On Studying the Effect of Sample Size in Evaluation of Bug Classifiers

Naresh Kumar Nagwani, Dr. Shrish Verma

[1]*Assistant Professor, Computer Science & Engineering;* [2]*Associate Professor & Head, Electronics & Tel. Communication Engg.*
National Institute of Technology Raipur,
[1]nknagwani.cs@nitrr.ac.in, [2]shrishverma@nitrr.ac.in

Abstract

Sampling is an important and necessary step in mining large size databases and is also very useful in performing mining operations, where performance is a critical issue. This study focuses on identifying the effect of sample size in classification of software bugs. To analyze the effect of sample size, experiments are performed using a number of classification algorithms with verities of sample sizes using the software bug repositories of three large open source software's namely Android, Mozilla and MySql. The relationship between the sample size with two primary classification performance parameters accuracy and F-measure is explored in this study. From experiments, it is identified that the parameter F-measure is affected more by the sample size than accuracy.

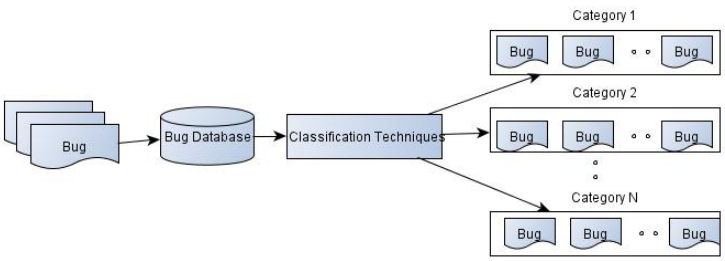
Keywords: Sampling, Sample Size, Classification, Software Bug, Performance, Classifier Evaluation

1. Introduction

Software bug classification is the task of categorizing the software bugs into the various pre-defined categories. Software bug classification is an important activity in software bug mining; it helps in understanding the domain of bug and its effects on the modules. A bug can be of any type like bugs related to memory, graphical user interface (GUI) or logical etc. The bug classification process is shown in Fig. 1. Generally the classification is performed in three stages, in the first stage the software bugs are collected from repositories, pre-processed and stored in the local database for applying data mining operations. In second stage classification algorithm is applied for categorizing the bugs and finally in third stage classification parameters are evaluated and result is post-processed for output representation.

Software bug repositories are not only consisting of defect information but also having the information of enhancement and other development tasks. The enhancement and small development tasks are also treated like defects which need to be implemented and resolved, and mostly included in the bug repositories. The software bug repositories for bigger software's are consist of thousands of the software bugs and its size (number of bugs) is still growing because of more bugs of newly modules and enhancement works are reported frequently. It is not possible to accommodate all the thousand number of bugs for evaluation and development of bug classifier, because it will make the classifier development process complex. To simplify this process sampling techniques can be used effectively. Sampling plays a major role in classification, and almost everywhere where the classifiers are developed, sampling techniques are frequent used for faster development and validation of the classifiers.

Fig.1. *Software Bug Classification Process.*



1.1 Classification Algorithms

There are a number of bug classification techniques proposed [Antoniol et. al., 2008; Ferzund et. al., 2009; Fluri et. al., 2008; Grottke and Trivedi, 2005; Guo and Sampath, 2008]. A software bug is consisting of a number of attributes like summary, description, comments, reported-date, assigned-to, reported-by etc. The textual attributes (surface features) summary, description and comments consist of important information of a bug and are the primary features used in classification [Jalbert and Weimer, 2008; Antoniol et al, 2008]. To evaluate the effect of sample size on various software bug classifiers five different classification algorithms are selected for experiments in this paper. The algorithms are Naïve Bayes (NB) classifier [Mccallum and Nigam, 1998], J48 [Quinlan, 1993], Support Vector Machine (SVM), [Vapnik, 1995; EL-Manzalawy, 2005], Classification Using Clustering (CC) [Kyriakopoulou and Kalamboukis, 2006] and CLUBAS (CLassification of software bugs Using Bug Attributes Similarity) [Nagwani and Verma, 2012b].

1.2 Naïve Bayes Classifiers (NB)

The Naïve Bayes classifier [Mccallum and Nigam, 1998] works on the basis of Bayes rule of conditional probability. It uses every attribute contained in the data, and analyses them in-

dividually. It is assumed that all the attributes are uniformly important and independent of each other.

1.3 J48

J48 [Quinlan, 1993] is one of the decision tree classifiers. A decision tree is a predictive machine-learning model that decides the dependent value of a new sample based on diverse attribute values of the existing data. The internal nodes of a decision tree denote the different attributes; the branches between the nodes indicates the possible values that these attributes can have in the observed samples, where as the leaf nodes indicates the final value (class values) of the dependent variable. The J48 decision tree classifier follows the simple algorithm to classify a new item; it first needs to create a decision tree based on the attribute values of the available training data. So, whenever it encounters a set of items (training set), it identifies the attribute that discriminates the various instances most clearly.

1.4 Support Vector Machine (SVM)

A support vector machine (or SVM) [Vapnik, 1995; EL-Manzalawy, 2005] is an algorithm that works as follows. It uses nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyper-plane. With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using support vectors and margins. SVM's are supervised learning methods used for classification.

1.5 Classification Using Clustering (CC)

In classification using clustering [Kyriakopoulou and Kalamboukis, 2006] technique, classification is performed using text clustering. The algorithm consists of three steps: clustering, expansion and classification step. It is considered that there is a one-to-one association between classes, topics and clusters.

1.6 Classification of Bugs Using Bug Attributes Similarities (CLUBAS)

CLUBAS [Nagwani and Verma, 2012b] is a hybrid algorithm, and is designed by using text clustering, frequent term calculations and taxonomic terms mapping techniques. The algorithm CLUBAS is developed using classification using clustering model. The algorithm works in three major steps, in the first step text clusters are created using software bug textual attributes data and followed by the second step in which cluster labels are generated using label induction for each cluster, and in the third step, the cluster labels are mapped against the bug taxonomic terms to identify the appropriate categories of the bug clusters.

2. Classifier Performance Evaluation

The accuracy and performance of prediction models for classification problem is typically evaluated using a confusion matrix. A confusion matrix contains information about actual and predicted classifications performed by classifiers. In this study, accuracy and F-measure are used to evaluate the classification models. These measures are derived from the confusion matrix, which is shown in the Fig. 2. TP stands for true positive, which indicates a positive value that the system has predicted as positives, TN is true negatives, that is negative values the system identifies as negatives, FP is false positives, negative values the system identifies as positives and FN is false negatives, positive values that the system predicted as negative.

Fig.2. *A confusion matrix*

		Predicted	
		positives	Negatives
Actual	positives	TP	FN
	negatives	FP	TN

Precision is the ratio of the number of correctly classified software bugs and the actual number of software bugs which was assigned to the type. Precision is the measurement of correctness and is also defined as the ratio of the true positives (TP) to total positives (TP+FP) and is calculated using Eq. (1). Recall rate is the ratio of the number of correctly classified software bugs and the number of software bugs which belongs to the type. It reflects the classifier's ability of searching extension and is calculated using Eq. (2).

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2)$$

F-Measure or F_1-Measure is a combined measure of Precision and Recall parameters. F-measure considers both precision and recall equally important by taking their harmonic mean. F-measure or F_1-measure is calculated using Eq. (3). Accuracy, or correctness of classifiers, is defined as the ratio of the number of bugs correctly classified to the total number of bugs and is calculated using Eq. (4) or Eq. (5).

$$F_1 = \frac{2 * precision * recall}{precision + recall} \quad (3)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

$$Accuracy(\%) = \frac{Correctly\_Classifies\_Software\_Bugs}{Total\_Software\_bugs} * 100 \quad (5)$$

3. Experiments

3.1 Data (Bug Repositories)

To evaluate the effect of sample size on software bug classifiers, bugs are selected from three major open source software repositories namely Android, Mozilla and MySql. There repositories contains the bug information online and can be accessed online by any user.

3.2 Sampling

Random sampling technique is used in the experiments, and to fetch the random records from the software bug repositories, random number generator has written in java to generate the random integer numbers. Using these numbers as the bug ids, bug records are extracted from the online software bug repositories using URL (Uniform Resource Locator) programming in java. (For example for the URL of Mozilla bug repositories, "https://bugzilla.mozilla.org/show_bug.cgi?id=***X***", where ***X*** can be appended to retrieve a particular software bug record from Mozilla repository.) The sample size of 200, 300, 500, 700, 1000, 1300, 1600 and 2000 is taken for the experiments from the repositories for the evaluation of the classifiers.

3.3 Programming Environment

Experiments are performed using Java and Weka API (Waikato Environment Knowledge Analysis – Application Programming Interface) for the implementation of random number generation (for bug-ids) and implementation of classification algorithms. Weka API provides the implementation of most of the algorithms used in this paper for the study of sample size effect on classifiers. Support Vector Machine (SVM) classification algorithm is implemented with the help of LIBSVM. LIBSVM [Chang and Lin, 2001] is an open source implementation of SVM, which can be integrated into Weka. The 10-fold cross validation technique is applied to measure the accuracy of these classifiers. The accuracy and F-measure performance parameters are calculated for all the classifiers.

3.4 Pre-processing and Generation of Taxonomic Terms

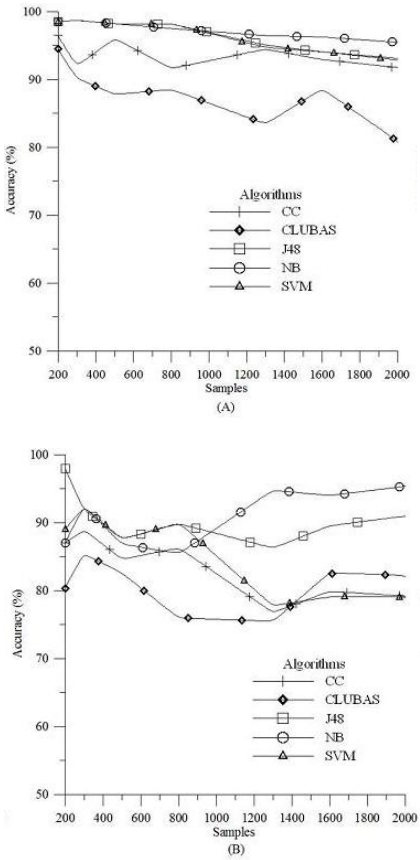
The taxonomic terms are generated using frequent terms analysis, tf-idf (Term Frequency – Inverse Domain Frequency) analysis and manual inspection [Nagwani and Verma, 2012a]. These terms serves the purpose of making the differentiation of the software bugs for different categories. Terms are generated using 2000 random sample software bugs selected from the three open source bug software projects in aggregation. Experiments in this paper are performed for binary classification of the bugs, where the bugs are categorized into the two categories – bugs and non-bugs (enhancement and other development tasks).

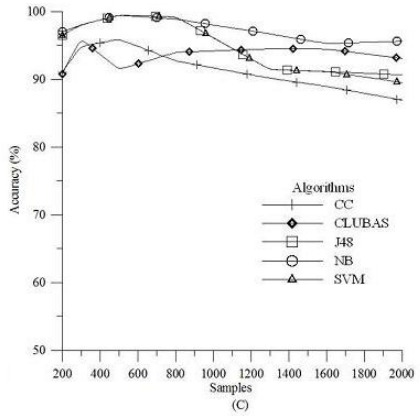
3.5 Classifier Evaluation

The experiment is first applied using the Android bug repository. Different numbers of random samples are selected from this repository and the algorithms CC, CLUBAS, NB, J48 and SVM are applied for classification and the parameters accuracy and F-Measures are calculated at each sampling point (for each sample size). The results for accuracy and F-measures for different number of samples are plotted in Fig. 3(A) and Fig. 4(A) respectively. From the plot Fig. 3(A), it is observed that initially as the sample size increases, some variation (minor increase and decrease) is identified in accuracy values, and after a certain level the accuracy values drops slightly. This is because the classifier update itself with new learning samples, and finally the classifier get established and gives the continuous stable accuracy values with slight variations. Accuracy wise the algorithms NB, SVM and J48 gives stable values of accuracy. Whereas the text clustering based algorithms CC and CLUBAS, some drop in accuracy values is observed with respect to increase in number of samples taken for classification. Another reason behind the behavior of accuracy is, at the initial level when the number of samples are less, less discriminative terms are identified for classification and classification gives better accuracy values, as the sample size increases more number of discriminative terms are identified by classification algorithm which causes drop in accuracy values because of higher true-negative and false-positive values (i.e. classifiers gets confused with the discriminative terms and causes wrong classification).

From the F-measure point of view (Fig. 4(A)), the same behavior like accuracy is observed in the experiments at different sampling point. Initially for small samples size some variation takes place in F-measure, then there is drop in F-measure values and finally the stable values are seen after the establishment of the classifiers learning and stability.

Fig.3. *Accuracy of various classifiers for (A) Android (B) Mozilla (C) MySql bug repository*



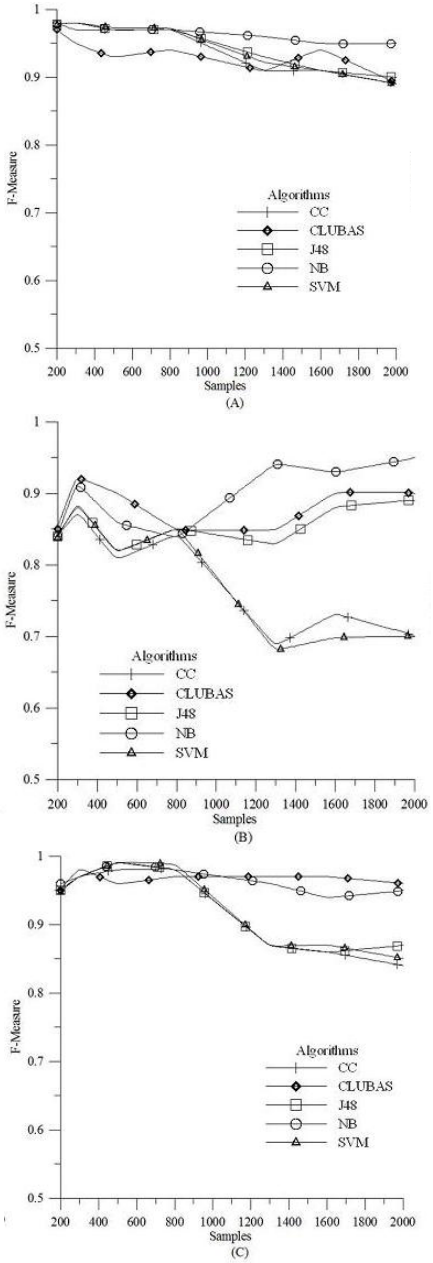


Similarly, the experiments are performed on Mozilla and Mysql bug repositories using the mentioned classification algorithms. The accuracy and F-measure results for these repositories are plotted against the different number of samples in Fig. 3-4(B-C) respectively. It is observed from the plots that the sample size is having less effect on the parameter accuracy, some minor drop in accuracy is observed with increase in the sample size for bug classification. But from the F-measure point of view the F-measure values starts falling after a certain increase in the sample size. The primary reason behind this is, as the number of bugs increase in the participation of classification, the classifiers discrimination system will get new information and upgrade itself for further learning, which results in drop of precision and recall values (and hence the F-measure, since F-measure is combined measure of precision and recall).

For NB, NBM and J48 algorithms, as sample size increase accuracy and F-measure fluctuate initially and then drops, after a particular level the values again increases and achieves stable values with minor fluctuations. This is due to the learning process and improvement in the probabilistic function for classifying the software bugs. At initial level, when there are less number of records (small sample size) the algorithms NB, NBM and J48 handles the discrimination of records effectively and hence the accuracy and F-measures and having good values, when the sample size starts increasing the discrimination power of the algorithms becomes less and there is drop in the parameters values (True-Positives decreases) and after a particular stage when there is enough learning in the probabilistic functions associated with the algorithms, again the parameter values increase (False-Positives and True-Negatives decreases and parameter values increases).

For clustering based (CC and CLUBAS) algorithms also the accuracy and F-measures fluctuates initially and after a particular stage the values drops and again fluctuate but never gain the higher values after this drop. The reason behind this is using these techniques an optimum number of text clusters are generated at any particular point then the newer data points falls in outliers.

Fig.4. *F-Measure of various classifiers for (A) Android (B) Mozilla (C) MySql bug repository*



For better understanding of sample size effect for various classifiers, 3D (three dimensional) views are created as contour maps. The contour map of parameter accuracy for the three bug repositories is presented in Fig. 5 (A-C). The sample size in thousands is mentioned along the X-axis, Y-axis represents the algorithm (numbered 1 to 5 i.e. 1-CC, 2-CLUBAS, 3-J48, 4-NB and 5-SVM) and accuracy values for the combination of X and Y values are represented as contour lines (Z values). It is observed from the figure that for Android and MySql bug repositories the accuracy values are always more than 80%, irrespective of sample size selected for classification, however in case of Mozilla bug repository the accuracy values drops below 80% (in the range of 60%-80%) for the algorithms CC, CLUBAS and SVM using the sample size more than 500. The primary reason behind this is observed manually which is the randomness in the textual

information present in Mozilla bug repository, the information in Mozilla repository is very vast and as the sample size grows more records are not likely to fit in the classifiers and hence minor drop in accuracy is observed. The other reason is CC, CLUBAS creates the optimum number of clusters and some bugs may fall is outliers resulting minor drop in accuracy values, similarly SVM optimizes the hyper-plane after enough increase in samples which may lead to minor drop in accuracy values.

Fig.5. *Contour Map of Accuracy of various classifiers for (A) Android (B) Mozilla (C) MySql bug repository*

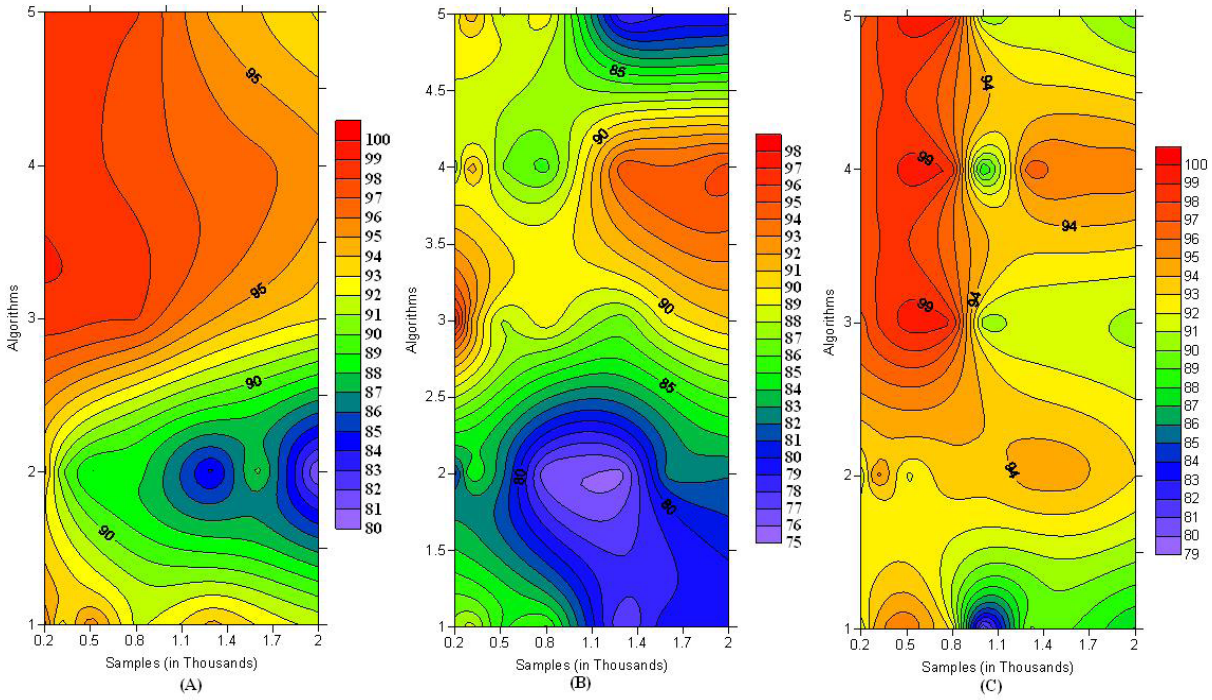
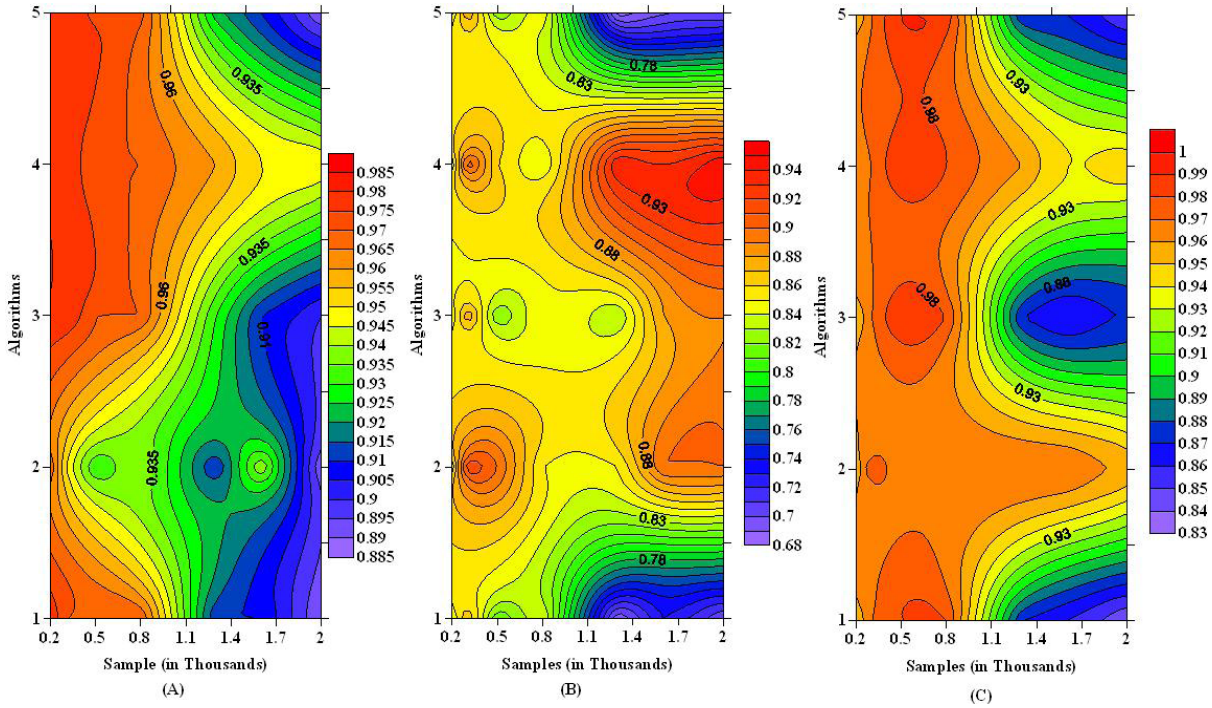


Fig.6. *Contour Map of F-Measure of various classifiers for (A) Android (B) Mozilla (C) MySql bug repository*



Similarly the 3D representation of F-measure for various classifiers using different number of samples is presented n Fig. 6 (A-C). The axis annotations are same as Fig. 5. From the figure it is observed that the F-measure values have more effect of change is sample size, it is found that after a certain point increase in sample size may drop the F-measures. Android bug repository is more affected than the other two bug repositories. This is because of the less correlation between the information present in the Android bug repositories and aggregate taxonomic terms generated for categorization of bugs from all the three bug repositories. The taxonomic terms are dominated by the other two software bug repositories i.e. Mozilla and MySql. The taxonomic terms are generated from the three repositories in aggregation and Mozilla and MySql bug repositories are dominating here being as mature repositories with more number of software bugs. Similarly minor effect is observed for Mozilla bug repository also, because of the same reason.

The above three dimensional representations can also be used for target programming, i.e. target based selection of sample size and algorithms. For example in Android bug repository if 95% accuracy is desirable then algorithm number 3, i.e. J48 algorithm can be used with the sample size around 1500. Similarly the same target driven programming models can be used for the F-measure values for different sample size and algorithms. These representations give the programmer a way to choose the algorithm and sample size for desired accuracy and F-measure values.

3.6 Threats to Validity

Three software bug's repositories namely, Android, Mozilla and MySql are selected for experimentations of identifying the effect of sample size for evaluating the bug classifiers. Different numbers of random samples are selected from every repository for analyzing the effect of sample size on various classifiers. Although the experiments are performed number of times and validation is also performed for the results, still there is a probability that the calculated parameter values may vary for other samples (selected randomly), where the textual bug attribute information is pitiable (the software bugs, where bug information is not described in details). The other limitation of the work can be derived from the Zipf's power distribution law [Li, 1992; Reed, 2001]. It states that most of users make use of restricted number of words commonly in the text documents. So if the selected records are user specific, some variation in results may also be observed.

4. Conclusions and Future Scope

The effect of sample size in bug classification task is explored in this work. The sample size parameter has the less effect on the classifier parameter accuracy and only minor variations are observed in accuracy for different number of samples, selected for classification. However the sample size have some effect on the F-measure values and it is observed from the experiments that as the sample size increases after a particular level, the F-measure values start decreasing and never get the higher values again. Two and three dimensional graphs are also generated for the analysis of sampling size effect in bug classification task. These graphs can also be used by programmers for target driven programming, where the programmers can choose the algorithms and sampling size for the desired values of accuracy and F-measure values. Random sampling is selected in this work for studying the effect of sample size, since it is the most common and effective sampling technique is software bug classification. Other sampling techniques like cluster sampling, stratified sampling and their effect on classification can also be taken as the future work to cover the overall sampling effect on the classification of software bugs.

5. References

1. Android Bug Repository - available at https://code.google.com/p/android/issues/list
2. Antoniol G, Ayari K, Penta M D (2008) Is it a Bug or an Enhancement? A Text-based Approach to Classify Change Requests. Proceedings of the 2008 conference of the center for advanced studies on collaborative research (CASCON '08), New York, USA, 304–318.
3. Chang C C, Lin C J (2001) LIBSVM - A Library for Support Vector Machines. URL http://www.csie.ntu.edu.tw/~cjlin/libsvm/.
4. EL-Manzalawy Y (2005) WLSVM: Integrating libsvm into WEKA environment. Software available at http://www.cs.iastate.edu/~yasser/wlsvm/.
5. Ferzund J, Ahsan S N, Wotawa F (2009) Software Change Classification using Hunk Metrics. Proceedings of IEEE International Conference on Software Maintenance (ICSM 2009), Edmonton, Canada, 471-474.
6. Fluri B, Giger E, Gall H C (2008) Discovering Patterns of Change Types. Proceedings of the 23rd International Conference on Automated Software Engineering (ASE), L'Aquila, Italy, 463-466.
7. Grottke M, Trivedi K S (2005) A Classification of Software Faults. Journal of Reliability Engineering Association of Japan, 27(7), 425-438.
8. Guo Y, Sampath S (2008) Web Application Fault Classification - An Exploratory Study. Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM 2008), Kaiserslautern, Germany, 303-305.
9. Jalbert N, Weimer W (2008) Automated Duplicate Detection for Bug Tracking Systems. IEEE International Conference on Dependable Systems & Networks, Anchorage, Alaska, 52-61.

10. Kyriakopoulou A, Kalamboukis T (2006) Text Classification Using Clustering. Proceedings of The 17th European Conference on Machine Learning and the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD),Burlin, Germany, 28-38.

11. Li W (1992) Random Texts Exhibit Zipf's-Law-Like Word Frequency Distribution. IEEE Transactions on Information Theory, 38(6), 1842-1845.

12. Mccallum A, Nigam K (1998) A Comparison of Event Models for Naive Bayes Text Classification. Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) Workshop on Learning for Text Categorization, Madison, Wisconsin, 41-48.

13. Mozilla (An open-source browser)Bug Repository, available at https://bugzilla.mozilla.org/

14. MySql - A free relational database management system, Bug Repository, available at http://bugs.mysql.com/

15. Nagwani N K, Verma S (2012) A Frequent Term Based Approach for Generating Discriminative Terms in Software Bug Repositories. IEEE 1st International Conference on Recent Advances in Information Technology (RAIT – 2012), Dhanbad, Jharkhand, India, 433-435.

16. Nagwani N K, Verma S (2012) CLUBAS: An Algorithm and Java Based Tool for Software Bug Classification Using Bug Attributes Similarities. Journal of Software Engineering and Applications, 5(6), 436-447.

17. Quinlan R (1993) C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA, ISBN 1-55860-238-0, 1-16.

18. Reed W J (2001) The Pareto, Zipf and other power laws. Economics Letters, 74(1), 15-19.

19. Vapnik V (1995) The Nature of Statistical Learning Theory. Springer-Verlag, ISBN:0-387-94559-8, 138-167.

20. Weka, available at http://www.cs.waikato.ac.nz/ml/weka/