Journal of Software Engineering and Applications, **2012, 5, 436-447**
http://dx.doi.org/10.4236/jsea.2012.56050 Published Online June 2012 (http://www.SciRP.org/journal/jsea)

CLUBAS: An Algorithm and Java Based Tool for Software Bug Classification Using Bug Attributes Similarities

Naresh Kumar Nagwani[1], Shrish Verma[2]

[1]Department of Computer Science & Engineering, National Institute of Technology Raipur, Raipur, India; [2]Department of Electronics & Telecommunication Engineering, National Institute of Technology Raipur, Raipur, India.
Email: {nknagwani.cs, shrishverma}@nitrr.ac.in

Received April 5th, 2012; revised May 1st, 2012; accepted May 10th, 2012

ABSTRACT

In this paper, a software bug classification algorithm, CLUBAS (Classification of Software Bugs Using Bug Attribute Similarity) is presented. CLUBAS is a hybrid algorithm, and is designed by using text clustering, frequent term calculations and taxonomic terms mapping techniques. The algorithm CLUBAS is an example of classification using clustering technique. The proposed 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. The cluster labels are generated using frequent and meaningful terms present in the bug attributes, for the bugs belonging to the bug clusters. The designed algorithm is evaluated using the performance parameters F-measures and accuracy. These parameters are compared with the standard classification techniques like Naïve Bayes, Naïve Bayes Multinomial, J48, Support Vector Machine and Weka's classification using clustering algorithms. A GUI (Graphical User Interface) based tool is also developed in java for the implementation of CLUBAS algorithm.

Keywords: Software Bug Mining; Software Bug Classification; Bug Clustering; Classification Using Clustering; Bug Attribute Similarity; Bug Classification Tool

1. Introduction

Software repositories are great source of knowledge. A software bug repository posses' useful information about software defects. A software bug record consists of number of attributes, many of which are of type text. To extract the knowledge from bug repositories text mining techniques can be used effectively. Classification is one of the popular data mining techniques to categorize the objects in a database. Software bug classification is the process of classifying the software bugs into different categories. A bug is a defect in software, which indicates the unexpected behavior of implemented software requirements and is identified during software testing process. Software bugs are managed and tracked using bug tracking tools, some example of such popular tools are Bugzilla [1], Perforce [2], JIRA [3], and Trac [4]. Software repositories contains the software bug information in the HTML (Hyper Text Markup Language) or XML (Extensible Markup Language) formats available through web interface, as online repositories. A software bug is consists of number of attributes like title/summary, description, reported-time, assigned-to, and user comments. Many of the important attributes such as summary, description and comments are of textual type. More details of the bug attributes and their descriptions are provided in the IEEE (The Institute of Electrical and Electronics Engineers) standard for the classification of software anomalies in 1993 [5], which was further revised on 2009 [6].

Majority of the work in mining software bug repositories includes predicting expert developers, predicting bugs, identifying duplicate bug reports, and classifying the change requests. An approach to recommend a ranked list of developers, to assist in performing software changes request is presented by Kagdi and Poshyvanyk [7]. An empirical evaluation of locating expertise developers is performed by Anvik and Murphy [8]. An approach is proposed by Kagdi *et al.* [9] that is based on Mining Software Repositories (MSR) to recommend a ranked list of candidate developers for source code change. Another approach named xFinder is proposed by Kagdi *et al.* [10]

to recommend expert developers by mining version archives of a system. Some investigation and analysis on the bug fixing data is done by Ayewah and Pugh [11].

A technique for predicting latent software bugs is proposed by Kim *et al*. [12], which classifies the change requests as buggy or clean. Vijayaraghavan and Kaner [13] have worked on classifying the testers (software quality engineers) in two categories; deductive testers and inductive tester. Antoniol *et al*. [14] pointed out that not all bug reports are related to software problems, but in some cases bug, reports correspond to feature requests also. A bug classification technique using the program slicing metrics have been proposed by Pan *et al*. [15]. An iterative pattern mining of software behaviors is performed by Lo *et al.* [16]. A semi-automated approach is proposed by Fluri *et al*. [17] to discover patterns of source code change types using agglomerative hierarchical clustering. Various discriminative models for information retrieval are proposed by C. Sun *et al*. [18] to detect duplicate bug reports in a software bug repository. Wang *et al*. [19], proposed identification of semantically similar terms in the bug reports, for detecting the duplicate bugs using WordNet database. A tool named CP-Miner is proposed by Li *et al*. [20], to efficiently identify copy-pasted code in large software and detect copy-paste bugs. Jalbert and Weimer [21] have proposed a system that automatically identifies the duplicate bug reports in the software bug repositories. A failure analysis of the Java Virtual Machine (JVM) is performed by Cotroneo *et al*. [22]. Factors, affecting the bug fixing in Windows Vista, and Windows 7 are explored by Guo *et al*. [23].

A method of improving the bug search technique is proposed by Williams *et al*. [24]. The comparison of various distance functions and similarity measurement techniques for text document clustering is studied by Huang [25]. Some of the problems of text clustering like high dimensionality of the data, large size of the databases and understandability of the cluster description, are analyzed and studied by Beil *et al*. [26]. An algorithm, Lingo is proposed by Osinski *et al*. [27] for clustering search results of text documents, Lingo uses algebraic transformations of the term-document matrix, and frequent phrase extraction using suffix arrays. Grouper [28] [29] is another snippet-based clustering engine. The main feature of Grouper is a phrase-analysis algorithm, called STC (Suffix Tree Clustering). STC is also one of the popular text clustering algorithms, which analyzes the text documents and extract the appropriate phrases for performing clustering of documents. Stefanowski and Weiss [30] proposed a dimension reduction algorithm; based on singular value decomposition (SVD) for text clustering.

Since most of the software, attributes are textual for software bugs, text mining technique can be applied for bug mining. Text clustering technique is explored in this work for software bug mining and a new technique for software bug clustering is proposed. Further using the bug clustering information bug classification is also performed and numbers of parameters are evaluated for the designed technique.

2. Methodology

The overall methodology of software bug clustering algorithm is shown in **Figure 1**, and is divided into the following steps: The first step includes retrieving the random software bugs from online software bug repositories, parsing the software bugs and saving to the local database. Software bugs are available in the format of HTML (Hyper Text Markup Language) or XML (Extensible Query Language) files, which needs to be parsed for attributes of the software bugs. After parsing the software bug reports retrieved from online repositories, local database schemas are defined to store the software bugs locally in the local database. Once a software bug record

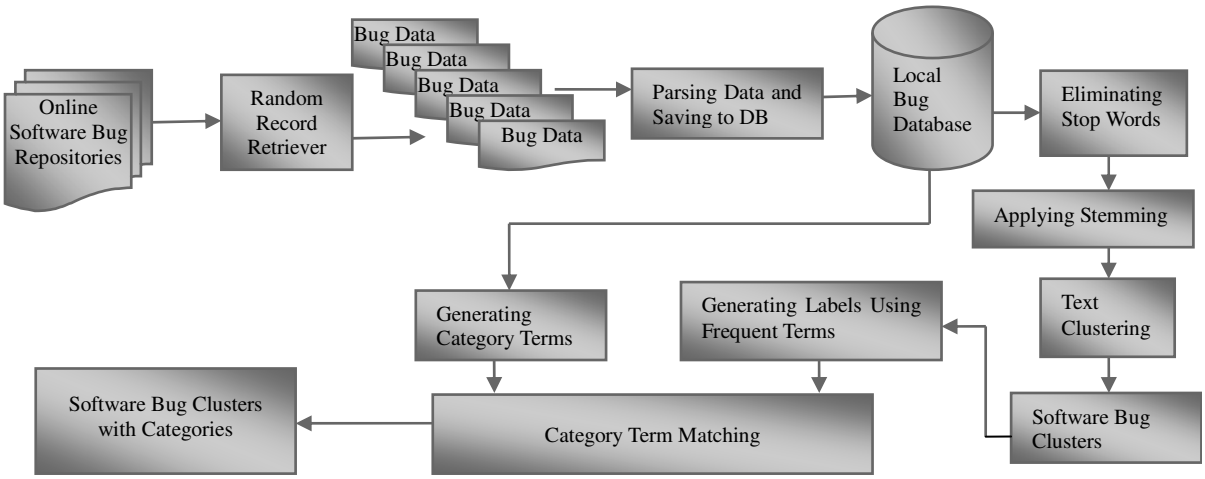

Figure 1. Major steps in the CLUBAS algorithm.

is retrieved at a local database and is available for performing data mining operation, it is transformed into the terms of a java object to enable its storage and further processing in the java collection API (Application Programming Interface). In the next step, the text mining pre-processing techniques like stop word elimination and stemming are performed to pre-process the software bug records. Stop words (useless words) do not make any sense in knowledge discovery and hence need to be eliminated. Stemming is required in order to unify the terms present in a text document, so that knowledge patterns can be discovered effectively.

The processed bug data is fed for the clustering, where bug clusters are created using weighted textual similarity of bug attributes. After creating the bug clusters, the cluster labels are generated for each cluster. The labels are generated using the frequent terms present in the pre-processed bug data belonging to a particular bug cluster. Stop words for cluster labels are identified and eliminated while generating the cluster labels. The next step is to map the terms present in the cluster labels to the software bug categories using the taxonomic terms identified for bug category, in order to classify the bug clusters. The final step is to generate the confusion matrix for the classified bug clusters, using which the performance parameters like precision, recall, F-measure and accuracy are calculated. At last the cluster information (mapping of bugs to cluster labels and categories) is visually generated for the user, using the java swing components.

3. Objective of the Work and Applications

The objective of the proposed work is to create the group of similar software bugs and then classify this group using discriminative terms identified from various software bug repositories. The application of the proposed work is to provide the effective management of the bug information and faster resolution of the reported bugs. Since the categorization is performed using the software bug clusters, which is a group of similar software bug, the project managers can use this analysis for numbers of managerial tasks. Some examples of such tasks are:

1) Creating the groups of similar set of software bugs;

2) The groups of developers can be identified for a group of categorized bugs. Similar new bugs can be assigned to the same group of developers for optimizing the fixing time;

3) Bug clusters (or categories) can be mapped into software modules, using which the module complexity analysis can be achieved by counting the bugs for each module;

4) Category wise bug distribution can be performed, which can help managers to understand the strength and weakness of the developers and software modules.

4. The CLUBAS Algorithm

In this section, the pseudo code and working of the CLUBAS algorithm is presented. CLUBAS is segmented into the five major steps. CLUBAS takes two parameter for performing the bug classification *i.e.* textual similarity threshold value (τ) and number of frequent terms in cluster label (**N**). The initial step in the CLUBAS is *Extract Data*, where the bug records from a particular bug repository is retrieved and stored in the local system. The bug record selection can be made in multiple ways e.g. randomly (generating a random number set for bug-id's), by specifying the bug-id range (minimum bug-id and maximum bug-id) or explicitly storing the bug-id's in a text file. One bug-id is selected at a time and appended to the URL (Uniform Resource Locator) of the software bug repository, and using URL network programming in java the specified bug record is fetched to the local system in the HTML (Hyper Text Markup Language) or XML (Extendible Markup Language) file format.

The next step in CLUBAS is *Pre-Processing Step*, where the software bug records available locally in HTML or XML file formats are parsed and bug attributes and their corresponding values are stored in the local database. After this the stop words elimination and stemming is performed over the textual bug attributes summary (title) and description, which are used for creating the bug clusters. In the following step (*Clustering)*, the pre-processed software bug attributes are selected for textual similarity measurement. Cosine similarity technique is used for measuring the weighted similarity between a pair of software bugs. The java based open source API, symmetric [31], which provides the implementation of various textual similarity implementation is used here for calculating the textual similarities. For all software bug pairs the weighted similarities are calculated and stored, using which the clusters are created. The clusters are created as follows—initially one cluster is created with a random bug, then if the similarity value for a paired bugs with this bug is less than the similarity threshold value (τ), then both of the bugs are mapped to the same clusters, otherwise the new cluster is created and the bug which is not belonging to the cluster is mapped to the new cluster. This similarity threshold value is one of the important parameters for the CLUBAS algorithm. If the value of τ (similarity threshold value) is the high, then high similarity between the software bug attributes is expected for clustering and vice-versa.

The next step (*Cluster Label Generation*) is to generate the cluster labels using the frequent terms present in the bugs of a cluster. In this step the summary (title) and descriptions of all the software bugs belonging to a particular clusters are aggregated and frequent terms present in this aggregate text data is calculated and the **N** (where **N** is the number of frequent terms in labels and is an user

supplied parameter) top most frequent terms are assigned to the clusters as the cluster labels. Mapping of the cluster labels to the bug categories using the taxonomic terms for various categories is carried out next (*Mapping Clusters to Classes*). In this step, the taxonomic terms for the entire bug categories are pre-identified and cluster label terms are matched with these terms. Matching of the terms indicates the belongingness of clusters to the categories. The last step (*Performance Evaluation and Output Representation*) is generating the confusion matrix, using which various performance parameters like precision, recall, and accuracy is calculated. The precision and recall can be combined together to calculate F-measure, the formulas for these parameters is mentioned in the next section. Finally the cluster information is visualized and represented as the output of the CLUBAS.

ALGORITHM CLUBAS

Returns: Clusters consisting of similar bugs
Category of each cluster
Arguments: τ—Similarity threshold
N—Number of frequent terms in cluster labels

Step 0 (*Extract Data*):
0*a*. Generate the numbers set R for bug data sources (bug-id range, randomly etc.);
0*b*. For each number $m \in R$, append it to the bug repository URL;
0*c*. Using URL programming, extract the HTML or XML page for the bug with the bug-id value as m.

Step 1 (*Pre-Processing Step*):
for-each bug record retrieved from the bug repository:
1*a*. Parse and extract the bug attributes from each bug file;
1*b*. Eliminate the stop words from bug summary, description and comments;
1*c*. Apply stemming to the textual attributes bug summary, description and comments.

Step 2 (*Clustering*):
2*a*. For each pair of bugs B_i and B_j, calculate the textual similarity between the attributes summary and description, using the similarity weights W_S and W_D such that the similarity value is normalized to 1, *i.e.* $W_S + W_D = 1$;
2*b*. $Sim(B_i, B_j) = W_S * Sim(B_{i\text{-summary}}, B_{j\text{-summary}}) + W_D * Sim(B_{i\text{-description}}, B_{j\text{-description}})$;
2*c*. IF $Sim(B_i, B_j) > \tau$ THEN Assign B_i, B_j to same cluster ELSE Create a new cluster and Assign B_j to this cluster.

Step 3 (*Cluster Label Generation—Using Frequent Terms for a Cluster*):
For each cluster C_i, get the lists of bugs belonging to this cluster:
3*a*. Extract the summary and description of these bugs;
3*b*. Concatenate this textual data to form the cluster text data;
3*c*. Calculate the N frequent terms $\{T_{i1}, T_{i2},\ldots,T_{iN}\}$ from each cluster text data, and assign them to these clusters as cluster labels;
3*d*. $Label(C_i) \leftarrow \{T_{i1}, T_{i2},\ldots,T_{iN}\}$.

Step 4 (*Mapping Clusters to Classes*):
4*a*. For each cluster C_i, get each term T_{iK} in the $Label(C_i)$ (cluster label) and match it with the bug taxonomic terms. The match indicates the belongingness of cluster in that bug category.

Step 5 (*Performance Evaluation and Output Representation*):
5*a*. Generate the confusion matrix;
5*b*. Calculation of the performance parameters Accuracy, Precision, Recall and F-Measure;
5*c*. Visualized representation of bug clusters and its labels.

5. 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 done by a classifier. In this work, the commonly used performance measures: accuracy and F-measure are used to evaluate and compare the algorithms. These measures are derived from the confusion matrix, which is shown in the **Figure 2**.

Where 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.

5.1. Accuracy

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 Equation (1) or Equation (2), given as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$\text{Accuracy}(\%) = \frac{\text{Correctly Classifies Software Bugs}}{\text{Total Software Bugs}} * 100 \quad (2)$$

		Predicted	
		Positives	Negatives
Actual	Positives	TP	FN
	Negatives	FP	TN

Figure 2. The confusion matrix.

5.2. Precision

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 Equation (3).

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \tag{3}$$

5.3. Recall

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 Equation (4).

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{4}$$

5.4. F-Measure

F_1 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 derived from F_β-measure, β being the weight factor which gives β times as much importance to recall as precision. Generally the value of β is taken as 1. The higher value of F-measure indicates higher quality of the classifiers. F_β-measure is calculated using Equation (5), whereas F-measure or F_1-measure is calculated using Equation (6).

$$\text{F}_\beta = \left(1+\beta^2\right) * \frac{\text{precision} * \text{recall}}{\left(\beta^2 * \text{precision}\right) + \text{recall}} \tag{5}$$

$$\text{F}_1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{6}$$

6. Implementation

Implementation is done using open source object oriented programming language Java [32], and MySql [33] is taken as local data base management system, Weka [34] API is used for implementing the stemming and other classification algorithms for comparison. The multi map data structure is also used for calculations and storing the clusters information at run time.

6.1. Datasets and Sampling

The random software bug records are selected from four open sources online software bug repositories namely, Android [35], JBoss-Seam [36], Mozilla [37] and MySql [38]. Random sampling technique is used and the sample size of 200, 300, 500, 700, 1000, 1300, 1600 and 2000 is taken for the experiments from these four repositories for the comparison of the classifiers. To retrieve the random records from the mentioned software bug repositories, a random number generator source code 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 MySql bug repositories [38], "http://bugs.mysql.com/bug.php?id=", "id" can be appended to retrieve a particular software bug record.)

6.2. Pre-Processing

After the software bug records are extracted and made available at local system, and then pre-processing of these records is performed. The pre-processing takes places in three stages: parsing, elimination of stop words and stemming. In parsing phase all the software bug attributes position is detected and their corresponding values are parsed and stored to the local database schema for all of the software bug repositories. Once these parsed values are available the stopping and stemming is performed on the textual attribute values. In stopping the first step is to identify the suitable stop list, which consists of the terms (words) not relevant for the classification of the software bugs. Terms with numeral and special characters are also eliminated in the process of stopping. However, the terms like "not" etc. are not removed during stopping, since such terms are relevant for software bug classification [14]. Finally, stemming is performed over the filtered values, where in stemming words are reduced to their root form (origin words), which can be performed using suffix and affix removal. In the present work Porter's stemming algorithm is used for stemming, which is a part of Weka [34] API (Application Programming Interface).

6.3. Mapping Bug Clusters to Categories

The categorical terms are generated from the software bug clusters labels. The **Table 1** is generated for 2000 random sample software bugs selected from the four open source bug software projects in aggregation. **Table 1** is consisting of 10 major categories of software bugs with their corresponding pre-processed (after stopping and stemming) taxonomic terms. Terms which are not covered in the mentioned categories are treated as the

Table 1. Category discriminative terms.

Category Name	Categorical Terms
Logical	assertion, annotation, asynch, argument, application, attempting, break, broken, behavior, badly, call , caus, code, clustering, component , core, default, doesn, error, exception, edit, found, fail, file, frame, handle, host, implement, incorrect, integrat, incomplete, java, librar, logic, miss, mension, work, npe, null, option, propert, pointer, parameter, pluggable, problem, portability, redirect, remove, read, replica, run, repea, server, session, submit, search, statement, status, service, start, throw, validat, wrong, not, proper, should
GUI	button, border ,background , blank ,bundle, css, container, captcha, display, event, font, html, item, imag, list, label, line, layout, locale, method, message, navigator, pixel ,page, render, resource , space , selection, show, tag, toolbar, typo, click, mouse, key, resolution, table
Datatype	array, blob, binary, char, numeric, hard, real, string, text, utf, undef, variable, value
Backend	attribute, column, connection, data, db, field, insert, index, join, load, query, record , store, serial, update
Enhancement	add, enhance, ignore, improve, optimi, performance, required, support, can, may, suggest
Build	ant, build, compile, config, debug, log, make, module, patch, redeploy, syntax, warnings
OS	concurrent, path, redhat, unix, windows
Security	access, admin, grant, privileges, roles, revoke, resultset, security, user
Memory	cache, crash, flow, fault, heap, infinite, memory, segmentation, segfault, smart, threads, profil
Analysis	case, comment, diagram, doc, fig, tutorial, test , unit

non-bug terms. The technique of generating these taxonomic terms from various bug repositories is given in [39]. 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

7. Experimentation and Comparative Analysis

The experiments was performed on four different software bug repositories, using Java and Weka data mining API and the classifiers are evaluated using 10-fold cross-validation technique. The proposed algorithm CLUBAS is compared with other standard classification algorithms for classifying the software bugs. CLUBAS is compared with Naïve Bayes (NB) [40,41], Naïve Bayes Multinomial [41], J48 [42], Support Vector Machine (SVM) [43,44] and Weka's Classification using Clustering (CC) [45] algorithms. LIBSVM [46] is an open source implementation of SVM, which can be integrated into Weka. The comparison is performed on the basis of two standard classification parameters F-measure (which is the combine measure of precision and recall) and accuracy of the classifiers.

The algorithms are first applied on the Android bug repository and the results for accuracy and F-measures for different number of samples are plotted in **Figures 3(a)** and **(b)** respectively. From the plot **Figure 3(a)**, it is observed that both NB and NBM performs better in terms of accuracy than the other algorithms and CLUBAS does not perform well accuracy wise, still it is able to maintain more than 80% accuracy and at certain points it performs better than the NBM algorithm. But from the F-measure point of view (**Figure 3(b)**), the algorithm CLUBAS shows stability and performs better than CC, SVM and J48, it maintains the F-measure values more than 0.9 for each experiment using different number of samples.

Similarly, the experiments are performed on JBoss-Seam, Mozilla and MySql bug repositories using the six algorithms. The accuracy and F-measure results for these repositories are plotted against the different number of samples in **Figures 4(a)** and **(b)**, **Figures 5(a)** and **(b)** and **Figures 6(a)** and **(b)** respectively. It is observed from the plots that Accuracy wise CLUBAS just performs well and maintains more than 80% accuracy for different samples in all of the repositories, and in case of MySql bug repository, only NB performs better CLUBAS algorithm. From the F-measure point of view it gives the stable results, irrespective of the number of samples in classification and mostly it maintains more than 0.9 values for all the repositories. In case of F-measure, for all of the repositories except MySql, only NB and NBM perform better than CLUBAS and in case of MySql repositories, CLUBAS gives better results than any other algorithm.

For larger F-measure, values CLUBAS algorithm provide consistent results. However the runtimes for each algorithm were noted and it was found that all of the algorithms took roughly 2 to 3 minutes for a single experiment up to 2000 random samples. So it can be concluded from the experiments and comparison that, when the precision, recall and F-measures are important CLUBAS gives the better and stable results irrespective of number of samples and software bug repositories. CLUBAS is also able to maintain more than 80% accuracy for the different number of samples from selected four bug repositories. In overall, it is also better than the other similar algorithm, *i.e*. Weka's CC algorithm in terms of accuracy and F-measure.

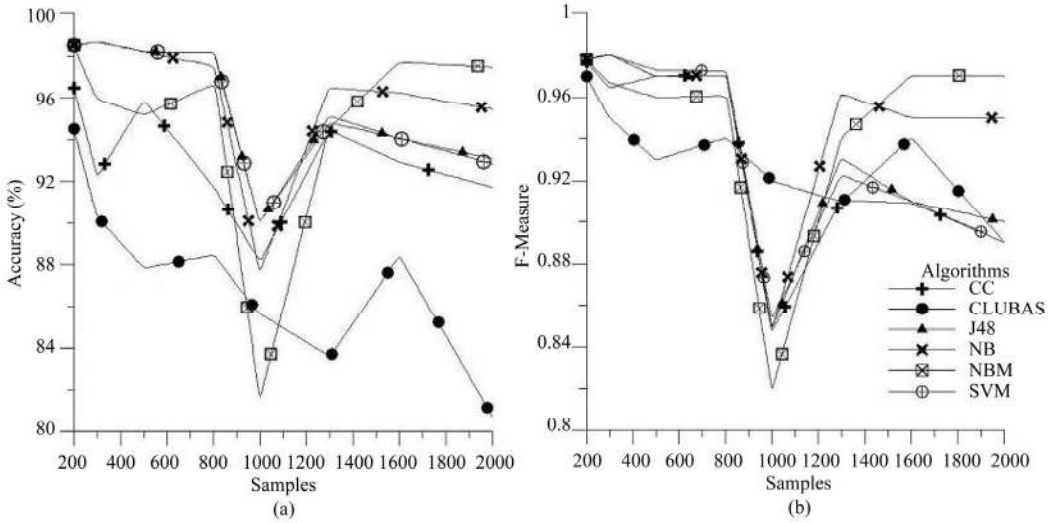

Figure 3. Performance of various classifiers over Android bug repository (a) Accuracy; (b) F-measures.

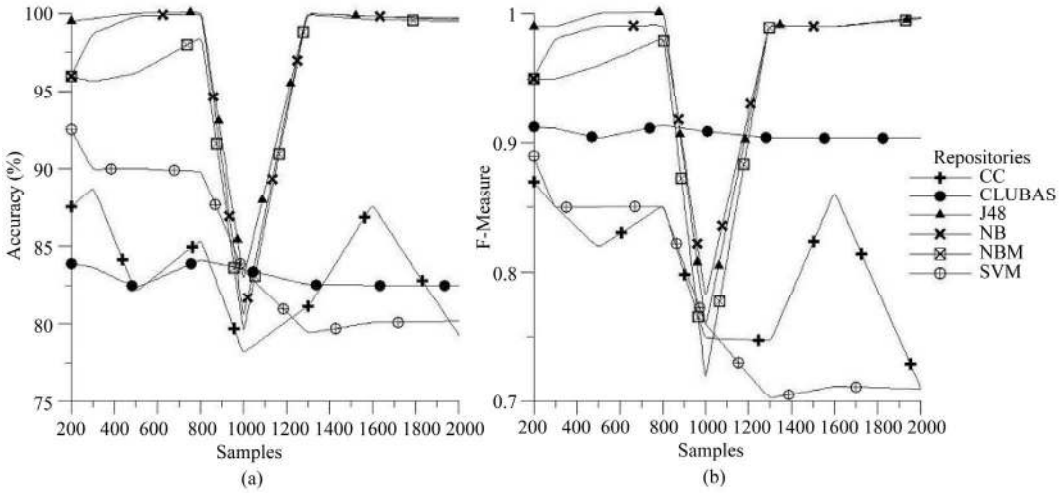

Figure 4. Performance of various classifiers over JBoss-Seam bug repository (a) Accuracy; (b) F-measures.

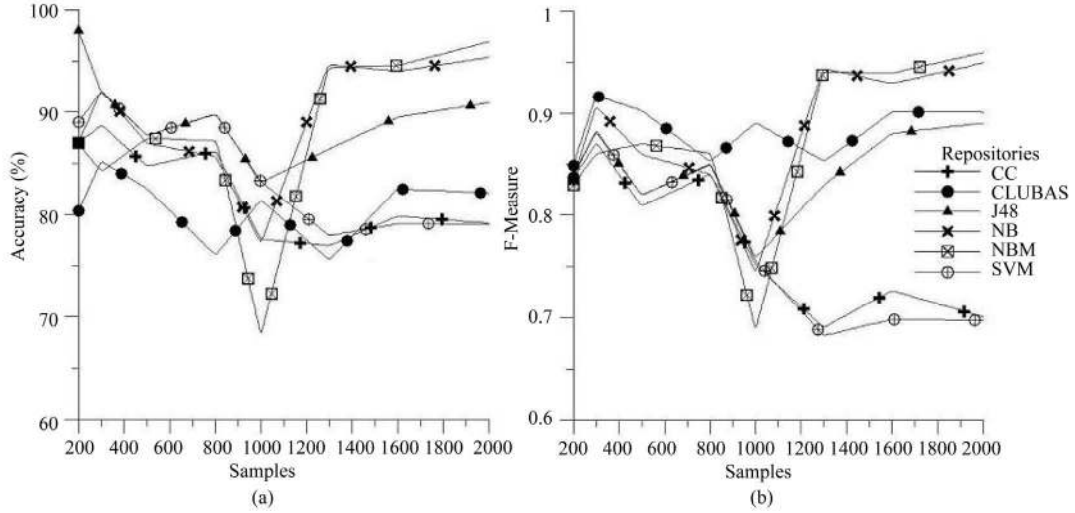

Figure 5. Performance of various classifiers over Mozilla bug repository (a) Accuracy; (b) F-measures.

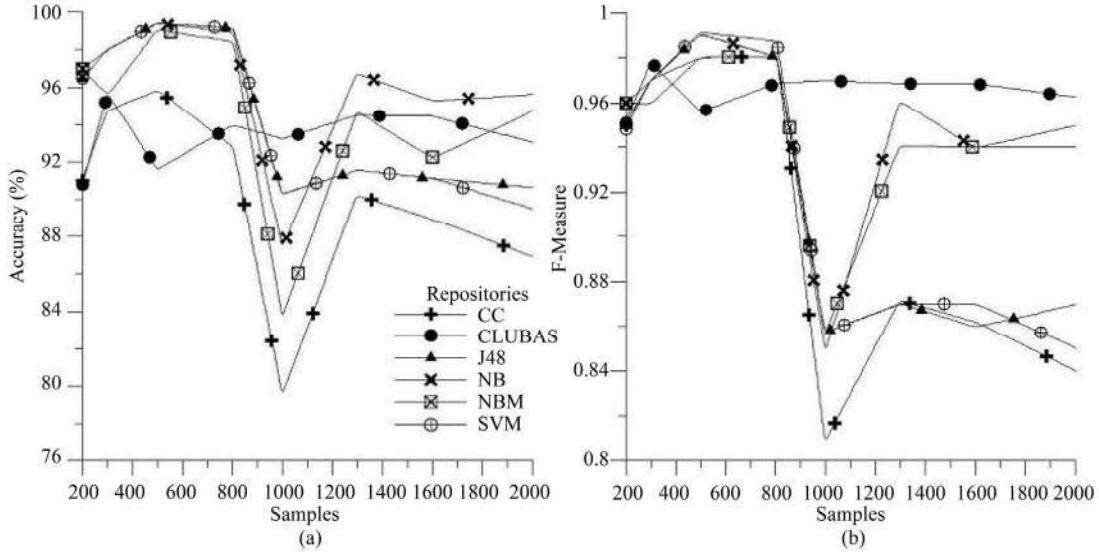

Figure 6. Performance of various classifiers over Mysql bug repository (a) Accuracy; (b) F-measures.

7.1. Effect of Similarity Threshold (τ)

The effect of similarity threshold over accuracy and F-measure for CLUBAS is plotted in **Figures 7(a)** and **(b)** respectively. 1000 random samples are taken from all the four bug repositories to identify the effect of similarity threshold value for CLUBAS. For JBoss-Seam and MySql bug repositories the parameters accuracy and F-measure values are stable, irrespective of the similarity threshold, however for Android and Mozilla bug repositories the same parameter decrease, with increase in the similarity threshold value in the algorith CLUBAS. This indicates that the textual similarity in software bug information stored in the Android and Mozilla is less than the other two bug repositories.

7.2. Effect of Frequent Terms in Cluster Labels (N)

The relationship between number of frequent terms in class label and accuracy/F-measure is plotted in **Figures 8(a)** and **(b)**, respectively. It is oserved from the plot that, as the number of frequent terms increses in the cluster labels the accuracy and F-measures also increases for all the bug repositories, and after a certain point (*i.e.* number of frequent terms is 10) the accuracy and F-measures becomes stable and are freed from the effect of number of frequent terms in CLUBAS. This point indicates the optimum number of terms in cluster label, which is identified to be 10 in this experiment for all the four repositories *i.e.* Android, JBoss-Seam, Mozilla and MySql with 1000 random samples and similarity threshold of 0.01.

8. GUI Based CLUBAS Tool

The proposed algorithm is implemented in java and a GUI based tool is created. Some screen shots are shown in this section as an illustration for the implemented tool. The GUI for selecting the various parameters for CLUBAS is shown in **Figure 9**. User can select the textual similarity technique and also can specify the similarity threshold value using this interface. User can also specify the name of output file, where the output of the algorithm along with various calculated parameters can be stored. Another GUI interface for displaying the output of the CLUBAS is shown in **Figure 10**. This interface is mainly consisting of three lists. The first list holds the cluster created with the labels assigned to it, second list shows the bugs belonging to the cluster selected in the first list, and the last list shows the category term for the selected cluster. Various list selection listeners are implemented in **Figure 10**, if user selects a particular cluster (shown as cluster-id: cluster-label) on the leftmost list, its corresponding software bugs and categories will be displayed on the other lists.

9. Threats to Validity

Four software bug's repositories namely, Android, JBoss-Seam, Mozilla and MySql are selected for experimentations and comparison in this present work. Different numbers of random samples are selected from every repository for validation of the work. Although the experiment is performed multiple number of times and averaging is also done for the results, there is a chance that the calculated parameter values may differ for other samples, where the textual bug attribute information is poor (the bug information is not described in details). The other limitation of the work can be derived from the Zipf's power distribution law [47,48]. It states that most of users use limited number of words (terms) frequently in the

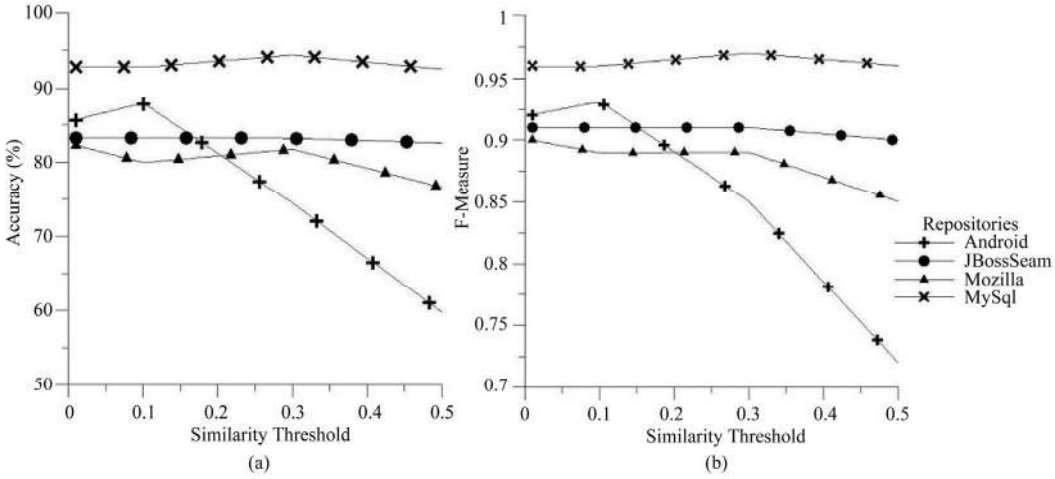

Figure 7. Effect of similarity threshold in CLUBAS (a) Accuracy; (b) F-measures.

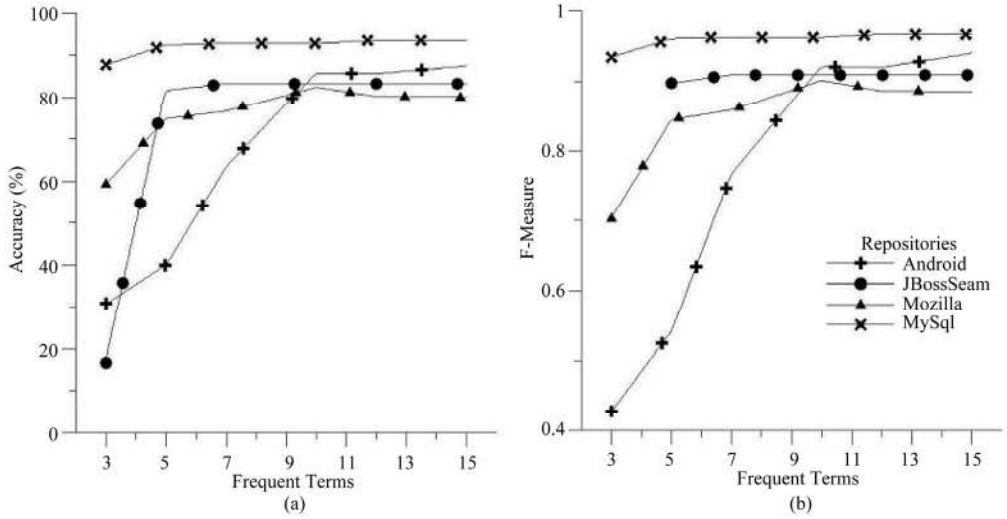

Figure 8. Effect of frequent terms in CLUBAS (a) Accuracy; (b) F-measures.

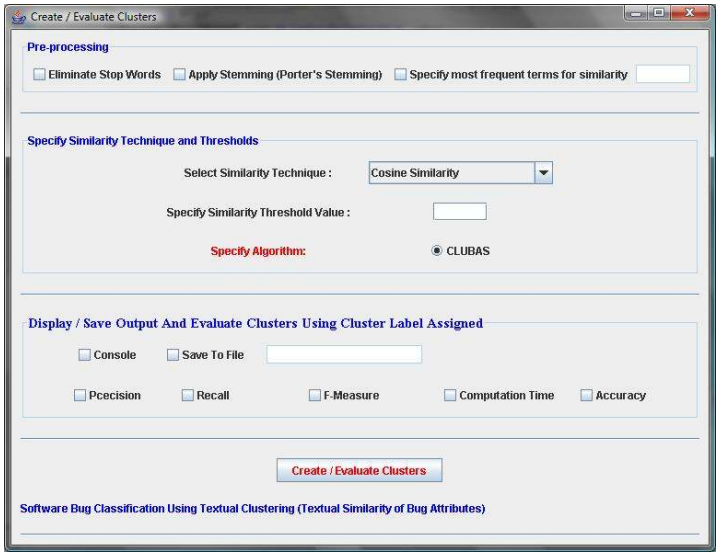

Figure 9. GUI snapshot for specifying various parameters in CLUBAS.

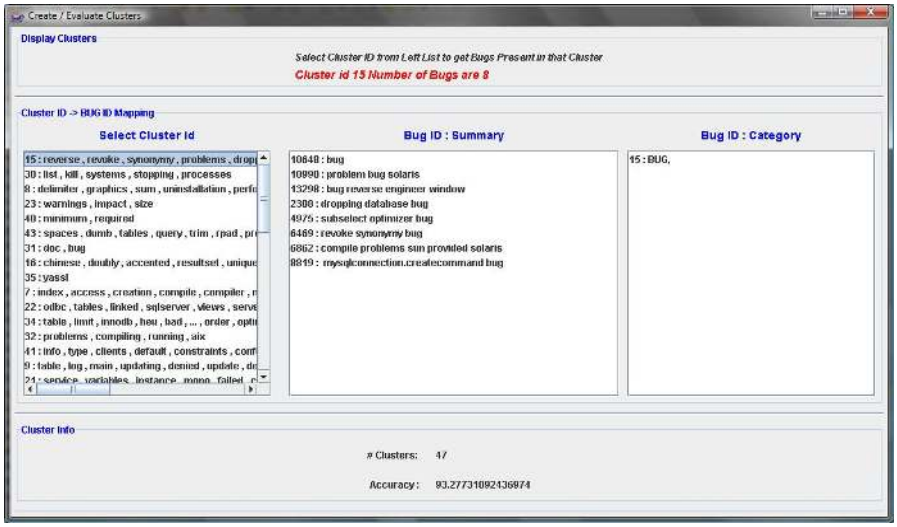

Figure 10. GUI screen for cluster output.

documents. On one end the Zipf's law supports the algorithm CLUBAS, since it is also derived from the frequent terms, however on the other end in few cases where the developers from different places are working and using the different set of vocabularies to represents the bug information, the accuracy values may drop.

10. Conclusion & Future Scope

In this paper, a text clustering and classification algorithm is developed and a GUI based tool for software bug classification CLUBAS is presented. The algorithm CLUBAS is designed using the technique of classification by clustering, in which first clustering is done using textual similarity of bug attributes and then proper labels are generated and assigned to each cluster. The cluster labels are further mapped to the bug classes using the cluster label and bug taxonomic terms matching. The algorithm uses two input parameters; similarity threshold value and number of terms in cluster label. The effect of these parameters for performance evaluation is also studied and graphs are plotted to visualize their effects. The algorithm CLUBAS is compared with number of standard classification algorithms for performance evaluation. The comparison is performed using the performance parameters; accuracy and F-measure (combined measure of precision and recall). From the experiments it is observed that CLUBAS is able to maintain more than 80% accuracy for all the bug repositories at different sampling points (number of samples), and always gives more than 0.9 as F-measure. From comparative analysis, it is found that accuracy wise, only algorithms NB and NBM performs better than CLUBAS, however F-measure wise, it the best algorithm, since it gives stable and higher values of F-measure, irrespective of the bug repositories and number of samples in classification. The future scope related to the proposed work can be applying advanced text pre-processing techniques for optimizing the clustering and classification work, and also modern text clustering and classification algorithms can be implemented and compared with the proposed algorithm. The validity of the Zipf's law can also be verified in the future work.

REFERENCES

[1] http://www.bugzilla.org

[2] http://www.perforce.com

[3] http://www.atlassian.com/software/jira/

[4] http://trac.edgewall.org

[5] IEEE Standard, Classification for Software Anomalies Working Group (IEEE 1044 WG Std), No. 1044, 1993, pp. 1-15.

[6] IEEE Standard, Classification for Software Anomalies Working Group (IEEE 1044 WG Std), No. 1044, (Revision), 2009, pp. 1-15.

[7] H. Kagdi and D. Poshyvanyk, "Who Can Help Me with This Change Request?" *IEEE International Conference on Program Comprehension*, Vancouver, 17-19 May 2009, pp. 273-277. doi:10.1109/ICPC.2009.5090056

[8] J. Anvik and G. C. Murphy, "Determining Implementation Expertise from Bug Reports," 4*th International Workshop on Mining Software Repositories*, Minneapolis, 20-26 May 2007, pp. 9-16. doi:10.1109/MSR.2007.7

[9] 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 and Evolution*: *Research and Practice*, Vol. 19, No. 2, 2007, pp. 77-131. doi:10.1002/smr.344

[10] H. Kagdi, M. Hammad and J. I. Maletic, "Who Can Help Me with this Source Code Change?" *IEEE International Conference on Software Maintenance*, Beijing, 28 September-4 October 2008, pp. 157-166.

doi:10.1109/ICSM.2008.4658064

[11] N. Ayewah and W. Pugh, "Learning from Defect Removals," 6*th IEEE International Working Conference on Mining Software Repositories*, Vancouver, Canada, 16-17 May 2009, pp. 179-182. doi:10.1109/MSR.2009.5069500

[12] S. Kim, E. J. Whitehead Jr. and Y. Zhang, "Classifying Software Changes: Clean or Buggy?" *IEEE Transactions on Software Engineering*, Vol. 34, No. 2, 2008, pp. 181-196. doi:10.1109/TSE.2007.70773

[13] G. Vijayaraghavan and C. Kaner, "Bug Taxonomies: Use Them to Generate Better Tests," *Software Testing Analysis and Review Conference* (*STAR EAST* 2003), Orlando, 2003, pp. 1-40.

[14] G. Antoniol, K. Ayari, M. D. Penta, F. Khomh and Y. G. Guéhéneuc, "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*, New York, 2008, pp. 304-318.

[15] K. Pan, S. Kim and E. J. Whitehead Jr., "Bug Classification Using Program Slicing Metrics," 6*th IEEE International Workshop on Source Code Analysis and Manipulation* (*SCAM*), Philadelphia, 2006, pp. 31-42. doi:10.1109/SCAM.2006.6

[16] D. Lo, H. Cheng, J. W. Han, S. C. Khoo and C. N. Sun, "Classification of Software Behaviors for Failure Detection: A Discriminative Pattern Mining Approach," *ACM Knowledge Discovery in Databases*, Paris, 2009, pp. 557-565.

[17] B. Fluri, E. Giger and H. C. Gall, "Discovering Patterns of Change Types," *Proceedings of the* 23*rd International Conference on Automated Software Engineering* (*ASE*), L'Aquila, 15-19 September 2008, pp. 463-466.

[18] C. N. Sun, D. Lo, X. Y. Wang, J. Jiang and S. C. Khoo, "A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval," *ACM International Conference on Software Engineering*, Cape Town, 1-8 May 2010, pp. 45-54.

[19] X. Y. Wang, L. Zhang, T. Xie, J. Anvik and J. Sun, "An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information," *ACM International Conference Software Engineering*, Leipzig, 10-18 May 2008, pp. 461-470.

[20] Z. Li, S. Lu, S. Myagmar and Y. Zhou, "CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code," *IEEE Transactions on Software Engineering*, Vol. 32, No. 3, 2006, pp. 176-192.

[21] N. Jalbert and W. Weimer, "Automated Duplicate Detection for Bug Tracking Systems," *IEEE International Conference on Dependable Systems & Networks*, Anchorage, 24-27 June 2008, pp. 52-61.

[22] D. Cotroneo, S. Orlando and S. Russo, "Failure Classification and Analysis of the Java Virtual Machine," *Proceedings of the* 26*th IEEE International Conference on Distributed Computing Systems*, Lisboa, 4-7 July 2006, pp. 1-10.

[23] P. J. Guo, T. Zimmermann, N. Nagappan and B. Murphy, "Characterizing and Predicting which Bugs Get Fixed: An Empirical Study of Microsoft Windows," *ACM International Conference on Software Engineering*, Cape Town, 1-8 May 2010, pp. 495-504.

[24] C. C. Williams and J. K. Hollingsworth, "Automatic Mining of Source Code Repositories to Improve Bug Finding Techniques," *IEEE Transactions on Software Engineering*, Vol. 31, No. 6, 2005, pp. 466-480. doi:10.1109/TSE.2005.63

[25] A. Huang, "Similarity Measures for Text Document Clustering," 6*th New Zealand Computer Science Research Student Conference*, Christchurch, 14-18 April 2008, pp. 49-56.

[26] F. Beil, M. Ester and X. Xu, "Frequent Term-Based Text Clustering," *ACM Special Interest Group—Knowledge Discovery in Databases*, Edmonton, 23-25 July 2002, pp. 436-442.

[27] S. Osinski, J. Stefanowski and D. Weiss, "Lingo: Search Results Clustering Algorithm Based on Singular Value Decomposition," *Proceedings of the Springer International Intelligent Information Processing and Web Mining Conference*, Zakopane, 17-20 May 2004, pp. 359-368.

[28] O. Zamir and O. Etzioni, "Grouper: A Dynamic Clustering Interface for Web Search Results," *Computer Networks*, Vol. 31, No. 11-16, 1999, pp. 1361-1374. doi:10.1016/S1389-1286(99)00054-7

[29] O. Zamir and O. Etzioni, "Web Document Clustering: A Feasibility Demonstration," *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval* (*SIGIR*), Melbourne, 1998, pp. 46-54. doi:10.1145/290941.290956

[30] J. Stefanowski and D. Weiss, "Comprehensible and Accurate Cluster Labels in Text Clustering," *Proceedings of the* 8*th Conference on Information Retrieval and Its Applications*, Pittsburgh, 2007, pp. 198-209.

[31] http://www.dcs.shef.ac.uk /~sam/simmetrics.html

[32] http://www.java.com/

[33] http://mysql.com

[34] www.cs.waikato.ac.nz/ml/weka

[35] http://code.google.com/p/android/issues

[36] https://issues.jboss.org/browse/JBSEAM

[37] https://bugzilla.mozilla.org

[38] http://bugs.mysql.com

[39] N. K. Nagwani and S. Verma, "A Frequent Term Based Approach for Generating Discriminative Terms in Software Bug Repositories," *IEEE* 1*st International Conference on Recent Advances in Information Technology*, Dhanbad, 15-17 March 2012, pp. 433-435.

[40] G. H. John and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," *Proceedings of* 11*th Conference on Uncertainty in Artificial Intelligence*, San Mateo, 18-20 August 1995, pp. 338-345.

[41] A. Mccallum and K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification," *Proceedings of AAAI*-98 *Workshop on Learning for Text Categorization*, Madison, 26-27 July 1998, pp. 41-48.

[42] R. Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufmann Publishers, San Mateo, 1993.

[43] V. N. Vapnik, "The Nature of Statistical Learning Theory," Springer-Verlag, New York, 1995.

[44] http://www.cs.iastate.edu/~yasser/wlsvm/

[45] A. Kyriakopoulou and T. Kalamboukis, "Text Classification Using Clustering," *Proceedings of ECML-PKDD*, *Discovery Challenge Workshop*, Burlin, 18-22 September 2006, pp. 28-38.

[46] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[47] W. Li, "Random Texts Exhibit Zipf's-Law-Like Word Frequency Distribution," *IEEE Transactions on Information Theory*, Vol. 38, No. 6, 1992, pp. 1842-1845. doi:10.1109/18.165464

[48] W. J. Reed, "The Pareto, Zipf and Other Power Laws," *Economics Letters*, Vol. 74, No. 1, 2001, pp. 15-19. doi:10.1016/S0165-1765(01)00524-9

