Predicting the Severity of Bug Reports using Classification Algorithms

Pushpalatha M N and Mrunalini M
Dept. of ISE and MCA
M.S.Ramaiah Institute of Technology
Bangalore, India
pushpalathamn1@gmail.com and mrunlini.ram@gmail.com

***Abstract*—Bug triaging is the process of prioritizing the bug reports based on the severity and is driven by the business needs and available resources. Majority times, only few of the reported bugs are selected to be fixed. The selected bugs are prioritized (ordered) based on their severity (e.g. the bug inhibits an important feature of the product, the bug affects a large number of users) and then fixed according to their priority. If the severity is assigned incorrectly then time and resources may be wasted to fix that bug. Hence there is a need for new techniques to avoid this misuse of resources. In this paper, bagging ensemble method is used for predicting the severity of bug reports. Also, bagging ensemble method is compared with C4.5 classifier. The results have shown that bagging ensemble method gives better accuracy compared to C4.5 general classifier on the given dataset.**

Keywords— Bug triage, Bugzilla, Bug Severity

I. INTRODUCTION

According to medical term triage means is the process of prioritizing patients based on the severity of their condition so that it helps for the high critical patient to be treated first and later giving preference to less critical patient. In a similar way, the bug triage is the process of prioritizing the bug reports based on the severity of the report. Predicting the severity of bug report and predicting the developer to fix the bug report is being automated. These are part of bug triage.

Manually assigning the severity sometimes may be incorrect, because there may be inexperienced users in open source software. During commercial software development, the one who tests the software will assign the severity, i.e., the test engineer is responsible for assigning the severity. If test engineer is new and is inexperienced in the process of assigning the severity, then he can make use of the recommendation system for assigning the severity. In a similar way, if the test engineer is busy with some other works then he can make use of the system for assigning the severity to the new bug reports [3]; the manual process takes lot of time.

In the works [11-15], the authors used the general classifier algorithms such as J48, decision trees, naïve bayes, SVM etc, for automatic assignment of bug report to developer. In [16], the authors have discussed the technique for automatic assignment of bug report to development team using stacked generalization ensemble method for industrial data. The bagging ensemble method is used for assigning the bug report to developer in [17]. Also in [17], ensemble method is compared with general classifiers and the results show that the ensemble method gives better accuracy over general classifier method. In a similar way [1, 2, 3, 4, 7, 8, 9, and 10] used the general classifier for predicting the severity. The literature shows that the ensemble methods are not addressed in available works.

In this paper, we presented the bagging ensemble classification method for predicting the severity of bug report. This is compared with the C4.5 classifier. The result shows that Bagging gives better accuracy over C4.5 classifier on the given dataset. For better prediction of severity of bug report bagging ensemble methods can be used.

The remaining part of the paper is organized as follows. In Section 2 related work is discussed; in Section 3 the methodology is discussed; Results and discussion is presented in Section 4. Conclusion and Future work is discussed in Section 5.

II. RELATED WORK

In [1], authors have used Mozilla, Eclipse and GNOME open source projects bug reports stored in Bugzilla is used for predicting the severity of bug reports as severe and non-severe using Naïve Bayes classification. They achieved the precision and recall in range of 0.65-0.75 for Mozilla and Eclipse and for GNOME in the range of 0.70-0.85. In the follow up work in [2] compared different classification algorithms for the performance. Authors compared Naive Bayes, Naive Bayes Multinomial, K-Nearest Neighbor, and Support Vector Machines on Eclipse and GNOME Bugzilla Bug repositories and concluded that Naive Bayes Multinomial gives best accuracy and fastest and works with less training data. In [3] authors have done case study in SEVERIS with data from NASA's PITS i.e., Project and Issue Tracking System is used for assigning severity levels to defect reports using RIPPER algorithm. In [3] for closed source project of NASA used different scale or level for assigning the severity. i.e., five-point scale is used which ranges one to five, worst to dullest. For robotic and human-rated missions different scale or level is used [3]. Nearest neighbour classification is used in [4] for predicting the severity using Eclipse, OpenSource and Mozilla open source Repositories. In [9] authors used Naïve Bayes, k-Nearest Neighbor, Naïve Bayes Multinomial, Support Vector

Machine, J48and RIPPER for determining severity of bug reports of NASA taken from the PROMISE repository.

Automatic assignment of bug report to suitable developer is done using different machine learning algorithms such as SVM, naïve bayes and C4.5 for Eclipse and Firefox open source projects [12]. In [11] authors used the naïve bayes algorithm for Eclipse bug reports. Naïve Bayes, RBF network, Random Forest, SVM and J48 algorithms are used in [13]. In [14] authors done the term selection using different term selection algorithm, then naïve bayes is used for bug triage of Eclipse open source bug reports. Different machine learning algorithm such as SVM, Expectation maximization, rules, Naive Bayes, C4.5 and nearest neighbor for recommending the developer for open source bug reports [15].

In [16] Stacking ensemble method is used for predicting the development team for industrial data and compared with Bayesian Networks, Naïve bayes, REPTree, SVM, Decision tree, rules and Random Forest machine learning algorithm. Bagging ensemble method is used in [17] for automatic prediction of developer to given bug report and compared with Naïve bayes algorithm.

III. METHODOLOGY

A. *Background*

Table I shows the sample bug report format for WineHQ. It contains Bug ID, reported emailID, summary, Severity, Product name, Component name and description, versions, platform, OS, browser, url etc

Bugs are usually logged by the development team and also by testers. The 'NEW' status is set to new bug report when it is submitted to the Bugzilla. The status will be 'ASSIGNED', when it is assigned to the developer. When the developer fixes the bug its status changes to 'RESOLVED'.

The status will be set 'VERIFIED' once it verified. Later the status is set 'CLOSED' for closing the bug. The resolution will be set to 'FIXED' if it is fixed successfully. If is a duplicate then resolution is set to 'DUPLICATE'. The bug report can be reopened when the status is 'RESOLVED' or 'VERIFIED' or 'CLOSED' for many reasons that time its status is set to 'REOPENED' and assigned to the developers to fix once again.

TABLE I: SAMPLE BUG REPORT

Bug ID	15
Reporter	emailed
Summary	A horizontal scrollbar is displayed
Severity	[blocker | critical | major | normal | minor | trivial | enhancement]
Product	wineHQ
Component	advapi32|advpack|atl
Component	Advanced API services library for security and registry calls. (
Description	dlls/advapi32/)
Version	0.9|0.9.1|0.9.2|0.9.3|0.9.4
Platform	[PC | Macintosh]
OS	[Windows | Mac | Linux]
Browser	[IE 6.0 | Firefox 1.5]
URL	http://www.mobilefish.com/tutorials/webdevelopment/webdevelopment_quickguide_bugtracking.html

TABLE II: THE SEVERITY FIELD DESCRIBES THE IMPACT OF A BUG

Block	Blocks development and/or testing work
Critical	Critical problem that prevents all applications from working
Major	Major loss of functionality for a wide range of applications
Minor	For minor loss of functionality, or other problem where an easy workaround is present
Trivial	For a UI glitch that doesn't affect running of a program
Enhancement	Request for enhancement

B. *Discussion about Bagging and C4.5 Classification Algorithm*

Bagging is also known as Bootstrap aggregating used for both classification and regression. Leo Breiman in 1994 proposed the Bagging technique. Bagging creates the composite model N* which reduces the variance of individual classifier by combining a series of N learned models, N_1, N_2…N_n.

Given a training set D of size n, the Bagging method works as follows:

T new training sets D_i is generated by sampling from D uniformly and with replacement, each of size s for iteration i (i=1,2,…..n). Some of the original tuples of D may not be included in D_i, whereas others may occur more than once, because sampling with replacement is used.

A classifier model, N_i, is learned for each training set, D_k. To classify an unknown tuple Y, each classifier N_i, returns its class prediction which counts as one vote. The bagged classifier N* counts the votes and assigns the class with the most votes to X [5].

C4.5 algorithm is called as J48 on the WEKA data mining tool. It is implemented using Java. C4.5 was developed by Ross Quinlan is an extension of ID3

algorithm. C4.5 is also called as statistical classifier because it generates decision tree which is used for classification task. The training data is a set $Z=z_1,z_2,...$ of already classified samples. Where each sample $z_i=a_1,a_{2,...}$ is a vector where a_1 is an attribute or feature of the sample. The training data is added with a vector $C = c_1, c_2...$ where $c_1, c_2,\ldots$ represent the class to which each sample belongs. C4.5 is used to construct the decision tree using set of attributes. At each node the attribute with the highest information gain is used for splitting the data sets into subsets make better in one class or the other. Then testing data is tested with the decision tree one attribute at time from the root to leaf with branches act as attribute value. Using C4.5 the decision tree using the concept of information entropy is built.

IV. RESULTS AND DISCUSSION

The bug data is collected from Bugzilla Bug repositories [19]. It includes different product and components from these repositories. Open source projects of Bugzilla repository have 7 severity levels as blocker, critical, major, minor, normal, and trivial and enhancement [10]. Each severity is explained in detail in Table II.The severity level enhancement is used for asking request feature in the form of a report using reporting mechanism and they do not represent real bug reports [1]. The default severity label is normal if reporter do not assign any severity level. In [1] the authors have considered trivial and minor as non-severe and major, critical and blocker as severe. In our study, we considered major, minor, critical and trivial four severity level.

Total of 690 bug reports are collected from the core component and WINE software. Core component is the shared component used by Firefox and other Mozilla software. We considered only BugId, Product, Component, Summary and Importance of bug reports for the classification purpose. In this, summary or description field is used for predicting the severity class. Classification is done using the WEKA a machine learning tool in the present study. Pre-processed the dataset using different pre-processing techniques in WEKA known by filters. After preprocessing supervised classification algorithm such as J48 and Bagging used for classification considering the severity field as the class label. 10-fold cross validation is used for evaluating the accuracy of classifiers. RandomForest classifier is used for creating the series of models.

Severity Class is represented as C. It belongs to either major or minor or critical or trivial.

$$\text{Precision}= \frac{Number\ of\ bug\ reports\ correctly\ predicted\ as\ C}{Number\ of\ bug\ report\ predicted\ as\ C} \quad (1)$$

$$\text{Recall}= \frac{Number\ of\ bug\ reports\ correctly\ predicted\ as\ C}{Number\ of\ bug\ reports\ of\ class\ C} \quad (2)$$

TABLE III: PRECISION AND RECALL FOR EACH CLASS USING DIFFERENT CLASSIFICATION ALGORITHM

	Precision	Recall	Class(C)	No
J48	0.48	0.55	major	1
	0.81	0.87	minor	2
	0.91	0.81	critical	3
	0.92	0.75	trivial	4
Bagging	0.62	0.49	major	1
	0.79	0.91	minor	2
	0.84	0.80	critical	3
	0.92	0.96	trivial	4

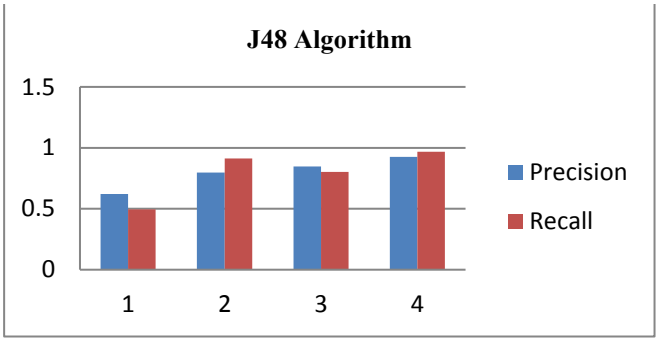

Figure 1: Precision and Recall using J48 Algorithm

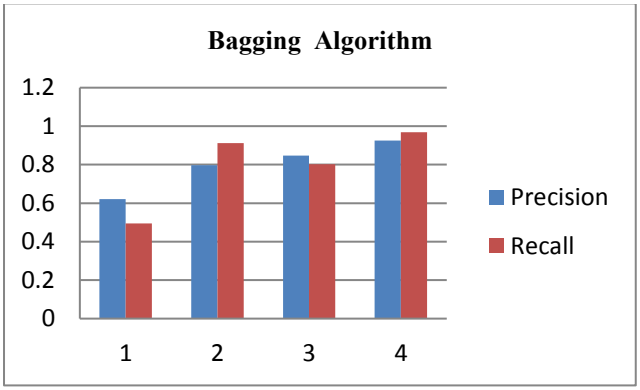

Figure 2: Precision and Recall using Bagging Algorithm

TABLE IV: ACCURACY OF DIFFERENT CLASSIFICATION ALGORITHM

Classification	J48	Bagging
Accuracy	79.82	81.27

$$\text{Accuracy}= \frac{predicted\ correctly}{total\ number\ of\ prediction} \quad (3)$$

Accuracy tells about the percent of prediction which were correct. Precision is also known as positive prediction value and tells about the percent of correctly predicted among all the predicted. Recall is also known as True positive rate or sensitivity. Recall tells about percent of correctly predicted over all the positive cases. To Predict the

severity of new bug reports by using different classification algorithm using reduced terms in order to save time and valuable resource. The given method is advantageous in correctly assigning the bug severity. It helps in utilizing the resources efficiently.

From Table IV and Figure 3 it can be observed that bagging gives better accuracy over J48 classification algorithm and Precision varies from 0.62 to 0.92 and recall varies from 0.494-0.968 using bagging ensemble method. Compared to J48 general classifier bagging gives slightly good precision and recall for some classes.

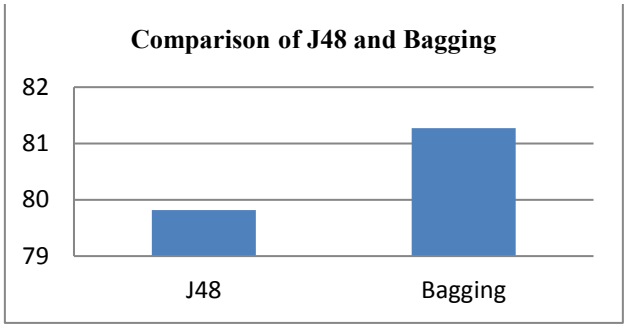

Figure 3: Comparison of J48 and Bagging algorithm

V. CONCLUSION AND FUTURE WORK

In this paper, the bagging ensemble is compared with C4.5 classification algorithm for predicting the severity of bug report of open source software collected from the bugzilla. Bagging gives better accuracy over C4.5 general classifier on the given dataset. Bagging also gives slightly better precision and recall over C4.5 classification algorithms. For better prediction of the severity of bug report, bagging ensemble method can be used, which is more robust to the effects of noisy data and over fitting. Which helps for prioritize the bug report for better fixing. Further, this method may be extended to predict the severity of bug reports for closed source software and cross-component context. Future work will be done for considering large number of bug reports.

REFERENCES

[1] Lamkanfi, A; Demeyer, S; Giger, E; Goethals, B (2010). Predicting the severity of a reported bug. In: 7th Working Conference on Mining Software Repositories, Cape Town, South Africa, 02 May 2010 - 03 May 2010,pp.1-10.

[2] Yuan Tian1, David Lo1, and Chengnian Sun2, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction", in 19th Working Conference on Reverse Engineering, 2012,pp.215-224

[3] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in IEEE International Conference on Software Maintenance, 2008, pp. 346–355,

[4] Ahmed Lamkanfi_, Serge Demeyer_, Quinten David Soetens_, Tim Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug", in 15th European Conference on Software Maintenance and Reengineering, 2011:249-258

[5] Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, 3rd ed, Morgan Kaufmann.

[6] Elmasri and Navathe, Fundamental of Database Systems, 4th ed, Pearson.

[7] K. K. Chaturvedi and V. B. Singh ,"Determining bug severity using machine learning techniques", in Int. Conf. Software Engineering (CONSEG), 2012, pp. 378–387.

[8] Meera Sharma, Madhu Kumari, R.K. Singh, and V.B. Singh, "Multi attribute based machine learning models for severity prediction in cross project context", in CCSA 2014, Part V, LNCS 8583, pp. 227–241.

[9] Chaturvedi, K.K., Singh, V.B, "Determining bug severity using machine learning techniques", in CSI-IEEE Int. Conf. Software Engineering (CONSEG), 2012, pp. 378–387.

[10] Chaturvedi, K.K., Singh, V.B, "An empirical comparison of machine learning techniques in predicting the bug severity of open and close source projects", Int. Journal of Open Source Software and Processes 4(2), 2013, pp. 32–59.

[11] D. Cubrani c and G. C. Murphy. "Automatic bug triage using text classification", Software Engineering and Knowledge Engineering, 2004, pages 92–97.

[12] John Anvik, Lyndon Hiew and Gail C. Murphy, "Who should fix this bug?", in 28th International Conference on Software Engineering, ACM, 2006, pp. 361–370.

[13] Ahsan S, Ferzund J, Wotawa F, "Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine", in 4h International Conference on Software Engineering Advances, 2009, pp 216_221.

[14] Alenezi M, Magel K, Banitaan S "Efficient bug triaging using text mining", Journal of Software 2013, pp 2185-2190.

[15] Anvik J, Murphy GC, "Reducing the effort of bug report triage: recommenders for development-oriented decisions", ACM Transactions on Software Engineering and Methodology, vol 20 issue 3, August 2011.

[16] Leif Jonsson, Markus Borg, David, Broman, Kristian Sandahl, Sigrid Eldh ,Per Runeson, "Automated bug assignment: ensemble-based machine learning in large scale industrial contexts", Empirical Software Engineering, vol 21, issue 4, pp 1533-1578, August 2016.

[17] Pushpalatha M N, Mrunalini M, "Automatic bug assignment using bagging ensemble method", International Journal of Advanced Information Science and Technology, vol.40, issue.40, pp 98-103, August 2015.

[18]Weka,http://www.cs.waikato.ac.nz/ml/weka/downloading.html

[19] Bugzilla, "https://bugzilla.mozilla.org/