Efficient Bug Triaging Using Text Mining

Mamdouh Alenezi, Kenneth Magel, and Shadi Banitaan
Department of Computer Science
North Dakota State University
Fargo, ND 58108, USA
mamdouh.alenezi, shadi.banitaan, kenneth.magel@ndsu.edu

***Abstract*—Large open source software projects receive abundant rates of submitted bug reports. Triaging these incoming reports manually is error-prone and time consuming. The goal of bug triaging is to assign potentially experienced developers to new-coming bug reports. To reduce time and cost of bug triaging, we present an automatic approach to predict a developer with relevant experience to solve the new coming report. In this paper, we investigate the use of five term selection methods on the accuracy of bug assignment. In addition, we re-balance the load between developers by clustering similar bug reports and ranking developers in each cluster based on their experience. We conduct experiments on four real datasets. The experimental results show that by selecting a small number of discriminating terms, the F-score can be significantly improved.**

***Index Terms*—Bug triage, term selection method, text classification, mining bug repositories**

I. Introduction

Software repositories, such as version control systems and bug tracking systems, comprise valuable information about software projects. This information can help to manage the progress of these projects. In the last decade, practitioners have analyzed and mined these software repositories to support software development and evolution (i.e., improve design, refactoring, and maintenance). One of the important software repositories is the bug tracking system (BTS). Many open source software projects have an open bug repository that allows both developers and users to submit defects or issues in the software, suggest possible enhancements, and comment on existing bug reports (e.g., Mozilla, Eclipse, Netbeans, and Apache).

For open source large-scale software projects, the number of daily bugs is so large which makes the triaging process very difficult and challenging. Bug fixing is a tedious and time-consuming process in software maintenance [1]. Many software projects use BTS to manage bug reports submitted by users, testers, and developers [2]. Each new reported bug must be triaged to determine if it describes a meaningful new problem or enhancement, and if it does, it must be assigned to an appropriate developer to fix it. Bug triage is an essential step in bug resolution where a relevant developer is assigned to a new bug.

Most triaging tasks, including bug assignment, rely heavily on manual effort, which is labor intensive and potentially error prone [3]. In practice, due to the frequent changes of software development teams, it is difficult to identify a correct developer who has some experience in fixing similar bugs using manual triage process [1]. For the Eclipse project, Anvik reports that an average of 37 bugs per day are submitted to the BTS and 3 person-hours per day are required for the manual triage [4]; the empirical study by Jeong et al. shows that 44% of bugs have been assigned to the wrong developer after the first assignment [3]. To solve these problems, some machine learning algorithms are employed to conduct automatic bug triage [1], [3], [4], [5], [6]. Most of the bug triage approaches are based on text categorization [1]. However, these approaches suffer from low-quality bug reports which may mislead the triage approach to assign bugs to wrong developers [6], [7]. These approaches also suffer from low recall values [1], [4].

In this paper, we present an approach for bug assignment using both classification and clustering techniques. In this approach, classification is used to build a predictive model which can be used to assign a developer to a newly coming bug report. Five term selection methods are used to reduce the dimensionality of terms and improve accuracy. In this approach, clustering is used to re-balance developers' loads by grouping similar bug reports and ranking developers in each group based on their experience.

To summarize, we make the following key contributions in this work:

- We conduct a comprehensive study on using different term selection methods to evaluate their effectiveness on improving the accuracy of bug assignment.
- We propose an approach to reduce time and cost of bug triaging. The approach has two main steps: 1) Build a classification model using the reduced terms to predict an experienced developer to fix a new reported bug. 2) Cluster similar bug reports based on the best terms obtained from the term selection methods and then rank the developers in each cluster based on their experience. Finally, redistribute the load of overloaded developers to other non-overloaded experienced developers.
- We perform experimental evaluation using four bug reports corpora obtained from real projects.

The rest of the paper is organized as follows: Section II discusses related work. Section III describes the proposed approach. The experimental evaluation and discussion are presented in Section IV. Section V concludes the paper.

II. Related Work

Text categorization dominates the existing bug triage approaches. The first work of bug triage is a supervised text categorization approach using Naive Bayes [2]. Čubranić et al., evaluated their approach using bug reports from the Eclipse project. Their approach achieved 30% accuracy. Anvik et al. [1] extended the work of [2] with a recommendation list, other supervised learning algorithms, and labeling heuristic. They reached precision levels of 57% and 64% on Eclipse and Firefox respectivley. Matter et al. [5] used a vocabulary based expertise model of developers to improve bug triage. Their approach compared vocabulary found in the developers source code with vocabulary found in bug reports. They achieved 33.6% top-1 precision and 71.0% top-10 recall using eight years of the Eclipse project.

Xuan et al. [6] proposed a semi-supervised learning approach with a weighted recommendation list for bug triage to solve the problem of the low quality of bug reports. Their approach improved the classification accuracy of bug triage by up to 6% on the Eclipse project. Park et al. [8] proposed a bug triaging approach which incorporated collaborative filtering and topic modeling to reduce the sparseness of the training data and enhance the quality of the triaging recommendation. Other approaches also address the problem of bug triaging such as the training set reduction approach [9] and the fuzzy-set approach [10].

III. Approach

To reduce the time spent triaging, we present an approach for automatic triaging by recommending one experienced developer for each new bug report. Our approach uses a machine learning algorithm to recommend a developer who may be appropriate for resolving the bug. We formulate the bug triaging process as a classification task where instances represent bug reports, features represent the terms of the report, and the class label represents the developer who fixed this report. This approach can help the triage process in two ways: 1) it may allow a triager to process a bug more quickly, and 2) it may allow a triager with less knowledge about systems and developers to perform bug assignments more accurately. Our approach requires a project to have had an open bug repository for some period of time in which the patterns of who solves what kinds of bugs can be learned.

Figure 1 shows a high level description of our proposed approach. Bug reports are unstructured data which may contain irrelevant words. Therefore, we apply the traditional text processing approach to transform the text data into a meaningful representation. We use the summary of bug reports as a description of bugs. The text processing includes white-spaces, punctuation, numbers, and stopwords removal. After that, the approach constructs a bug-term matrix weighted by term frequency.

Then, different term selection methods are applied to reduce both the dimentionality and the sparseness of data. The next step is to build a classifier using the Naive Bayes approach, a simple and effective classification approach. The classifier is trained using the training data set (bug reports). When a new report arrives, it follows the same steps to produce the reduced bug-term vector, and then it is assigned to a developer using the predictive model.

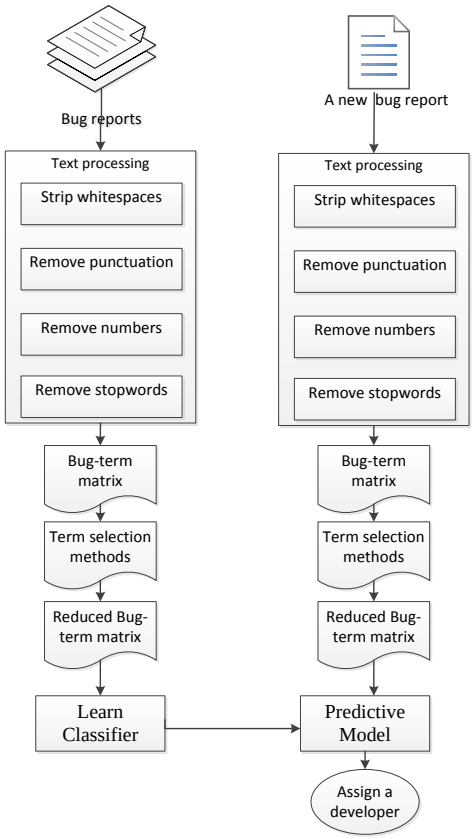


Fig. 1. Our Classification Approach.

A. Representation framework

We have a collection of bug reports, B = $\{b_1, \cdots, b_{|B|}\}$. Each bug report has a collection of terms, T = $\{t_1, \cdots, t_{|T|}\}$, and a class label (developer), $c \in C = \{c_1, \cdots, c_{|C|}\}$. Figure 2 shows a simple bug-term matrix (Corpus) representation.

	t_1	t_2	t_3	t_4	label
b_1	1	0	0	1	c_1
b_2	0	0	1	1	c_2
b_3	1	1	1	0	c_3
b_4	1	1	0	0	c_1
b_5	1	0	1	0	c_1

Fig. 2. A toy example of a bug-term matrix.

B. Term Selection Methods

Term selection methods are used to reduce the high dimensionality of term space by selecting the most discriminating

terms for the classification task. The methods give a weight for each term inwhich terms with higher weights are assumed to contribute more for the classification task than terms with lower weights.

In this work, we use five term selection methods. A short description of these methods appears below.

1) Log Odds Ratio (LOR): Log Odds Ratio measures the odds of the word occurring in the positive class normalized by the negative class. The idea is that the distribution of terms on the relevant documents is different from the distribution of terms on the non-relevant documents. It is defined as follows [11]:

$$LOR(t, c_i) = log \frac{P(t|c)[1 - P(t|\bar{c_i})]}{[1 - P(t|c_i)]P(t|\bar{c_i})}$$

where t and c_i represent a term and a class respectively.

2) Chi-Square (X^2): In statistics, the x^2 test is used to examine independence of two events. The events, X and Y, are assumed to be independent if P(XY) = P(X)P(Y). In term selection, the two events are the occurrence of the term and the occurrence of the class. Terms are ranked with respect to the following equation [12]:

$$CHI2(t, c) = \sum_{t \in \{0,1\}} \sum_{c} \frac{(N_{t,c} - E_{t,c})^2}{E_{t,c}}$$

where N is the observed frequency and E is the expected frequency for each state of term t and class c. $CHI2$ is a measure of how much expected counts E and observed counts N deviate from each other.

3) Term Frequency Relevance Frequency (TFRF): The basic idea behind the TFRF method is that the more high frequency for a term in the positive category than in the negative category, the more contributions it makes in selecting the positive instances from the negative instances. The equation for the TFRF method is computed as follows [13]:

$$TFRF = tf * log_2(2 + \frac{a}{max(1, c)})$$

where a is the number of documents in the positive category which contain the term, c is the number of documents in the negative category which contain the term, and tf is the term frequency. The value 2 is added to the equation to prevent giving a zero weight for a number of other terms.

4) Mutual Information (MI): Mutual information measures the mutual dependence of two random variables. MI computes $X(t, c)$ as the mutual information (MI) of term t and class c. MI measures how much the presence and the absence of a term contributes to making the correct classification decision on c using this equation [12]:

$$I(U; R) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P(U = e_t, R = e_c) * W$$

$$W = log_2 \frac{P(U = e_t, R = e_c)}{P(U = e_t)P(R = e_c)'}$$

where U is a random variable that takes values e_t = 1 if the document contains term t and e_t = 0 if the document does not contain t. R is a random variable that takes values e_c = 1 if the document is in class c and e_c = 0 if the document is not in class c.

5) Distinguishing Feature Selector (DFS): DFS is a new novel term selection method. It provides global discriminatory powers of the features over the entire text collection rather than being class specific. DFS considers the following requirements: 1) a term that occurs frequently in a single class and not in other classes is distinctive, 2) a term that rarely occurs in a single class and not in other classes is irrelevant, 3) a term that occurs frequently in all classes is irrelevant, and 4) a term that occurs in some of the classes is relatively distinctive. DFS assigns scores to the features between 0.5 (least discriminative) and 1.0 (most discriminative). It can be formally calculated as [14]:

$$DFS(t) = \sum_{i=0}^{M} \frac{P(c_i|t)}{P(\bar{t}|c_i) + P(t|\bar{c_i}) + 1}$$

where M is the number of classes and $P(t|\bar{c_i})$ is the conditional probability of term t given the classes other than c_i.

C. What about cost?

According to our approach, some developers could be overloaded (i.e., the approach may assign a large number of bug reports to one developer while assigning a small number of reports to another developer). This scenario will increase the cost of the bug triaging process. A direct approach to solve this problem is to predict three developers (class labels) for each new bug report. The approach assigns a bug report to the second developer if the first developer is overloaded and so on. This can be done by modifying the Naive Bayes algorithm. In Naive Bayes, the class label of a bug report is predicted as class c_1 inwhich P(c_1 |report) is the highest P(c_i |report) where P(c_i |report) is the probability of class c_i given the bug report and $c_i \in C$. In our modified Naive Bayes, the three class labels of a bug report are predicted as classes c_1, c_2, and c_3 which have the highest P(c_i |report).

We also propose another approach based on clustering to reduce the cost of the bug triaging process. The approach is divided into the following steps:

- Represent bug reports using sub-profiles induced by the best term selection method.
- Cluster bug reports using K-means clustering algorithm.
- Determine the developers who fixed the bug reports in each cluster.
- Rank the developers on each cluster based on their experience.

- Redistribute the extra bug reports of each overloaded developer to non-overloaded developer as follows:
  - Assign each extra bug report to a cluster.
  - Assign the bug report to the top-ranked developer if not overloaded. Otherwise, it is assigned to the second top-ranked developer and so on.

In our work, a developer is considered overloaded if the number of assigned bugs for a developer is greater than the average number of assigned bugs for all developers. To cluster bug reports, we use one of the most commonly used clustering algorithms (K-means). K-means aims to partition data into K clusters in which each instance belongs to the cluster with the nearest mean. In K-means, the number of clusters (K) needs to be specified in advance. To specify the number of clusters (K), we use the following equation [15]:

$$K = \frac{m \times n}{t}$$

where m is the number of bug reports, n is the number of terms, and t is the number of non-zero entries in the bug-term matrix. The idea here is that if the number of nonzero entries is increased, the similarity among bug reports is increased which leads to smaller number of clusters.

IV. Experimental Evaluation

In this Section, we report the results of our proposed approach on real datasets. A description of the datasets that are used is first shown then an analysis of the results is provided.

A. Dataset

We evaluate our approach based on bug repositories of Eclipse[1], NetBeans[2], and Maemo[3]. In our work, we collect the bug reports that have the status of [Closed, Verified, and Resolved] and the resolution of [Fixed]. For each bug report, we extract the bug ID, the assignee, opened, changed, and the summary fields.

For Eclipse, we choose all bug reports for SWT component (7685). We also choose all bug reports for UI component from June 1^{st}, 2009 until October 27^{th}, 2012 (7688). For NetBeans, we choose (11974) bug reports from June 1^{st}, 2011 until October 27^{th}, 2012. For Maemo, we choose all bug reports (4505).

We want to refine the training set further to remove reports that are assigned to inactive developers (i.e., developers who no longer work on the project or developers who have only fixed a small number of bugs). To determine active developers, other work considered only developers who have fixed a minimum number of bug reports as a threshold. This approach is not accurate (i.e., software teams usually change overtime) which may lead to assigning a new reported bug to a developer who no longer available. Thus, we consider developers who have fixed at least 25 bug reports in the last year. Table I shows a summary of the refined datasets.

[1]https://bugs.eclipse.org/bugs/

[2]http://netbeans.org/bugzilla/

[3]https://bugs.maemo.org

TABLE I
A summary of Bug Reports.

Name	# of bug reports	# of terms	# of developers
Eclipse-SWT	7561	6560	21
Eclipse-UI	6791	6104	58
NetBeans	11311	9284	56
Maemo	3505	4659	33

B. Results and Discussion

We use the Naive Bayes classifier in our bug triaging approach and we recommend one developer for each new bug report to the triager. We apply each of the five term selection methods, LOR, X^2, TFRF, MI, and DFS on four datasets described in the previous section. Term selection methods are implemented using R language version 2.15.1. The WEKA tool is used for classification task[4]. The corpus for each dataset is created using the tm package[5]. Precision, recall, and F-score are used to evaluate the efficacy of applying different term selection methods on the classification task. Precision is the number of correct recommendations divided by the number of recommendations made. Recall is the number of correct recommendations divided by the number of possible relevant developers. F-score is the harmonic mean of precision and recall.

For term selection methods, we apply different percentages of the terms (1% to 10%) to investigate the effect of using different number of terms on the accuracy of classification. It is important to note that the number of selected terms will be less than or equal to the percentage of selected terms (i.e., we may have some selected terms for two developers that are common and these terms will be counted once). For the baseline approach, we consider all the terms (after processing) in the bug summary weighted by term frequency. The class label for a bug report is represented as the developer who have fixed that bug report.

For evaluation, the dataset is divided into training and testing sets. To obtain unbiased evaluation results, we perform a 10-fold cross-validation. Table II shows the precision and the recall for the datasets when different term selection methods are applied. P denotes precision while R denotes recall. For Eclipse-SWT, the precision and recall for the baseline approach are 0.290 and 0.270 respectively. The best precision and recall for Eclipse-SWT is obtained using X^2 when the percentage of terms is 2%. For Eclipse-UI, the precision and recall for the baseline approach are 0.014 and 0.014 respectively. The best precision for Eclipse-UI is obtained using X^2 when the percentage of terms is 1% while the best recall is also obtained using X^2 when the percentage of terms is 8%. For NetBeans, the precision and recall for the baseline approach are 0.014 and 0.015 respectively. The best precision for NetBeans is obtained using X^2 when the percentage of terms is 1% while the best recall is obtained using MI when the percentage of terms is 10%. For Maemo,

[4]http://www.cs.waikato.ac.nz/ml/weka/

[5]http://cran.r-project.org/web/packages/tm/index.html

TABLE II
PRECISION AND RECALL OF CLASSIFICATION USING DISCRMINATING TERMS ON FOUR DATASETS.

Method	Dataset	Percentage of selected terms									
		0.01	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**	**0.1**
		P-R	P-R	P-R	P-R	P-R	P-R	P-R	P-R	P-R	P-R
LOR	Eclipse-SWT	0.347-0.295	0.314-0.290	0.301-0.282	0.300-0.277	0.299-0.278	0.292-0.276	0.293-0.277	0.293-0.276	0.290-0.272	0.291-0.272
X2		0.380-0.300	**0.386-0.306**	0.372-0.306	0.372-0.304	0.362-0.303	0.362-0.303	0.343-0.301	0.344-0.301	0.344-0.301	0.336-0.298
TFRF		0.235-0.244	0.253-0.249	0.2680.256	0.276-0.262	0.276-0.261	0.283-0.265	0.284-0.266	0.287-0.266	0.286-0.266	0.287-0.267
MI		0.367-0.291	0.353-0.297	0.320-0.290	0.318-0.287	0.315-0.285	0.309-0.283	0.303-0.281	0.303-0.282	0.300-0.280	0.301-0.279
DFS		0.320-0.277	0.385-0.294	0.351-0.300	0.353-0.300	0.342-0.302	0.342-0.302	0.332-0.299	0.320-0.293	0.321-0.294	0.314-0.290
LOR	Eclipse-UI	0.445-0.278	0.437-0.310	0.395-0.323	0.376-0.331	0.375-0.337	0.366-0.340	0.366-0.344	0.36-0.345	0.359-0.344	0.357-0.343
X2		**0.509**-0.272	0.498-0.301	0.479-0.325	0.454-0.335	0.456-0.344	0.451-0.349	0.447-0.356	0.432-**0.358**	0.429-0.358	0.413-0.353
TFRF		0.197-0.191	0.226-0.215	0.249-0.233	0.258-0.245	0.272-0.262	0.278-0.268	0.286-0.273	0.289-0.277	0.296-0.283	0.301-0.289
MI		0.441-0.271	0.419-0.301	0.382-0.315	0.36-0.322	0.354-0.328	0.356-0.332	0.353-0.331	0.352-0.331	0.356-0.335	0.348-0.333
DFS		0.437-0.288	0.428-0.323	0.407-0.342	0.395-0.346	0.393-0.352	0.392-0.353	0.399-0.353	0.394-0.351	0.387-0.353	0.38-0.351
LOR	NetBeans	0.385-0.132	0.356-0.186	0.351-0.193	0.324-0.212	0.319-0.218	0.315-0.218	0.303-0.217	0.291-0.219	0.286-0.220	0.274-0.220
X2		**0.503**-0.128	0.488-0.181	0.471-0.191	0.456-0.202	0.419-0.208	0.412-0.210	0.387-0.212	0.377-0.212	0.377-0.213	0.373-0.215
TFRF		0.105-0.095	0.158-0.132	0.166-0.145	0.186-0.155	0.205-0.170	0.208-0.176	0.228-0.180	0.234-0.193	0.241-0.198	0.244-0.200
MI		0.426-0.134	0.372-0.188	0.35-0.194	0.333-0.204	0.321-0.209	0.316-0.212	0.312-0.219	0.305-0.223	0.3-0.222	0.301-**0.224**
DFS		0.343-0.170	0.352-0.184	0.344-0.189	0.377-0.196	0.326-0.207	0.320-0.211	0.317-0.211	0.306-0.213	0.301-0.214	0.303-0.215
LOR	Maemo	0.486-0.424	0.420-0.430	0.417-0.440	0.411-0.437	0.395-0.430	0.389-0.425	0.390-0.424	0.377-0.409	0.376-0.407	0.272-0.401
X2		**0.507**-0.447	0.473-0.468	0.419-0.468	0.433-0.466	0.427-0.465	0.421-0.462	0.422-0.463	0.420-0.463	0.419-0.458	0.418-0.453
TFRF		0.220-0.334	0.240-0.307	0.259-0.302	0.275-0.305	0.274-0.303	0.279-0.305	0.296-0.316	0.299-0.318	0.301-0.322	0.306-0.325
MI		0.452-0.418	0.425-0.423	0.381-0.417	0.383-0.420	0.376-0.415	0.373-0.413	0.374-0.413	0.368-0.406	0.366-0.404	0.360-0.399
DFS		0.467-0.464	0.417-0.468	0.407-0.470	0.411-**0.472**	0.411-0.472	0.402-0.467	0.423-0.461	0.420-0.459	0.420-0.460	0.420-0.460

the precision and recall for the baseline approach are 0.344 and 0.365 respectively. The best precision for Maemo is obtained using X^2 when the percentage of terms is 1% while the best recall is obtained using DFS when the percentage of terms is 4%. The results show that using term selection methods improve precision and recall over the baseline approach by up to 50% and 35% respectively.

Figure 3 shows the F-score of classification after applying the term selection methods. The x-axis represents the percentage of selected terms and the y-axis represents the F-score measure. For the Eclipse-SWT dataset, the best F-score for LOR (0.319) is achieved when the percentage of terms is 1% (the number of selected terms is 46). The best F-score for X^2 (0.341) is achieved when the percentage of terms is 2% (the number of selected terms is 111). The best F-score for TFRF (0.277) is achieved when the percentage of terms is 10% (the number of selected terms is 333). The best F-score for MI (0.325) is achieved when the percentage of terms is 1% (the number of selected terms is 41). The best F-score for DFS (0.333) is achieved when the percentage of terms is 2% (the number of selected terms is 131). It is clear that X^2 achieves the best F-score while the number of selected terms is small (46). The baseline F-score for Eclipse-SWT is 0.280 and the number of terms is 6560. Therefore, X^2 achieves 6.1% improvement over the baseline approach with only 111 selected terms. Moreover, the X^2 method outperforms other selection methods for all of the selected terms (0.01 to 0.1).

For the Eclipse-UI dataset, X^2 achieves the best F-score and TFRF achieves the lowest F-score. The baseline F-score for Eclipse-UI is 0.014 and the number of terms is 6104. The X^2 method achieves 38.2% improvement over the baseline approach with only 381 selected terms.

For the NetBeans dataset, X^2 achieves the best F-score when the percentage of selected terms is at least 2% and TFRF achieves the lowest F-score. The baseline F-score for Eclipse-UI is 0.014 and the number of terms is 9284. The X^2 method achieves 26.6% improvement over the baseline approach with only 312 selected terms.

For the Maemo dataset, X^2 achieves the best F-score when the percentage of selected terms is at least 1% and TFRF achieves the lowest F-score. The baseline F-score for Maemo is 0.354 and the number of terms is 4659. The X^2 method achieves 12.1% improvement over the baseline approach with only 33 selected terms.

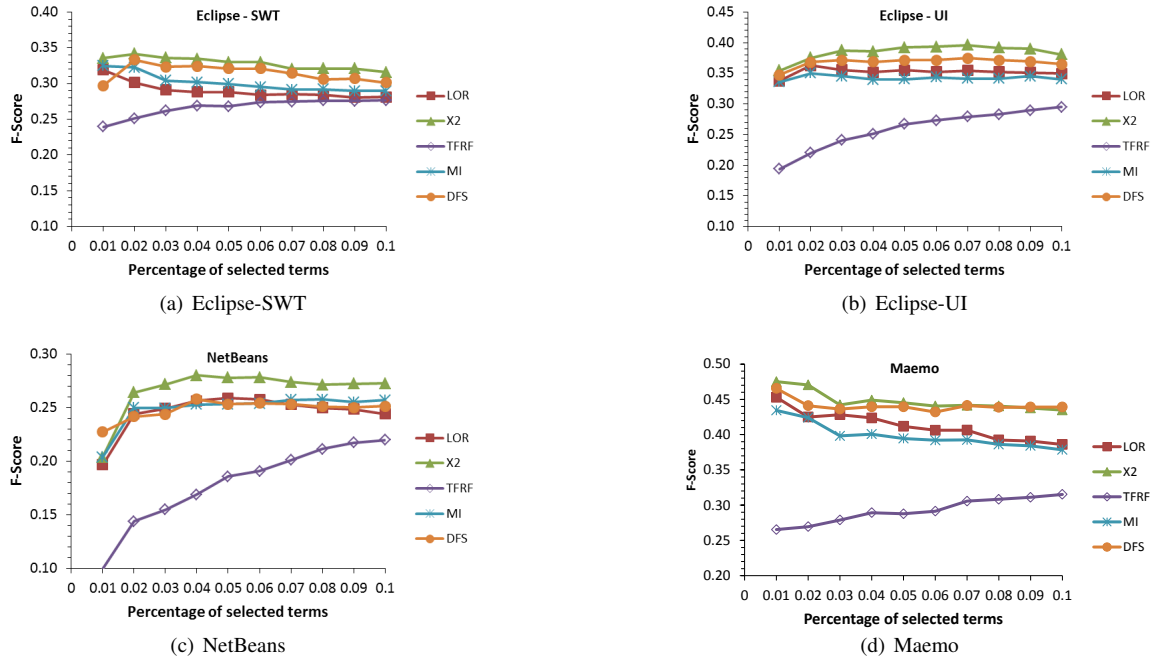
TABLE III
RESULTS OF K-MEANS CLUSTERING

Project	K	TW	BE	ACS	ADC
Eclipse-SWT	203	335	4356	37	3
Eclipse-UI	364	1640	7041	18	4
NetBeans	458	1423	7345	24	4
Maemo	416	67	2077	8	2

For re-balancing the developers' loads, we apply K-means clustering on the selected datasets. For K-means implementation, we use Hartigan's clustering method implemented in R language. The bug reports are represented using the terms selected by X^2. The results of clustering appear in Table III where K represents the number of clusters, TW represents the total within-cluster sum of squares, BE represents the between-cluster sum of squares, ACS denotes the average cluster size, and ADC denotes the average number of developers per cluster. It is clear from Table III that TW is small compared to BE which proves that bug reports in each cluster are similar while bug reports in different clusters are not.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented an approach to automatically assign bug reports to developers with the appropriate expertise. Our approach used term selection methods to choose the most discrimniating terms to describe bug reports. We then built a predictive model using the Naive Bayes classifier to predict a developer for each newly coming bug report. We also



(a) Eclipse-SWT (b) Eclipse-UI

(c) NetBeans (d) Maemo

Fig. 3. F-Score of classification on four datasets.

incorporated cost in order to redistribute bugs to re-balance the load between developers by using a clustering-based approach.

Our experimental results showed that the X^2 term selection method outperforms the other selected term selection methods in terms of the F-score for all datasets. Moreover, X^2 improved the F-score over the baseline approach by 6.2%, 38.2%, 26.5%, and 12.1% on Eclipse-SWT, Eclipse-UI, NetBeans, and Maemo respectively. The experimental results demonstrate that our proposed approach is very effective for the bug assignment problem.

In the future, we want to investigate the effect of using other term selection methods. Furthermore, we want to implement and validate our modified version of Naive Bayes. We are also planning to include other factors (such as number of comments) to rank developers in each cluster.

REFERENCES

[1] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 361–370.

[2] D. Čubranić and G. C. Murphy, "Automatic bug triage using text categorization," in *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering*. Citeseer, 2004, pp. 92–97.

[3] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," 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*. ACM, 2009, pp. 111–120.

[4] J. Anvik, "Automating bug report assignment," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 937–940.

[5] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 2009, pp. 131–140.

[6] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," in *Proc. Intl. Conf. Software Engineering & Knowledge Engineering (SEKE 10)*, 2010, pp. 209–214.

[7] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 308–318.

[8] J.-W. Park, M.-W. Lee, J. Kim, S. won Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting systems." in *AAAI*, W. Burgard and D. Roth, Eds. AAAI Press, 2011.

[9] W. Zou, Y. Hu, J. Xuan, and H. Jiang, "Towards training set reduction for bug triage," in *Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference*, ser. COMPSAC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 576–581.

[10] A. Tamrawi, T. Nguyen, J. Al-Kofahi, and T. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 365–375.

[11] D. Mladenic, "Machine learning on non-homogeneous, distributed text data." Ph.D. dissertation, University of Ljubljana, Faculty of Computer and Information Science, 1998.

[12] C. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 1.

[13] M. Lan, C. L. Tan, J. Su, and Y. Lu, "Supervised and traditional term weighting methods for automatic text categorization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 4, pp. 721–735, apr 2009.

[14] A. K. Uysal and S. Gunal, "A novel probabilistic feature selection method for text classification," *Knowledge-Based Systems*, vol. 36, no. 0, pp. 226 – 235, 2012.

[15] F. Can and E. Ozkarahan, "Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases," *ACM Transactions on Database Systems (TODS)*, vol. 15, no. 4, pp. 483–517, 1990.