IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 1, Ver. III (Jan – Feb. 2016), PP 27-35
www.iosrjournals.org

Predicting software aging related bugs from imbalanced datasets by using data mining techniques

Amir Ahmad
Faculty of Computing and Information Technology, King Abdulaziz University, Rabigh, Saudi Arabia

Abstract: *Software aging bugs are related with the life-span of the software. Rebooting is one of the solutions of this problem, however, it is time consuming and causes resources loss. It is difficult to detect these bugs during the time-limited software testing process. Data mining techniques can be useful to predict whether a piece of software has aging related bugs or not. The available datasets of software aging bugs present a challenge as they are imbalanced datasets. In these datasets, the number of data points with bugs is very small as compared to the number of data points with no bugs. It is important to predict the rare class (Bugs). In this paper we carried out experiment with a dataset containing data points related to aging-related bugs found in an open-source project MySQL DBMS. Data mining techniques developed for imbalanced datasets were compared with general data mining techniques. Various performance measures were used for the comparative study. The results suggest that data mining techniques developed for imbalanced datasets are more useful for correct prediction of data points related to aging related bugs. Data mining techniques developed for imbalanced datasets performed better than general data mining techniques on G-mean measure which is an important performance measure for imbalanced datasets.*

Keywords: *Aging-related Bugs, Data mining, Imbalanced datasets, Performance measures.*

I. Introduction

Software engineering deals with design, development and maintenance of software [1, 2]. Software bugs are defects which occur due to error in software design or software development process [3]. Due to these bugs, software produces incorrect results or does not produce expected results. As software plays a very important part now a days, software bugs can lead to financial losses and effort losses. For example, in 1996, the $1.0 billion rocket called Ariane 5 was destroyed because of a software bug [4]. Software bugs prevention is the first step to quality assurance of software. Tasks like training, requirements and code reviews, and other activities that promote quality software come under this step. Software testing is an important part of software development process [1, 2]. It finds out software bugs in a given piece of software [1, 2]. In software testing processes, software is tested by using given test cases. A large number of software bugs are generally detected during software testing. The software design and software code are corrected to remove the bugs. However, software testing is a time consuming process and all the possible test cases cannot be tested. In other words, some bugs still may go undetected. Some bugs are detected by customers; the cost of removing these bugs is higher. It is commonly believed that the earlier a defect is found, the cheaper it is to fix it. Efforts are being made to make software testing faster, cheaper, and more reliable. Test automation is an important step in which instead of human tester dedicated software is used for testing [5].

Software aging is related with the life-span of the software. As the software gets older it shows degraded performance, an increased failure occurrence rate and system hanging or crashing [6, 7]. As the runtime period of the software increases, its failure rate also increases. Aging effects in a system can only be detected while the system is running. It occurs because of software faults, called Aging-Related Bugs (ARBs) [6, 7]. However, due to the nature of this error (errors occur over time), it is difficult to correct the software faults for good during the time-limited testing process. Rebooting is a short term solution of this problem. The aging effect cannot be reversed without external intervention. ARBs can produce resource leakage, unterminated threads and processes, unreleased files and locks, numerical errors, and disk fragmentation [6, 7].

Data mining techniques analyse huge datasets [8, 9]. Supervised learning and unsupervised learning are two important data mining problems. In supervised learning, the output is given. The task is to create a model from a given training data (data points with output) such that when a new data point without output comes, the model predicts the output of the data point. When the output is class, the problem is called classification problem. If the output is a real number, the problem is called a regression problem. Decision trees, Naïve Bayes classifiers, neural networks etc. are popular classifiers [8, 9]. In clustering, the output of data points are not provided, the task is to group data points in different clusters such that the data points in a cluster are more similar than the data points in different clusters. K-means clustering, Hierarchical clustering etc. are popular clustering methods [8, 9].

The problem discussed in the paper is a classification problem as we want to predict whether given software has bugs or not.

Classifier ensembles are methods to improve the accuracy of classification algorithms. Classifier ensembles are combination of accurate and diverse classifiers. The final result of an ensemble depends on output of individual classifiers. Classifier ensembles generally perform better than single classifiers [10]. Bagging [11] and Boosting [12] are two popular classifier ensembles.

Some datasets have class-imbalanced problem [13]. In two-class setting, most of the data points belong to one class. This problem is also called imbalanced dataset problem. The aim of this problem is to detect a rare but important case. The classifier algorithms may not perform well on these datasets. To overcome this problem, different approaches have been suggested. Undersampling of majority class and oversampling of minority class are two popular preprocessing approaches to overcome class-imbalanced problem [13]. A classifier algorithm with a cost matrix is another approach to overcome this problem. The cost matrix gives more weightage to classification error of minority class point. The decision threshold is shifted towards minority class. In other words, the trained model is biased towards minority class [13].

Software repositories contain datasets about software projects. Mining these datasets can be used to uncover interesting and actionable information about software projects [14, 15]. This information can be used by software practitioners to improve the quality of software projects. Mining software engineering data has emerged as a research direction in test automation. Application of data Mining software metrics have been widely used in the past for predictive and explanatory purposes. These techniques try to find out relationship between software metrics and the bugs in a program. Therefore, these techniques can predict which software program is more likely to have bugs without doing any manual testing [14, 15].

Czibula et al. [16] apply association rules to predict software defects. Moeyersomsa et al. [17] use decision tree ensembles with rule extraction algorithm for comprehensible software fault and effort prediction. El-Sebakhy [18] applies functional networks forecasting framework to forecast software development efforts. Dejaeger et al. [19] present a comparative study of various data mining techniques for software efforts. The study suggests that Least Squares Regression in combination with a logarithmic transformation performs best.

Cotroneo et al. [20] use various data mining techniques to predict which source code files are more prone to Aging-Related Bugs. Despite the fact that the datasets are imbalanced they use the normal data mining techniques. In this paper, a detailed study of general classification algorithms and algorithms for imbalanced data is carried out on an Aging-Related Bugs dataset.

This paper is organized in the following way. In Section 2, we discussed the data mining techniques and the dataset used in the study. The results are discussed in Section 3. Conclusion and future work is presented in Section 4.

II. Material & Methods

We used various general data mining techniques and data mining techniques for imbalanced datasets in our study. In this section, we will discuss these techniques. We will also discuss about the dataset used in the study.

2.1 General Data Mining Techniques

i. **Decision Trees** - Decision trees are very popular classifiers [8, 9]. They create rules which human can understand easily. These trees are created by recursive partitioning. Started with root node, at each node of the tree the data points available at the node are split in different braches depending upon the chosen split criterion eg. Information Gain ratio, Gini index etc. Splitting process terminates after a particular stopping criterion is reached. C4.5 [21] tree is a very popular decision tree, it uses Information Gain Ratio criterion.

ii. **Multilayers Perceptron (MLP)** - Neural networks classifiers are inspired by biological neural networks [8, 9]. They are interconnected nodes. These networks have weights which represent connection strengths between neurons. These neural networks can approximate all kinds of decision boundaries [8, 9]. They are very popular because of their high accuracy. However, they are slow. The selection of parameters is an important step, improper parameters can reduce the accuracy of neural networks.

iii. **Bagging** - Bagging is a method to create classifier ensembles [11]. This is a general method and can be used with any classifier. As discussed in Section 1, for classifier ensembles we need accurate and diverse classifiers. Bagging creates diverse classifiers by resampling. In each round, the training datasets is created by sampling the data from training data uniformly and with replacement. Hence, in each round some data points are not selected whereas some data points are selected more than one time. This process create diverse datasets, classifiers trained on these datasets are diverse and hence create accurate ensembles.

iv. **AdaBoost.M1** - AdaBoost.M1 is a popular classifier ensembles method [12]. In each round of training, weights are assigned to each data points such that the hard to classify data points will get more weights. In other words, in subsequent rounds, classifiers try to concentrate more on difficult to predict data points. It

is a general method and work with all kinds of classifiers. However, it is more successful with decision trees and neural networks. AdaBoost.M1 has problem with datasets with class noise.

2.2 Data mining techniques for imbalanced datasets

i. **C4.5 for imbalanced datasets** – Cost-sensitive classifiers are developed for imbalanced datasets [13]. For normal datasets, while calculating accuracy, all data points have same weights, whereas, in classifiers with cost-sensitive matrix, the error of data points of minority class is given more weight. In other word, it is better to have classification error for majority class data points than for minority class data points. The classifier tries to minimize the error function with cost matrix. Ting [22] proposes cost-sensitive trees in which the weights are assigned to each data point and the initial data point weights determine the type of tree. The selection of cost-matrix is an important step in the algorithm.

ii. **Multilayer Perceptron with Back Propagation Training Cost-Sensitive (NNCN) -** Cost-sensitive learning technique is also useful for neural networks. Zhou et al. [23] show that the threshold-moving approach can be useful for neural networks for imbalanced datasets. For normal datasets the threshold for classification is in the middle of the output range. Threshold-moving moves the output threshold toward inexpensive classes such that data points with higher costs (minor class data points) become harder to be misclassified.

iii. **SMOTEbagging** – Undersampling of majority class and over-sampling of minority class are used to reduce the class imbalance. Chawla et al [24] propose an over-sampling approach (Synthetic Minority Over-sampling Technique (SMOTE)) in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the *k* minority class nearest neighbours. They show that combining the proposed under-sampling technique with general over-sampling method produce good results. In SMOTEbagging [25], in each round a training dataset is created by bagging process and then the data imbalanced is removed by SMOTE process. A classifier is trained on this dataset. Classifiers trained in different rounds are combined to get an ensemble.

iv. **SMOTEBoost –** SMOTEboost combines SMOTE sampling technique with boosting [26]. Unlike standard boosting, where all misclassified examples are given equal weights, in SMOTEBoost technique the weights are updated such that the data points from the minority class are oversampled by creating synthetic minority class examples.

v. **MSMOTEbagging** – Modified synthetic minority oversampling technique (MSMOTE) [27] is a modified version of SMOTE. In this algorithm, the minority class data points are divided into three subclasses, safe, border and latent noise instances by the calculation of the distances among all data points. For safe data points, the algorithm randomly selects a data point from the *k* nearest neighbours; for border data points, it only selects the nearest neighbour; for latent noise instances, it does nothing. MSMOTE combines with bagging (as SMOTE combine with bagging to create SMOTEbagging) to create MSMOTEbagging.

vi. **MSMOTEboosting** – MSMOTEboosting [28] combines the MSMOTE sampling method with AdaBoost. This way the minority class is better represented in each round of AdaBoost. For many datasets, MSMOTEboost has shown better results than SMOTEBoost.

vii. **Underbagging** – Undersampling technique is used to reduce the number of majority class data points so that the data imbalanced can be reduced. In Underbagging, a training dataset is created by under-sampling majority class randomly to build a classifier. Various diverse classifiers created by this method are combined to create an Underbagging ensemble [29].

viii. **Overbagging** – To reduce data imbalance, oversampling is used to increase the number of minority class data points. In Overbagging, a training dataset is created by oversampling minority class randomly to build a classifier. These diverse classifiers are combined to get an ensemble. This method is called Overbagging [25].

2.3 Dataset Used In The Study

Cotroneo et al. [20] present a data set for ARB found in MySQL_DBMS (dataset_mysql_innodb-http://wpage.unina.it/roberto.natella/datasets/aging_related_bugs_metrics/dataset_mysql_innodb.arff). The data set has 402 files. 370 files has no bugs (0), whereas 32 files has age related bugs (1). It can be considered as a two class problem. It is an imbalanced data as the ratio of minority class to majority class is 1 to 11.6. Each file is represented by 83 attributes. The first attribute is the name of the file. 49 attributes come from "Program size" metrics for the file. These metrics are related to amount of lines of code, declarations, statements, and files. 18 attributes came from "McCabe's cyclomatic complexity" metrics for the file. These metrics represent the control flow graph of functions and methods. "Halstead" metrics for the file are represented by 9 attributes.

These metrics represent operands and operators in the program. "Aging-related" metrics for the file are represented by 6 attributes. These attributes are related with memory usage. For detail information about the attributes readers may refer to [20].

III. Experiments

Experiments were carried out by using the KEEL software [30]. Table 1 presents the implementation of different data mining algorithms. Defaults parameters were used in the experiments.

Table 1- Implementation of different data mining algorithms in KEEL software.

Name of the algorithm	KEEL implementation
C4.5 [21]	Decision trees
MLP-normal [8, 9]	Multilayer Perceptron with Back propagation Training
Bagging [11]	Bootstrap Aggregating with C4.5 Decision Tree as Base Classifier
Adaboost.M1 [12]	Adaptive Boosting First Multi-Class Extension with C4.5 Decision Tree as Base Classifier
C4.5-Imbalanced [22]	C4.5 Cost-Sensitive
NNCS [23]	Multilayer Perceptron with Back propagation Training Cost-Sensitive
SMOTEbagging [25]	Synthetic Minority Over-sampling Technique Bagging with C4.5 Decision Tree as Base Classifier
SMOTEadaboost [26]	Synthetic Minority Over-sampling Technique Boosting with C4.5 Decision Tree as Base Classifier
MSOMTEbagging [27]	Modified Synthetic Minority Over-sampling Technique Bagging with C4.5 Decision Tree as Base Classifier
MSMOTEboosting [28]	Modified Synthetic Minority Over-sampling Technique Boost with C4.5 Decision Tree as Base Classifier
Overbagging [25]	Over-sampling Minority Classes Bagging with C4.5 Decision Tree as Base Classifier []
Underbagging [29]	Under-sampling Minority Classes Bagging with C4.5 Decision Tree as Base Classifier

3.1 Performance Measures

There are many performance measures for classifiers. However, all performances measures are not good for imbalanced datasets. First, we discuss various performance measures. Then, we will explain which performance measures are good for imbalanced datasets [13]. The present problem is a two-class problem. The minority class (Bug class) is taken as positive class whereas the majority class (Non-Bug class) is taken as negative class. Table 2 presents the confusion matrix for this problem.

Table 2 - Confusion Matrix

		True Class	
		Positive	**Negative**
Predicted Class	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

3.1.1 Definitions of various performance measures

1. Accuracy – Accuracy performance measure calculates the number of data points correctly predicted. It is calculated by the following formula;

Accuracy = (TP + TN)/(TP + FN + FP + TN);

2. Recall (True positive rate (TPR)) – It calculates the fraction of positive class data points predicted as positive. The following formula is used to calculate it;

Recall (TPR) = TP/(TP + FN);

3. True negative rate (TNR) = It calculates the fraction of negative class data points predicted as negative. It is calculated by using the following formula;

TNR = TN/(TN + FP)

4. Average of TNR and TPR – The average of TNR and TPR is a performance measure which depends on both TNR and TPR. It is calculated by using following formula;

Average = (TNR + TPR)/2

5. G-mean – The geometric mean of TNR and TPR is another performance measure which depends on both TNR and TPR. This is a very popular performance measure for imbalanced datasets. It is calculated by following formula;

$$\text{G-mean} = (\text{TPR} \times \text{TNR})^{1/2}$$

6. Precision- Precision is the fraction of predicted positive class that is actually positive class. It is calculated by following formula;

Precision = TP/(TP + FP)

7. F-measure – F-measure depends on recall and precision. It is another important performance measure. It is calculated by following formula

F-measure = 2*Precision*Recall/(Precision + Recall)

Not all performance measures are good for imbalanced datasets. Accuracy is not a good measure for these datasets as any classifier which will classify all the points to majority class will be very accurate. However, this is not desirable. The most important thing is that the classifier should be able to predict minority class data points well (High TPR). At the same time it should have high TNR. The G-mean is a good measure for imbalanced datasets. F-measure which depends on recall and precision is also a good measure [13]. The user should select the performance measure depending upon his application.

IV. Results And Discussion

The experiments were carried out by using the 5 fold cross validation. The results are presented in Table 3 and Fig 1 – Fig. 5. Results suggest that the accuracy of general machine learning algorithms is high (> 0.9). TNR is also quite high. However, TPR is generally low. This is due to the fact that these algorithms are predicting most of the testing data points as Non-Bug. As the number of Bug data points is very small (32 data points out of 402 data points). The classification accuracy is quite high. However, these algorithms are not able to capture Bug data points correctly. This is the reason of low TPR. The G-mean which is the most important performance measure for imbalanced datasets is low for these algorithms [13]. Generally, the important point of these kinds of datasets is that we should be able to predict minority class correctly (High TPR) without affecting the predicting accuracy of majority class much (High TNR). The G-mean captures this fact. Low G-mean means that they are not able to do it. Generally, these algorithms also have low F-measure.

Table 3- A comparative study of various data mining techniques with different performance measures. The bold result shows the best performance for a given performance measure.

	Criterion / **Method**	**Accuracy**	**TPR (Recall)**	**TNR**	**Mean = TPR +TNR/2**	**G-mean = (TPRxTNR)$^{1/2}$**	**Precision**	**F-measure**
General Machine Learning Methods	C4.5	0.9080	0.1250	0.9757	0.5503	0.3492	0.3077	0.1778
	MLP-normal	0.6045	0.5938	0.6054	0.5996	0.5995	0.1152	0.1929
	Bagging	0.9080	0.0625	**0.9811**	0.5218	0.2476	0.2222	0.0976
	Adaboost.M1	**0.9154**	0.2813	0.9703	0.6258	0.5224	**0.4500**	**0.3462**
Machine Learning methods for imbalanced datasets	C4.5-Imbalanced	0.6841	**0.9375**	0.6622	**0.7998**	**0.7879**	0.1935	0.3209
	NNCN	0.5821	0.6875	0.5730	0.6302	0.6276	0.1222	0.2075
	SMOTEbagging	0.6741	0.8750	0.6568	0.7659	0.7581	0.1806	0.2995
	SMOTEadaboost	0.7488	0.6875	0.7541	0.7208	0.7200	0.1947	0.3034
	MSOMTEbagging	0.7164	0.7813	0.7108	0.7460	0.7452	0.1894	0.3049
	MSMOTEboosting	0.7463	0.6250	0.7568	0.6909	0.6877	0.1818	0.2817
	Overbagging	0.8234	0.3750	0.8622	0.6186	0.5686	0.1905	0.2526
	Underbagging	0.6990	0.8750	0.6838	0.7794	0.7735	0.1931	0.3164

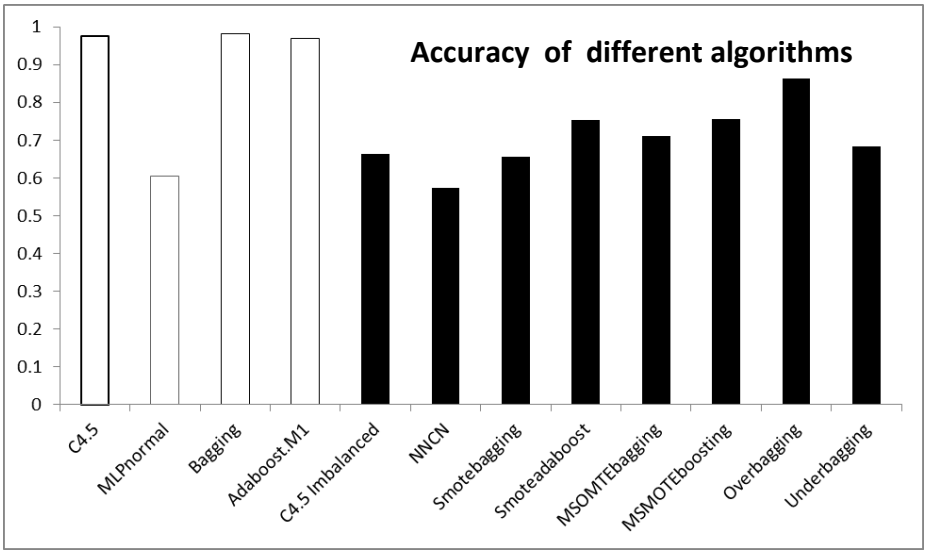

Figure 1- Accuracy of different data mining algorithms. General data mining algorithms, represented by white columns, show better accuracy as compared to data mining techniques for imbalanced datasets, represented by black columns.

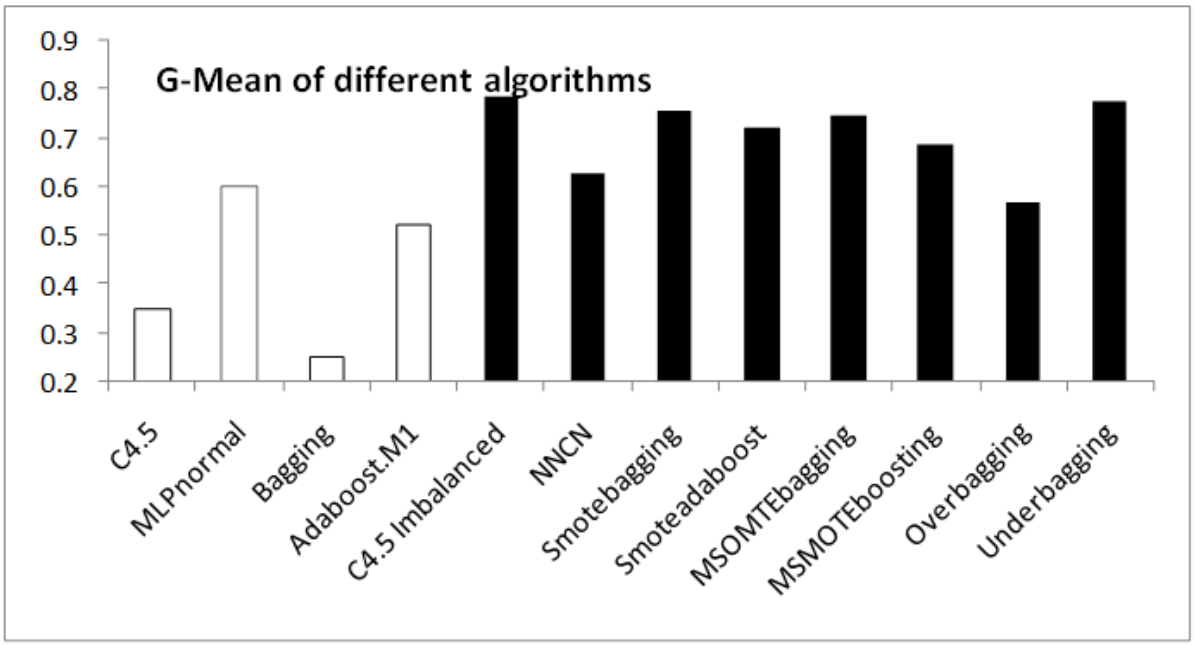

Figure 2- G-mean of different data mining algorithms. Data mining techniques for imbalanced datasets, represented by black columns, show better G-mean than that of general data mining algorithms, represented by white columns.

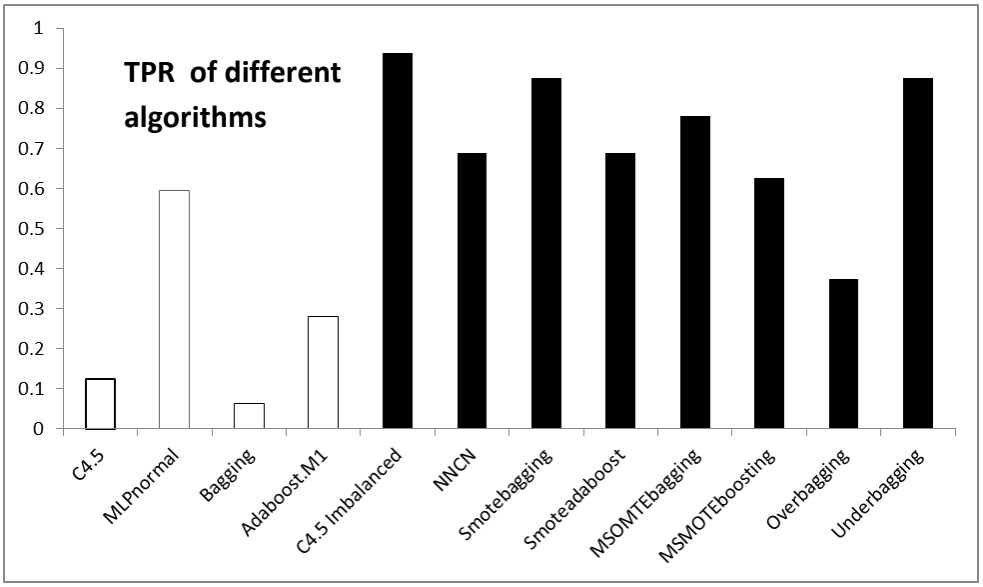

Figure 3- TPR of different data mining algorithms. Data mining techniques for imbalanced datasets, represented by black columns, show better TPR than that of general data mining algorithms, represented by white columns.

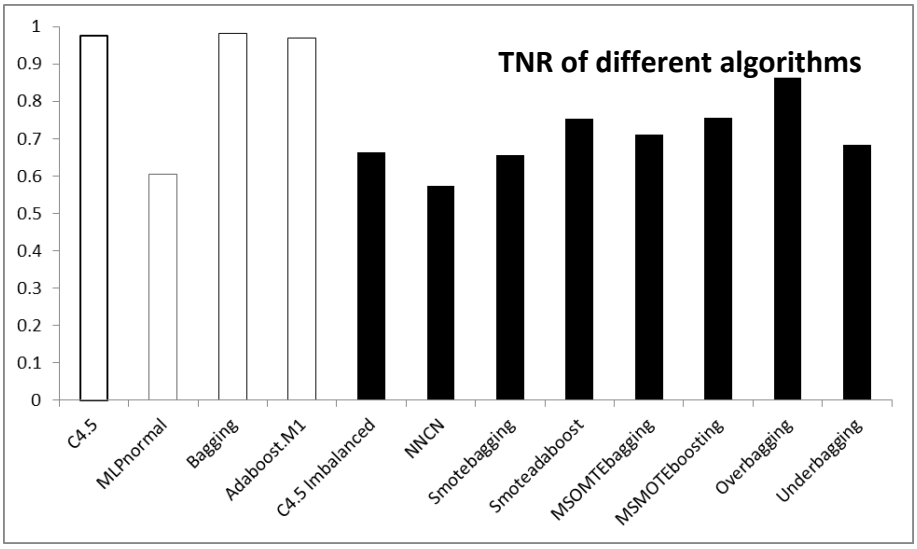

Figure 4- TNR of different data mining algorithms. General data mining algorithms, represented by white columns, show better TNR as compared to data mining techniques for imbalanced datasets, represented by black columns.

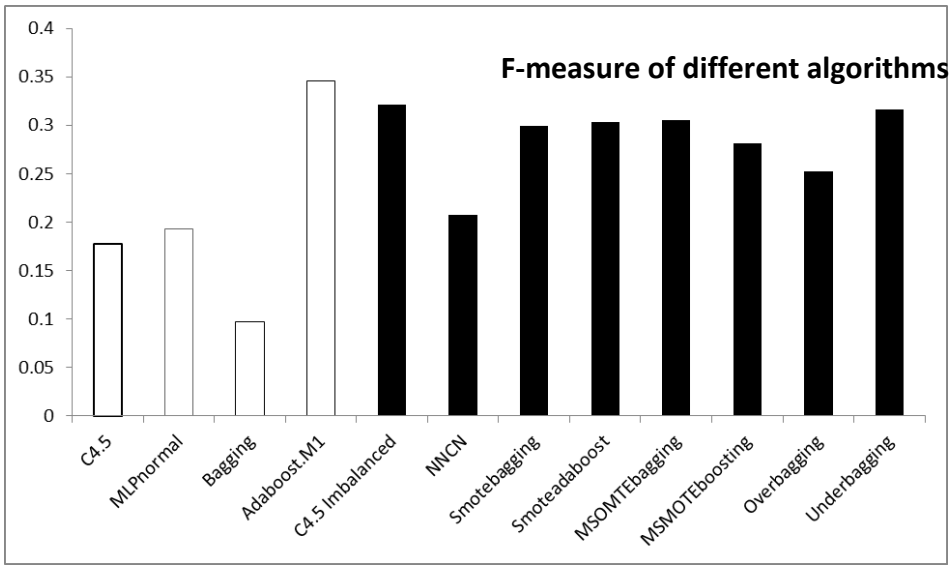

Figure 5- F-measure of different data mining algorithms. Data mining techniques for imbalanced datasets, represented by black columns, show better F-measure than that of general data mining algorithms, represented by white columns.

Table 4- Confusion matrix for C4.5 decision tree.

		True Class	
		Bug	No-Bug
Predicted Class	Bug	4	9
	No-Bug	28	361

Table 5- Confusion matrix for AdaBoost.M1.

		True Class	
		Bug	No-Bug
Predicted Class	Bug	9	11
	No-Bug	23	359

Table 6- Confusion matrix for C4.5-Imbalanced decision tree.

		True Class	
		Bug	No-Bug
Predicted Class	Bug	30	125
	No-Bug	2	245

To understand this in a better way, the confusion matrix of C4.5 classifier is presented in Table 4. The classifier predicted only 4 out of 32 Bug data points correctly. Hence, low TPR is achieved. Only 13 data points are classified as Bug. In other words, almost all the points are predicted Non-Bug (389 out of 402 data points).

Adaboost.M1 gives the best F-measure. To understand this in a better way, the confusion matrix of AdaBoost.M1 is presented in Table 5. The algorithm predicted 9 out of 32 Bug data points. However, the precision in this case is relatively high (0.4500). The high precision leads to high F-measure. However, the G-measure is relatively low (0.5224).

The accuracy of data mining algorithms for imbalanced datasets is relatively low. However, the G-mean is generally high (most of the cases more than 0.7). The best case is C4.5 for imbalanced dataset (0.7998). The TPR for these algorithms is relatively high. C4.5 for imbalanced dataset gave best TPR (0.9375). This suggests that these algorithms are able to predict large number of Bug data points without affecting TNR much. To analyse the performance, we presented the confusion matrix for C4.5 decision tree for imbalanced dataset in Table 6. The matrix suggests that the algorithm was able to predict 30 out of 32 Bug data points correctly. However, it also predicted a large number of Non-Bug data points as Bug data points. That leads to lower TNR as compare to those of general data mining algorithms. The F-measure of these algorithms is also relatively high.

It is important to note that the selection of performance measures for a given problem is an important step. The G-mean and F-measure are the best performance measure for the imbalanced datasets. The data mining algorithms for imbalanced datasets performed well on these two measures. The results suggest that for Aging Related Bug problem in which the number of Bug data points is small, data mining algorithms for imbalanced datasets should be used.

V. Conclusion

In this paper, we applied data mining techniques for imbalanced datasets to predict ARBs in a dataset related with an open source software; MYSQL DBMS [20]. Results suggest that these data mining algorithms are more useful than general data mining algorithms for predicting ARBs. Data mining algorithms for imbalanced datasets have better prediction (TPR) for Bugs data points (minority class). These algorithms also have better G-mean measure. More datasets will be used in the future. Feature selection techniques will also be applied in future to remove the insignificant features. One class classifier is an approach to analyse dataset when we have points of only one class [31]. In future, we will see the application of one class classifiers for predicting ARBs.

References

[1]. S. L. Pfleeger and J. M. Atlee, Software Engineering: Theory and Practice, Prentice Hall; 4 edition, 2009.
[2]. Ian Sommerville, Software Engineering, Pearson; 9 edition, 2010.
[3]. M. McDonald, R. Musson and R. Smith, The Practical Guide to Defect Prevention, Microsoft Press, 2007.
[4]. M. Dowson, The Ariane 5 Software Failure, Software Engineering Notes 22 (2): 84. 1997.
[5]. D. Huizinga, Dorota and A. Kolawa, Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. 2007.
[6]. M. Grottke, R. Matias, R., and K. S. Trivedi, The fundamentals of software aging, In IEEE Proceedings of Workshop on Software Aging and Rejuvenation, in conjunction with ISSRE. Seattle, WA. 2008.
[7]. D. Controneo, R. Natella, R. Pietrantuono, and S. Russo, A survey on software aging and rejuvenation studies, IACM Journal on Emerging Technologies in Computing Systems (JETC), 2014.
[8]. J. Han, M. Kamber and J. Pei, Data Mining: Concepts and Techniques, Morgan Kaufmann; 2011.
I. H. Witten, E. Frank, M. A. Hall (Author) Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann; 2011.
[9]. L. I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms. Wiley-Interscience, 2004.
[10]. L. Breiman. Bagging predictors. Machine Learning 24, 1996, pp. 123-140.
[11]. R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. Machine Learning 37, 1999, pp. 297-336.
[12]. N. Chawla, Data mining for imbalanced datasets: An overview. In. Maimon O., Rokach L. (eds.) The Data Mining and Knowledge Discovery Handbook, Springer 2005, pp. 853-867.
[13]. A. E. Hassan and T. Xie. Mining software engineering data. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, 2010, pp. 503–504.
[14]. H. H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance, 19(2), 2007, pp. 77–131.
[15]. G. Czibula, , Z. Marian and I. G. Czibula, Software defect prediction using relational association rule mining. Information Sciences, 264, 2014, pp. 260–278.
[16]. J. Moeyersomsa, E. Junqué de Fortunya, K. Dejaegerb, , B. Baesensb, D. Martensa, Comprehensible software fault and effort prediction: A data mining approach, Journal of Systems and Software, Volume 100, Feb. 2015, pp. 80–90.
[17]. E. A. El-Sebakhy, Functional networks as a novel data mining paradigm in forecasting software development efforts, Expert Systems with Applications, 38(3), March 2011, pp. 2187–2194.
[18]. K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens. Data mining techniques for software effort estimation: A comparative study. IEEE TSE, 38(2), 2012, pp. 375–397.
[19]. D. Cotroneo, R. Natella, R., and R. Pietrantuono, Predicting aging-related bugs using software complexity metrics. Performance Evaluation, 70(3), 2013, pp. 163–178.

[20]. J.R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kauffman, 1993.
[21]. K.M. Ting. An instance-weighting method to induce cost-sensitive trees. IEEE Transactions on Knowledge and Data Engineering 14:3, 2002, pp. 659-665.
[22]. Z.-H. Zhou, X.-Y. Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. IEEE Transactions on Knowledge and Data Engineering 18:1, 2006, pp. 63-77.
[23]. N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Oversampling TEchnique. Journal of Artificial Intelligence Research,16, 2002, 321–357.
[24]. S. Wang, X. Yao. Diversity analysis on imbalanced data sets by using ensemble models. IEEE Symposium Series on Computational Intelligence and Data Mining (IEEE CIDM 2009). Nashville TN (USA, 2009), 2009, pp. 324-331.
[25]. N.V. Chawla, A. Lazarevic, L.O. Hall, K.W. Bowyer. SMOTEBoost: Improving prediction of the minority class in boosting. 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2003). Cavtat Dubrovnik (Croatia, 2003), 2003, pp. 107-119.
[26]. S. Wang, X. Yao. Diversity analysis on imbalanced data sets by using ensemble models. IEEE Symposium Series on Computational Intelligence and Data Mining, Nashville TN, USA, 2009, pp. 324-331.
[27]. S. Hu, Y. Liang, L. Ma, Y. He. MSMOTE: Improving classification performance when training data is imbalanced. 2nd International Workshop on Computer Science and Engineering, Qingdao, China, 2009, pp. 13-17.
[28]. R. Barandela, R.M. Valdovinos, J.S. Sánchez. New applications of ensembles of classifiers. Pattern Analysis and Applications 6, 2003, pp. 245-256.
[29]. J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. Journal of Multiple-Valued Logic and Soft Computing 17:2-3, 2011, pp. 255-287.
[30]. S.S. Khan, and M.G. Madden . A survey of recent trends in one class classification. Lect. Notes Comput. Sci. 6206, 2010, pp. 188−197.