2013 12th International Conference on Machine Learning and Applications

Bug Reports Prioritization: Which Features and Classifier to Use?

Mamdouh Alenezi
College of Computer & Information Sciences
Prince Sultan University
Riyadh 11586, Saudi Arabia
enezi.firstName@gmail.com

Shadi Banitaan
Department of Mathematics, Computer Science and Software Engineering
University of Detroit Mercy
Detroit, MI 48221, USA
banitash@udmercy.edu

***Abstract*—Large open source bug tracking systems receives large number of bug reports daily. Managing these huge numbers of incoming bug reports is a challenging task. Dealing with these reports manually consumes time and resources which leads to delaying the resolution of important bugs which are crucial and need to be identified and resolved earlier. Bug triaging is an important process in software maintenance. Some bugs are important and need to be fixed right away, whereas others are minor and their fixes could be postponed until resources are available. Most automatic bug assignment approaches do not take the priority of bug reports in their consideration. Assigning bug reports based on their priority may play an important role in enhancing the bug triaging process. In this paper, we present an approach to predict the priority of a reported bug using different machine learning algorithms namely Naive Bayes, Decision Trees, and Random Forest. We also investigate the effect of using two feature sets on the classification accuracy. We conduct experimental evaluation using open-source projects namely Eclipse and Firefox. The experimental evaluation shows that the proposed approach is feasible in predicting the priority of bug reports. It also shows that feature-set-2 outperforms feature-set-1. Moreover, both Random Forests and Decision Trees outperform Naive Bayes.**

***Keywords*—bug triaging, text classification, predictive model, bug priority**

I. Introduction

Most open source software projects contain a bug tracking system (BTS) to collect and manage bug reports. BTS allows users from different geographical areas to report their error findings in a unified environment. BTS helps developers to track and communicate about bug reports and development issues which results in fixing bug reports in reasonable time. These bug reports are usually used to guide several software maintenance activities in order to produce more reliable software systems. One of the important software maintenance activities is bug triaging. The triager examine the new filed bug report to determine if it is valid and assigns a potential developer to fix it.

When a bug tracking system receives a new filed bug report, the triager makes decisions about several characteristics of bug reports such as priority and severity levels. The bug priority level indicates the importance of that bug from business perspective. It gives an indication of the order in which bug reports should be fixed. Developers usually use the value of this feature to prioritize their work by fixing the highly important bugs first. The values of this field range from P1 to P5 where P1 represents the highest priority while P5 represents the lowest priority. Bug prioritization process usually performed manually which makes it error-prone and labor intensive. It relies heavily on the triager judgment and experience. Many bug reports have been assigned incorrect priority levels and many of them are usually left blank since bug prioritization needs a deep knowledge of bug reports. Wrong assignments of priority levels may lead to utilizing resources ineffectively (e.g., wasting time and effort by fixing unimportant bugs first). In this work, we present an approach to solve the aforementioned problems by automatically prioritize bug reports.

Machine learning techniques such as Naive Bayes and Support Vector Machines are used to build predictive models to categorize instances into different class labels based on historical data. These techniques have been previously used to automate the bug triaging process [1], [2], [3]. Even though, little work has been done to predict other characteristics such as severity and priority of bug reports. In this paper, we investigate whether we can accurately predict the priority of a reported bug by using several features such as the textual description of bug reports or other meta-data features such as the severity level.

The contributions of this paper include the following:

- We investigate the effectiveness of applying several machine learning techniques namely Naive Bayes, Decision Trees, and Random Forests on the classification performance.
- We evaluate the impacts of using different feature sets to build the predictive model. The first feature set is based on the textual contents of bug reports while the second feature set is based on meta data information of bug reports.
- We conduct experimental evaluation using two bug reports datasets namely Eclipse and Firefox obtained from open source projects.

The rest of the paper is organized as follows: Section II presents some background information about bug reports. Section III describes the proposed approach. The experimental evaluation and discussion are presented in Section IV. Section VI discusses some threats to validity. Section V discusses related work and Section VII concludes the paper.

978-0-7695-5144-9/13 $31.00 © 2013 IEEE
DOI 10.1109/ICMLA.2013.114

IEEE computer society

II. Background

We provide some necessary background information about bug reports in Section II-A. The life-cycle of a bug report is presented briefly in Section II-B.

A. Bug Report

Bug reports in Bugzilla consist of predefined fields, text description, attachments and dependencies. Predefined fields represent attributes of a bug. Some attributes are unchangeable such as creation date and the reporter who filed the bug. Other attributes maybe changed over bug lifetime such as product, component, priority and severity. Some attributes maybe frequently modified such as the assignee, the current state and the final resolution. The text description of a bug report refers to the natural language contents, including the title of the bug report and a full description of the bug. Figure 1 shows an example of a bug report.

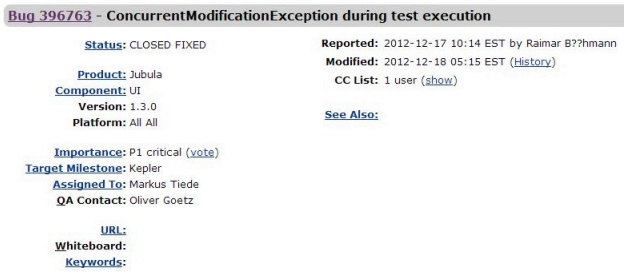

Fig. 1. An example of a bug report.

B. Bug Life-cycle

There are different states inwhich a bug report can experience in its life-cycle. Figure 2 depicts the life-cycle of bugs in Bugzilla-based projects. When a new bug report is filed, it is assigned a NEW state. Once it has been triaged and assigned to a developer, its state is then changed to ASSIGNED. After closing this bug, its state is set to RESOLVED, VERIFIED or CLOSED. The resolution to this bug is marked in several ways; the resolution status in the report is used to record how the report was resolved. If the resolution results in changing code base, this bug is marked as FIXED. When a bug is considered as a duplicate to other bugs, it is set to DUPLICATE. If a bug will not be fixed, or it is not an actual bug, it will be set to WONTFIX or INVALID respectively. If a bug was resolved but has been reopened, it is marked as REOPENED [4].

III. The Approach

In this Section, we present an approach for predicting the priority of each newly coming bug report using bug reports history obtained from BTS. We formulate the problem as a classification task. Three class labels are used to categorize bug reports namely *High*, *Medium*, and *Low*. *High* represents both P1 and P2, *Medium* represents P3, and *Low* represents P4 and P5. This representation aims at helping developers on fixing high priority bug reports first. We start by describing the feature sets under investigation. Then, we present the machine learning algorithms used in this study. Finally, we present the evaluation metrics which are used to evaluate the approach.

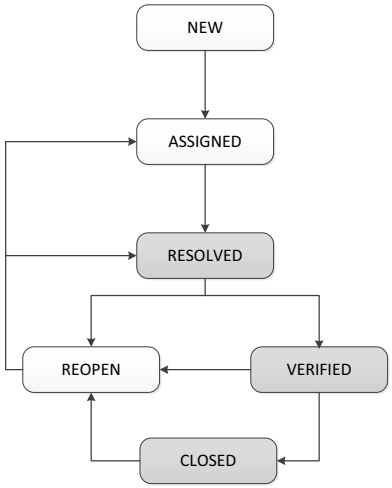

Fig. 2. Bug Report Life-cycle.

A. Feature-Set-1

In this set, we only use the textual description of bug reports. The textual description of a bug report is available in two fields namely summary (title) and description. We consider the summary only as the textual data since the description of bug reports holds many terms that are unrelated to the functionality of bug reports [5].

The summary of bug reports is unstructured data that needs a pre-processing step in order to convert it into structured data. Therefore, we apply the traditional text processing approach to transform the text data into a meaningful representation as follows:

- Tokenization and filtering: the first step is to split the summary of each bug report into tokens (terms). Then, filter out unnecessary terms which include stop-words, punctuations, white-spaces and numbers.
- Vector space representation: each bug report is represented as a vector where each word in the bug report represents a feature. We use the Term Frequency (TF) of the word to get the value of each word feature.

B. Feature-Set-2

We would like to investigate the usage of other features than the textual content of bug reports on the effectiveness of classification accuracy. In this set, we use the following meta-data features:

- Component: it represents the component in which the bug belongs to.
- Operating system: it represents the operating system the bug was observed on.
- Severity: it represents the degree of severity of the bug.

We use these fields as features because they contain useful information that may help in discriminating between priority

levels. Other available fields in BTS such as Bug id and Changed (when the bug was last updated) do not contain useful information. Moreover, some fields like Keywords can not be used because it is optional (i.e., most bug reports have empty values for these fields).

C. Machine Learning Techniques

In this section, we briefly present the machine learning techniques used in this work.

1) Naive Bayes Classifier: Naive Bayes is a probabilistic classifier which assumes that all features are independent. It finds the class with maximum probability given a set of features values using the Bayes theorem.

2) Decision Trees Classifier: Decision Tree is a classifier in the form of a tree structure. It is a predictive model that decides the dependent value of a new sample based on diverse attribute values of the existing data. Each internal node in the tree represents a single attribute while leaf nodes represent class labels. Decision trees classify each instances by starting at the root of the tree and moving through it until a leaf node.

3) Random Forests Classifier: Random Forests is an ensemble learning method that generates several decision trees at training time. Each tree gives a class label. The Random Forests classifier selects the class label that has the mode of the classes output by individual trees.

D. Evaluation Metrics

We evaluate the approach using widely used metrics in classification and information retrieval namely Precision, Recall, and F-measure as follows:

$$\text{Precision} = \frac{\text{Number of correctly classified}}{\text{Number of classifications made}}$$

$$\text{Recall} = \frac{\text{Number of correctly classified}}{\text{Number of possible relevant classifications}}$$

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

IV. Experimental Evaluation

In this Section, details of the datasets used in our experimental evaluation are shown in Section IV-A. The results and discussion are presented in Section IV-B.

A. Datasets

We choose bug reports from two different projects namely Eclipse and Firefox. We choose only bug reports that are marked as RESOLVED, CLOSED or VERIFIED because the priorities of such reports have been confirmed. We extract bug reports dated between January 1^{st}, 2010 and December 31^{st}, 2012. Table I shows a summary of the datasets.

As mentioned before, there are three class labels namely *High*, *Medium*, and *Low*. Table II shows the distribution of these priority levels in bug reports. It is clear from Table II that the data is imbalanced (e.g, 67742 bug reports are labeled as *Medium* while 553 bug reports are labeled as *Low* in Eclipse).

TABLE I. Summary of the Datasets

Project	# of Bugs	From	To
Eclipse	74183	Jan 01, 2010	Dec 31, 2012
Firefox	7284	Jan 01, 2010	Dec 31, 2012

TABLE II. Priority Distribution

Project	High	Medium	Low
Eclipse	5888	67742	553
Firefox	1142	6009	133

1) Imbalanced Data: A training dataset is considered imbalanced if one or more of the class labels are represented by significantly less number of instances compared to other class labels. This problem leads to skewed data distribution between classes which is known to hinder the learning performance of classification. Weiss and Provost [6] indicated in their study that balancing the dataset usually achieves better classification results. On the same hand, it is generally more important to correctly classify the smaller class instances. Therefore, we re-balanced the distribution of the class labels by randomly selecting equally representative instances of each class label.

B. Results and Discussion

Since all class labels are important, we present the classification results for each class label to investigate whether we can predict each one of them accurately using the two feature sets. Table III shows number of features in each Feature set. It is clear from Table III that feature-set-2 has significantly less dimensions.

TABLE III. Number of Features

Project	Feature-set-1	Feature-set-2
Eclipse	3259	287
Firefox	1097	45

Figure 3 shows the classification results for Eclipse and Firefox. For Eclipse, feature-set-2 outperforms feature-set-1 dramatically for all priority levels using the three classifiers in terms of Precision, Recall, and F-measure.For instance, the F-measure of the *Low* class label is 0.639 and 0.282 for feature-set-2 and feature-set-1 respectively using Decision Tree (i.e., feature-set-2 improves the F-measure by 35.7% over feature-set-1). The average F-measure of feature-set-1 is 0.421, 0.364, and 0.434 for Naive Bayes, Decision Trees, and Random Forest respectively. For feature-set-2, the average F-measure is 0.593, 0.603, and 0.611 for Naive Bayes, Decision Trees, and Random Forest respectively. We can conclude that Random Forest outperforms both Naive Bayes and Decision Trees in both feature sets.

For Firefox, feature-set-2 outperforms feature-set-1 significantly in terms of Precision, Recall, and F-measure. For instance, the F-measure of the *High* class label is 0.476 and 0.298 for feature-set-2 and feature-set-1 respectively using Random Forest (i.e., feature-set-2 improves the F-measure by 17.8% over feature-set-1). The average F-measure of feature-set-1 is 0.340, 0.355, and 0.356 for Naive Bayes, Decision Trees, and Random Forest respectively. For feature-set-2, the average F-measure is 0.460, 0.491, and 0.476 for Naive Bayes, Decision Trees, and Random Forest respectively. We can conclude that both Random Forest and Decision Trees outperform Naive Bayes in both feature sets. To sum up,

	Class Label	Feature set 1			Feature set 2		
		Precision	Recall	F-measure	Precision	Recall	F-measure
Naive Bayes	High	0.413	0.391	0.402	0.653	0.501	0.567
	Medium	0.394	0.467	0.427	0.507	0.685	0.583
	Low	0.455	0.396	0.423	0.645	0.57	0.605
	avg	0.421	0.418	0.419	0.602	0.585	0.593
Decision Tree	High	0.389	0.199	0.263	0.659	0.535	0.591
	Medium	0.337	0.627	0.438	0.519	0.629	0.569
	Low	0.366	0.23	0.282	0.647	0.631	0.639
	avg	0.364	0.352	0.358	0.608	0.599	0.603
Random Forest	High	0.408	0.519	0.457	0.648	0.624	0.636
	Medium	0.408	0.382	0.395	0.564	0.566	0.565
	Low	0.485	0.385	0.429	0.622	0.644	0.633
	avg	0.434	0.429	0.431	0.612	0.611	0.611

(a) Eclipse

	Class Label	Feature set 1			Feature set 2		
		Precision	Recall	F-measure	Precision	Recall	F-measure
Naive Bayes	High	0.353	0.406	0.378	0.465	0.444	0.454
	Medium	0.303	0.271	0.286	0.366	0.256	0.301
	Low	0.362	0.346	0.354	0.525	0.707	0.603
	avg	0.339	0.341	0.340	0.452	0.469	0.460
Decision Tree	High	0.374	0.368	0.371	0.544	0.466	0.502
	Medium	0.291	0.188	0.228	0.439	0.218	0.291
	Low	0.385	0.526	0.445	0.484	0.797	0.602
	avg	0.35	0.361	0.355	0.489	0.494	0.491
Random Forest	High	0.343	0.263	0.298	0.574	0.406	0.476
	Medium	0.282	0.233	0.255	0.369	0.286	0.322
	Low	0.422	0.594	0.493	0.485	0.737	0.585
	avg	0.349	0.363	0.356	0.476	0.476	0.476

(b) Firefox

Fig. 3. Classification results of Eclipse and Firefox.

feature-set-2 outperforms feature-set-1 dramatically for both datasets.

Wilcoxon test is a non-parametric statistical hypothesis test [7]. We use Wilcoxon test to compare the F-measure values of each feature-sets to find out if feature-set-2 is significantly better than feature-set-1. The p-value of our test is < 0.001 which means that the classification results of feature-set-2 is significantly better than feature-set-1. Besides the low results of feature-set-1 compared to feature-set-2, the vocabulary that are used to describe bugs may change over time [8] (e.g., reporters and developers change overtime). Therefore, we are recommending to use feature-set-2 since it does not depend on the textual description of bugs, it gives better classification results, and it contains much smaller number of features (See Table III). Regarding the classification techniques, we are recommending to use Random Forest or Decision Trees since they both achieve comparable results and outperform Naive Bayes.

V. Related Work

Text mining has been successfully used to predict different meta-data about bug reports such as severity and security. Gegick et al. [9] proposed an approach to predict whether a bug is security related or not using a classification technique. They have applied their approach on dataset obtained from a Cisco software project. Their model correctly classified 78% of the security bug reports as validated by security engineers. Lamkanfi et al. [10] investigated the effectiveness of using several classification algorithms in predicting the severity level of each bug report. They applied their approach on Eclipse, Mozilla, and GNOME projects. Their results are varied between 0.65 and 0.85. Wang at el., [11] presented a text classification model using three classifiers to predict the component label of a new bug report. They used TF-IDF and Chi-square to construct the vector space. Their experiments on Eclipse showed that the accuracy of SVM classifier reached up to 81.21%.

Yu et al. [12] proposed an approach to predict the priority of defects using Neural Network. They used several features extracted from the software testing process such as milestone, workflow, and module. They defined four levels of priority and evaluated their approach on five international health care products. Their experimental results showed that their approach is feasible and effective.

Different machine learning techniques have been applied on open bug repository data to identify duplicate bug reports [13], assign the most experienced developer to resolve a new bug automatically [14], [3], estimate the required effort to fix bug reports [15] and predict the files that have most post-release defects [16]. In this work, we apply three machine learning techniques on different feature sets.

VI. Threats to Validity

In this section, we discuss the threats that affect the validity of our proposed approach. First, we only select two open source projects. Other projects may give different conclusions. Therefore, we should apply the approach on more projects in order to generalize the results. Second, we only consider projects that use Bugzilla as their bug tracking system. Other bug tracking systems model bug reports in a different way such as Gnats. Therefore, the proposed approach should be applied to other bug tracking systems other than Bugzilla.

VII. Conclusion

In this paper, we presented an approach to prioritize bug reports using classification. Three classification techniques were employed namely Naive Bayes, Decision Trees, and Random Forests. In addition, two feature sets were investigated. The first feature set is based on the textual description of bug reports. The second feature set is based on pre-defined meta-data of bug reports. The experimental results showed that both Random Forest and Decision Tree gave better classification results than Naive Bayes. Moreover, the results showed that feature-set-2 improved the results significantly compared to feature-set-1 in terms of Precision, Recall, and F-measure. Future directions include applying the proposed approach on other projects including close-source projects. We are also

planning to investigate the effect of using different combinations of features on the classification accuracy.

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] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 3, p. 10, 2011.

[3] S. Banitaan and M. Alenezi, "Tram: An approach for assigning bug reports using their metadata," in *Communications and Information Technology (ICCIT), 2013 Third International Conference on*. IEEE, 2013, pp. 215–219.

[4] 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.

[5] A. J. Ko, B. A. Myers, and D. H. Chau, "A linguistic analysis of how people describe software problems," in *Proceedings of the Visual Languages and Human-Centric Computing*, ser. VLHCC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 127–134.

[6] G. M. Weiss and F. J. Provost, "Learning when training data are costly: The effect of class distribution on tree induction," *J. Artif. Intell. Res. (JAIR)*, vol. 19, pp. 315–354, 2003.

[7] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[8] G. Chrupala, "Learning from evolving data streams: online triage of bug reports," in *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, ser. EACL '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 613–622.

[9] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 11–20.

[10] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. IEEE, 2011, pp. 249–258.

[11] D. Wang, H. Zhang, R. Liu, M. Lin, and W. Wu, "Predicting bugs components via mining bug reports," *Journal of Software*, vol. 7, no. 5, pp. 1149–1154, 2012.

[12] L. Yu, W.-T. Tsai, W. Zhao, and F. Wu, "Predicting defect priority based on neural networks," in *Proceedings of the 6th international conference on Advanced data mining and applications - Volume Part II*, ser. ADMA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 356–367.

[13] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 461–470.

[14] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *Journal of Software*, vol. 8, no. 9, pp. 2185–2190, 2013.

[15] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "Predicting effort to fix software bugs," *Softwaretechnik-Trends*, vol. 27, no. 2, 2007.

[16] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, ser. PROMISE '07. Washington, DC, USA: IEEE Computer Society, 2007.