nonparametric and non prior knowledge properties. To further prove the generality of our framework, we also aim to do more experiments on a wider range of categories, projects, and bug tracker systems.

Acknowledgements

This work has been supported by JSPS KAKENHI Grant Number 26540029, and Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers: Interdisciplinary Global Networks for Accelerating Theory and Practice in Software Ecosystem.

References

[1] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. How long will it take to fix this bug? In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 1–, Washington, DC, USA, 2007. IEEE Computer Society.

[2] Hongyu Zhang, Liang Gong, and Steve Versteeg. Predicting bug-fixing time: An empirical study of commercial software projects. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1042–1051, Piscataway, NJ, USA, 2013. IEEE Press.

[3] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 495–504, New York, NY, USA, 2010. ACM.

[4] Cyrille Artho, Kuniyasu Suzaki, Roberto Di Cosmo, Ralf Treinen, and Stefano Zacchiroli. Why do software packages conflict? In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 141–150, Piscataway, NJ, USA, 2012. IEEE Press.

[5] Kim Herzig, Sascha Just, and Andreas Zeller. It's not a bug, it's a feature: how

misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press, 2013.

[6] Nachai Limsettho, Hideaki Hata, and Kenichi Matsumoto. Comparing hierarchical dirichlet process with latent dirichlet allocation in bug report multiclass classification. In *Proceedings of the 15th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 1–6, June 2014.

[7] Naresh Kumar Nagwani and Shrish Verma. A comparative study of bug classification algorithms. *International Journal of Software Engineering and Knowledge Engineering*, 24(01):111–138, 2014.

[8] Natthakul Pingclasai, Hideaki Hata, and Kenichi Matsumoto. Classifying bug reports to bugs and other requests using topic modeling. In *Proceedings of the 20th International Conference on Software Engineering, Asia-Pacific*, volume 2, pages 13–18, Dec 2013.

[9] Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, 19(2):167–199, 2012.

[10] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 91–100, New York, NY, USA, 2009. ACM.

[11] Ilan Gabriel Caron, Alan W Carter, Dennis Mark Canady, and Tom Corbett. Cross-project namespace compiler and method, July 4 2000. US Patent 6,083,282.

[12] Senthil Mani, Rose Catherine, Vibha Singhal Sinha, and Avinava Dubey. Ausum: approach for unsupervised bug report summarization. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 11. ACM, 2012.

[13] David Carmel, Haggai Roitman, and Naama Zwerdling. Enhancing cluster labeling using wikipedia. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 139–146, New York, NY, USA, 2009. ACM.

[14] David Carmel, Elad Yom-Tov, Adam Darlow, and Dan Pelleg. What makes a query difficult? In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 390–397. ACM, 2006.

[15] Nachai Limsettho, Hideaki Hata, Akito Monden, and Kenichi Matsumoto. Automatic unsupervised bug report categorization. In *Proceedings of the 6th International Workshop on Empirical Software Engineering in Practice*, pages 7–12. IEEE, 2014.

[16] Jason Baldridge. The opennlp project. *URL: http://opennlp. apache. org/index. html,(accessed 2 February 2012)*, 2005.

[17] Kalyanasundaram Somasundaram and Gail C. Murphy. Automatic categorization of bug reports using latent dirichlet allocation. In *Proceedings of the 5th India Software Engineering Conference*, ISEC '12, pages 125–130, New York, NY, USA, 2012. ACM.

[18] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th international conference of Automated Software Engineering*, pages 70–79. IEEE, 2012.

[19] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. How to effectively use topic models for software

engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 522–531. IEEE Press, 2013.

[20] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical Dirichlet processes. *Journal of the American statistical association*, 101(476), 2006.

[21] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *International Journal of Machine Learning Research*, 3:993–1022, March 2003.

[22] Hanna M. Wallach. Topic modeling: Beyond bag-of-words. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 977–984, New York, NY, USA, 2006. ACM.

[23] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. Is it a bug or an enhancement?: a text-based approach to classify change requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, page 23. ACM, 2008.

[24] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, 2000.

[25] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: An update; sigkdd explorations, 2009. *Software available at http://www. cs. waikato. ac. nz/ml/weka.*

[26] Jey Han Lau, Karl Grieser, David Newman, and Timothy Baldwin. Automatic labelling of topic models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1536–1545. Association for Computational Linguistics, 2011.

[27] Abram Hindle, Neil A Ernst, Michael W Godfrey, and John Mylopoulos. Automated topic naming. *Empirical Software Engineering*, 18(6):1125–1155, 2013.

[28] Erik F. Tjong Kim Sang and Sabine Buchholz. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, ConLL '00, pages 127–132, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[29] Xin Jin and Jiawei Han. Expectation maximization clustering. In Claude Sammut and GeoffreyI. Webb, editors, *Encyclopedia of Machine Learning*, pages 382–383. Springer US, 2010.

[30] J Ross Quinlan. *C4.5: programs for machine learning.* Elsevier, 2014.

[31] Alexander Genkin, David D Lewis, and David Madigan. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.

[32] Juan Jose Amor, Gregorio Robles, and Jesus M. Gonzalez-Barahona. Effort estimation by characterizing developer activity. In *Proceedings of the 2006 International Workshop on Economics Driven Software Engineering Research*, EDSER '06, pages 3–6, New York, NY, USA, 2006. ACM.

[33] John Anvik and Gail C Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(3):10, 2011.

[34] Hui Zeng and D. Rine. Estimation of software defects fix effort using neural networks. In *Proceedings of the 28th Annual International Computer Software and Applications Conference*, volume 2, pages 20–21 vol.2, Sept 2004.

[35] Ferdian Thung, David Lo, and Lingxiao Jiang. Automatic defect categorization. In *Proceedings of the 19th International Working Conference on Reverse Engineering*, pages 205–214, Oct 2012.

[36] John Anvik, Lyndon Hiew, and Gail C Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[37] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. Recalling the imprecision of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 61. ACM, 2012.

[38] John Anvik. Automating bug report assignment. In *Proceedings of the 28th international conference on Software engineering*, pages 937–940. ACM, 2006.

[39] Andy Podgurski, David Leon, Patrick Francis, Wes Masri, Melinda Minch, Jiayang Sun, and Bin Wang. Automated support for classifying software failure reports. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 465–475. IEEE, 2003.