An Empirical Comparison of Machine Learning Techniques in Predicting the Bug Severity of Open and Closed Source Projects

K. K. Chaturvedi, Indian Agricultural Statistics Research Institute, New Delhi, Delhi, India

V.B. Singh, Delhi College of Arts & Commerce, University of Delhi, New Delhi, Delhi, India

ABSTRACT

Bug severity is the degree of impact that a defect has on the development or operation of a component or system, and can be classified into different levels based on their impact on the system. Identification of severity level can be useful for bug triager in allocating the bug to the concerned bug fixer. Various researchers have attempted text mining techniques in predicting the severity of bugs, detection of duplicate bug reports and assignment of bugs to suitable fixer for its fix. In this paper, an attempt has been made to compare the performance of different machine learning techniques namely Support vector machine (SVM), probability based Naïve Bayes (NB), Decision Tree based J48 (A Java implementation of C4.5), rule based Repeated Incremental Pruning to Produce Error Reduction (RIPPER) and Random Forests (RF) learners in predicting the severity level (1 to 5) of a reported bug by analyzing the summary or short description of the bug reports. The bug report data has been taken from NASA's PITS (Projects and Issue Tracking System) datasets as closed source and components of Eclipse, Mozilla & GNOME datasets as open source projects. The analysis has been carried out in RapidMiner and STATISTICA data mining tools. The authors measured the performance of different machine learning techniques by considering (i) the value of accuracy and F-Measure for all severity level and (ii) number of best cases at different threshold level of accuracy and F-Measure.

Keywords: *10-fold Cross Validation, Bug Repositories, Bug Severity, Multiclass Classification, Supervised Classification, Text Mining*

DOI: 10.4018/jossp.2013040103

INTRODUCTION

With the increasing use of software in every sphere of life, it is often found that software does not function properly or some of the functionalities need minor change for improvement. It is becoming very important to record these problems or changes using suitable bug reporting system. A bug or fault is a program defect that is encountered while operating the product either under test or in use. In order to provide the details of bug to the development team, different bug tracking systems are being used by the industry. The bug tracking systems are helpful in bug reporting as well as tracking the progress of bug fixes. Various bug tracking systems have been proposed in the available literature that includes Bugzilla (http://www.bugzilla.org/), Jira (http://www.atlassian.com/software/jira/), Mantis (http://www.mantisbt.org/), etc. Recently, a bug tracking and reliability assessment tool has been proposed (Singh & Chaturvedi, 2011), which helps in tracking the progress of fix as well as reporting the bug. Various parameters are filled up by the tester or user who reports the bug. During bug reporting, different bug attributes namely bug title, short description or summary, detailed description, priority, severity are filled up. The value of severity and priority may not be accurate as far as the submitted report is concerned because users may not have complete information about the modules/ components in which the bug has occurred. Priority of a reported bug represents the urgency of its fix; accordingly a number or level associated with priority has to be assigned. There are five different numbers or levels defined for priority. In addition to the priority, another important parameter that affects the priority of the reported bug is severity. Severity is defined as the impact of bug in working functionality of a component or the system. The impact of the bug varies from user to user. Generally, people do require that their bug must be taken on a priority basis irrespective of the impact on the user or the developer or the system itself. Software projects have clear guidelines on how to assign a severity level to a bug. But, due to non-awareness, people often commit mistakes in assigning the severity level during bug reporting. The severity level can be categorized broadly in five to seven categories. Bug repository of closed source projects of NASA's PITS (Projects and Issue Tracking System) dataset has five severity levels vary from the fullest to the dullest or from 1 to 5. Bug repositories of open source projects have defined seven severity levels, and these severity levels vary from the blocker, critical, major, enhancement, minor, normal, to trivial or from 1 to 7. Level 1 of severity represents fatal errors and crashes whereas level 5 or 7 of severity mostly represents cosmetic changes such as formatting, alignment, comments and display messages. Other severity levels are assigned for bugs due to addition of new features and enhancement of the existing feature. The default value "Normal" is normally assigned by most of the reporters because they do not analyze the bug seriously. Severity is a critical factor in deciding the priority of a reported bug. The numbers of reported bugs are usually quite high, hence, it is becoming necessary to have a tool/technique which can determine or verify the severity of a reported bug. It is the need of the hour to automate the process of determining the bug severity for the reported bug. Our work is based on the following research questions:

Research Question 1: Are the Machine Learning Techniques applicable in predicting the severity level of a reported bug?

Research Question 2 What is the order of applicability of different machine learning techniques on the basis of different performance measures?

Research Question 3: Is there any effect of number of terms on the performance of different Machine Learning Techniques?

To answer the above research questions, a study has been conducted by applying the Naïve Bayes, Decision Tree, RIPPER, Random Forest and Support Vector Machine in the bug repositories of open as well as closed source projects. The performance measures namely accuracy, precision, recall and F-Measure have been calculated using 10-fold cross validation by varying the number of terms. The paper is organized as follows:

The paper starts with related work and describes pre-processing and representation of textual data of bug reports followed by text mining and classification algorithms. Separate section mentions the selection of projects/ components and collection of reported bugs from bug repositories of open and closed source projects and discusses about classifiers building and evaluation. Results are presented using graphical representations and findings are discussed thoroughly by including the threats to validity of results. Finally the paper is concluded and provided the future research directions.

RELATED WORK

The short description or summary attribute of the reported bug contains running summarized textual information which is important and useful to the triager. This information is also useful in determining other parameters of the report such as duplicate bug report, possible fixer, severity level, priority etc. This can be analyzed by applying one of the sub areas of data mining, i.e. text mining. Text mining can be utilized to mine the knowledge from this description (Han & Kamber, 2006; Konchady, 2006; Weiss et al., 2005). SAS Text Miner was developed to generate a term-by document frequency matrix and used for predictive modeling (SAS Institute, 2004; Cerrito, 2006; Sarma, 2007). The summary or short description attribute contains useful and concise information that may be of great use while dealing with the text mining. Ko et al. (2006) discuss how people describe the software problems using linguistic analysis. If we can mine the description and gain knowledge from already submitted reports, this knowledge can be helpful in assessing the quality of bug report. The effort have been made in assessing the quality of bug reports (Bettenburg et al., 2008; Herraiz et al., 2008; Hooimeijer & Weimer, 2007) and assignments of bug to a suitable fixer. Further, Linstead and Baldi (2009) define a new information-theoretic measure of coherence i.e., Latent Dirichlet Allocation (LDA) to the problem of mining bug reports by applying statistical text mining algorithms for estimating bug report quality. Text mining techniques have been previously applied on the descriptions of bug reports to automate the bug triaging process (Anvik et al., 2006; Gaeul et al., 2009; Guo et al., 2010; Matter et al., 2009; Tamrawi et al., 2011). These techniques have been also applied to detect duplicate bug reports (Cubranic & Murphy, 2004; Runeson et al., 2007; Wang et al., 2008). Sureka and Jalote (2010) have used N-Gram technique to determine the duplicate bug report. Yu et al. (2010) predict the bug priority during software testing process using Artificial Neural Network (ANN) and Naïve Bayes classifier. Kanwal and Maqbool (2010 and 2012) compare and evaluate the Support Vector Machine (SVM) and Naïve Bayes classification techniques to prioritize the bug report based on categorical and textual attributes. They conclude that the performance of SVM is better than the Naïve Bayes with textual attributes. On the other hand, Naïve Bayes performance is better than SVM while using with categorical attributes. The automated process will save the triager's time (Bhattacharya & Neamtiu, 2010) during bug triaging.

Machine learning techniques have been used to automate the process of determining the bug priority, severity and assignee. The summary or short description given by users and testers contain useful information about the bug. Antoniol et al. (2008) have used text mining in determining the change request as a new feature or enhancement. Gegick et al. (2010) have proposed text mining approach to distinguish security bug reports from the non-security bug reports using summary and long description field. Prifti et al. (2011) use the information retrieval techniques in recently submitted bug reports to avoid the duplicate submission of bug reports. They suggest that user should submit the comments on already existing bugs instead of submitting a duplicate bug. Baysal et al. (2011) have studied the tale of two browsers (Mozilla and Chrome, a web browser from Google) based on the short version cycles, longer bug fix cycles, and a user base for adoption based on mining bug fix patterns by studying the bug reports and co-evolution. Wu et al. (2011) have developed a BugMiner tool based on historic data of bug repositories to predict the missing categorical fields of the newly submitted bug reports on the basis of the available fields. Moreover this tool finds the duplicate bugs.

In the available literature, very few efforts have been made to determine the severity of a reported bug on the basis of textual description. Firstly, Menzies and Marcus (2008) make an attempt on bug repository of closed source NASA's Pits (Projects and Issue Tracking System) dataset using Repeated Incremental Pruning to Produce Error Reduction (RIPPER), a rule based learner. They have applied the RIPPER on summary attribute of the reported bugs. They have shown that unstructured text might be a better candidate for generating severity assessment than the structured data base fields. Authors have presented an automated method named SEVERIS (SEVERity ISsue Assessment), which assists the test engineer in assigning severity levels to the reported bug. The authors have done their experiments on a limited data set and rule based technique but did not investigate the issues raised in research question 1, 2 and 3 in our paper. Further, Lamkanfi et al. (2010) attempts the severity prediction in binary classes, i.e. severe and non-severe using the Naive Bayes classification. Later on Lamkanfi et al. (2011) have extended earlier study to compare with a few other data mining algorithms such as Naive Bayes Multinomial, K-Nearest Neighbour and Support Vector Machine for classification of bug severity into binary classes. They have further shown that it is possible to predict the severity of a reported bug based on summary or short description instead of long description of a reported bug using a Naïve Bayes classifier. Recently, an attempt has been made by Chaturvedi and Singh (2012) in determining the severity of a reported bug of closed source bug repositories of NASA datasets.

To the best of our knowledge, no efforts have been made in the literature to predict the class levels of bug severity (1 to 5) using machine learning technique with 10-fold cross validation in open as well as closed source projects. In this paper, an attempt has been made to demonstrate the applicability of supervised machine learning classification techniques in predicting the severity level of the bug reports based on summary or short description using one-against-one for multi-class classification. The applicability has been validated using various performance measures namely accuracy, precision, recall and F-Measure by applying the 10-fold cross validation.

PRE-PROCESSING AND REPRESENATION OF TEXTUAL DATA

Textual data contains group of sentences or paragraphs that determine its purpose and use. We may not use these documents for any analysis as it will be quite difficult to analyze because of not having any predefined structure. This data may also contain some garbage terms, special characters, other language alphabets, sentences, etc., which will affect in extracting the features from these documents. These reports/documents are represented by applying the two stage process, i.e. pre-processing of bug reports/documents and structuring of these bug reports/documents.

Pre-Processing of Textual Data

The standard steps for pre-processing of any textual document contains following steps:

- **Tokenization:** Tokenization represents the process of converting a stream of characters into sequence of tokens. The token can be a sentence or a paragraph or a line or a word or an alphabet. Tokenization refers to separate the token from the given text. Here, we consider the token as a word or a term. We remove all punctuation marks, non-printable characters and other meaningless symbols as they may not contribute in the classification task. All capital letters are also replaced by small case letters.

- **Stop Word Removal:** Stop words are those words which are commonly used in the texts but do not carry useful meaning. A list of English stop words, i.e. prepositions, conjunctions, articles, verbs, nouns, pronouns, adverbs, adjectives, etc. can be downloaded from www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words. In our study, we have used SMART data set as a stop word list. We have also created our own user defined list as a dictionary of stop words which contains some other language characters, numbers and other special characters.
- **Stemming:** A process of converting the derived word to their base word is called stemming. The base word is known as stem. As we know that each individual term can be expressed in many forms and carry special meaning. For example, the terms "computerized", "computerize", "computarization" and "computation" share the same base stem as "computer". Instead of considering all terms as individual terms, it can be replaced by a single term. In this way, the weightage can be increased for such terms and numbers of unnecessary terms can be avoided. Standard Porter stemming algorithm (Porter, 1980) can be utilized for word stemming.
- **Feature Reduction:** Most frequently and less frequently occurring terms may be removed from the dataset. These terms/features may not be able to discriminate the documents or may lead to overfit/ underfit the model. Generally, in classification, the term which has less than three or more than forty/fifty occurrences will generally be removed from the dataset as most of the data mining algorithm may not be able to handle large feature sets.
- **Information Gain or InfoGain:** It is an entropy based measure used for feature selection. Information gain is helpful in determining the importance or relevance of the attribute or feature or term or token. Information gain is used to rank all the terms in the data set. Selecting the top few terms helps in removing the non-relevant features in the data set. This is also helpful in determining the effect of number of terms to show any significant improvement in the performance of machine learning techniques.

Representation of Textual Data

Text data is generally available in unstructured format. To make it analytical friendly, there is a strong need to make it structured. The textual data can be divided into multiple attributes, i.e. title, heading, sections, subsections, paragraphs, etc. There are many ways to make the textual reports structured. Each textual document contains terms as tokens and there are large numbers of terms/tokens available in repository of bug reports. The bug reports in the repository are represented as document*term matrix where the rows are considered as bug reports/documents and columns are considered as terms or tokens or words or features (Salton & Buckley, 1988). This matrix can be populated in either of the following ways:

- **Binary Representation:** The matrix will be filled up by zero or one for the absence or presence of a term in the document. Such type of representation will be helpful in probability based model such as Naive Bayes classification where presence or absence of a term is considered. This type of representation is biased towards a larger document. The rows of the matrix represent the document while the columns represent the term in the document set. If a document contains a particular term, it will set to the particular cell filled by value "1" otherwise "0".
- **Term Frequency Representation:** In this type of representation, the frequency of a term in the document will be counted and stored in the matrix. This type of representation will give importance to the term in

a particular document as in Multinomial Naive Bayes Classification. This will be normalized with the length of the document to avoid the bias towards the larger documents. Term frequency (TF) is a number of times a term appears in the document. This value is biased towards the larger documents. To avoid the biasness of the larger document, this should be normalized. The normalization of the term frequencies can be achieved by dividing the TF (Term-Frequency) by total number of terms in that document.

- **TF*IDF representation**: TF*IDF stands for "term frequency (TF) times inverse document frequency (IDF)". It is generally used to determine the importance of a term in the complete dataset or document set. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. The inverse document frequency is obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient. Such type of representation is used in ranking of terms and selecting top few terms.

TEXT MINING AND CLASSIFICATION ALGORITHMS

Text mining is a specialized data mining process. Seshasai (2000) describes the Ask Jeeves techniques for question answering. Weiss et al. (2000b) have evaluated the document matching algorithm and further generating model cases by document clustering (Weiss et al., 2000a). Sebastiani (2002) have surveyed various machine learning techniques for text categorization. Weiss and Verma (2002) have proposed a market intelligence system based on many sources of news on the web. Spam classifier for email filtering based on Mozilla data have been proposed by Graham (2003). Damerau et al. (2004) conducted a categorization experiments with the Reuters data set. Weiss et al. (2005) summarizes various predictive methods for analyzing unstructured information using text mining. Feldman and Sanger (2006) have widely studied Machine Learning in document classification. Pang and Lee (2008) have specified feature selection in text classification for improving classification effectiveness, computational efficiency, or both. Reddy et al. (2010) have used complementary Naïve Bayes for classifying the movie reviews in binary classes. Classification or categorization is a way of automatically assigning a document to a predefined category. In other words, categorizing the documents is based on their topic of relevance. Classification is broadly categorized either as supervised or unsupervised. In supervised classification, the classes of the training data set are already known whereas in unsupervised classification, the classes of the data are unknown. Classification algorithms have been developed and matured such as Decision Trees, Nearest Neighbor, Naive Bayes, Multinomial Naive Bayes, Support Vector Machine, Rule based learners, etc. (Mitchell, 1997). The data set is very sparse in nature because each document i.e., the summary of bug report contains very few terms. These terms are the representative terms to describe the nature of the report or documents. Following classification algorithms have been used in this paper:

- **Naïve Bayes Classifier (NB):** A naïve Bayes classifier estimates the class-conditional probability with assumption that all attributes are conditionally independent (Tan et al., 2006). The conditional independence assumption can be stated as

$$P(\mathbf{X} \mid Y = y) = \prod_{i=1}^{d} P(X_i \mid Y = y)$$

Where each attribute set **X**= {X_1, X_2, …,X_d} consists of d attributes and Y is the set of class labels. With the conditional independence assumption, instead of computing the class-conditional probability for every combination of **X**, we have to estimate the conditional probability of each X_i, given Y. To classify a test record, the naïve Bayes computes the posterior probability for each class Y as

$$P(Y \mid \mathbf{X}) = \frac{P(Y)\prod_{i=1}^{d} P(X_i \mid Y)}{P(\mathbf{X})}$$

Since P(**X**) is fixed for every Y, it is sufficient to choose the class that maximizes the numerator term, $P(Y)\prod_{i=1}^{d} P(X_i \mid Y)$. In this way, the conditional probabilities for each term with the given class is calculated.

For continuous attributes, there are two ways to estimate the class-conditional probabilities. (a) Discretize the continuous attribute and replace the value with its corresponding interval, (b) By assuming the probability distribution for the continuous attribute, the parameter of that probability distribution is estimated on training data. In general, the continuous attributes follow the Normal or Gaussian distribution to represent class-conditional probabilities. After calculating the conditional probabilities of each term with respect to every class, a new record can be classified based on the probabilities for each class of each term occurring within the document.

- **J48 (Java Implementation of C4.5):** J48 is an open source Java implementation of the C4.5 algorithm. C4.5 Algorithm (Quinlan, 1993), an extension of Quinlan's ID3 (Interactive Dichotomizer) algorithm (Quinlan, 1986) is developed by Ross Quinlan. This algorithm is used to generate a decision tree which is being used for classification for many days. C4.5 chooses one attribute of data that effectively split the set into subsets at each node of the tree. The information gain has been chosen for the splitting the data. The attribute with highest information gain will be chosen to make the split. The implementation of this algorithm is available in the Waikato Environment for Knowledge Analysis (WEKA) data mining tool (Hall et al., 2009) and also available in RapidMiner as WEKA extension.
- **Repeated Incremental Pruning to Produce Error Reduction (RIPPER):** RIPPER has been developed and implemented in java (Cohen, 1995) and quite useful to generate small rule sets used for classification. RIPPER, a rule covering algorithms that runs over the data in multiple passes (Cohen, 1995). This algorithm learns one rule at each pass for the majority class. For two class problems, RIPPER chooses the majority of class as its default class and learns the rules for detecting the minority class. The classes are ordered according to their frequencies in multi class problem. For example, let $(y_1, y_2, \ldots, y_c)$ be set of ordered classes where y_1 is the least frequent and y_c is the most frequent class. During the first iteration, the instances that belong to y_1 are labeled as positive examples, while others are labeled as negative examples. The sequential covering method is used to generate rules that discriminate the positive and negative examples. Next, RIPPER extracts rules that distinguish y_2 from other remaining classes. This process is repeated until we left with y_c (Tan et al., 2006). Implementation of RIPPER in RapidMiner is available as WEKA extension (W-JRIP).
- **Random Forest (RF):** A random forest is a collection of unpruned decision trees. The decision trees in the forest are built from a random subset of the training data (Breiman, 2001). The each subset of training data is used to construct a decision tree. Each decision tree is build to its specified maximum size without any pruning. Overall prediction has to be made based on the majority vote from all constructed decision trees in the forest.
- **Support Vector Machine (SVM):** Support Vector Machine (SVM) has been introduced by Vapnik (1995). SVMs are based on the structural risk minimization (SRM) principle (Joachims, 1997). This principle of SRM incorporates capacity control to prevent over-fitting and thus is a partial

solution to the bias-variance trade-off dilemma. SVM implementation requires two key elements namely, mathematical programming and kernel functions. The parameters can be calculated by solving a quadratic programming problem with linear equality and inequality constraints; rather than by solving a non-convex, unconstrained optimization problem. The flexibility of kernel functions allows the SVM to search a wide variety of hypothesis spaces.

Primarily SVM is developed for binary class (positive class and negative class) classification. Now this is being extended to m-class (multi-class) classification. Multi-class classification is handled in SVM using one against one, one against all and DAGSVM (Directed Acyclic Graph Support Vector Machine). Hsu and Lin (2002) have compared the performance with three methods of binary classifications: one-against-one, one-against-all and Directed Acyclic Graph - Support Vector Machine (DAGSVM) (Platt et al., 2000) for multi-class classification. They have found that one-against-one and DAGSVM are suitable and applicable for practical implementation with respect to time during the model building. The geometrical interpretation of support vector classification (SVC) is that the algorithm searches for the optimal separating surface, i.e. the hyperplane which is equidistant from the two classes (Burges, 1998). SVC is outlined first for the linearly separable case. If the data have not been separated by a linear decision surface, the features can be transformed into high dimensional space by using non-linear mapping. This mapping can be done by applying kernel function to the original problem space. Kernel functions have been introduced to develop non-linear decision surfaces. This technique has also been applied in text mining because it works well with the high dimensions of data and avoids the curse of dimensionality problem. Support Vector Machine will be useful when there is a sparse distribution of the features. The SVM can be used to predict the classes of the documents due to its robustness in handling of sparse data.

Learned models can be validated using n-fold cross validation. The confusion matrix for the binary (i.e., C1 and C2) classification is shown in Table 1.

We assume that C1 and C2 are the positive and negative classes respectively. From the Table 1, True Positives (TP) means C1 report predicted as C1 while false positives (FP) means C2 report predicted as C1. If the C1 is predicted as C2, called false negatives (FN) while C2 is predicted as C2 called true negatives (TN). Various accuracy measures i.e., accuracy, precision, recall and F-Measure can be calculated to validate the performance of the models. Accuracy is the percentage of correctly predicted sets of the documents. This covers the percentage of true prediction and provides the overall accurate prediction.

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

Precision is the percentage of predicted positive documents out of the total positive predictions. This covers the impact of false positives in prediction. If FP are increasing, the value of precision diminishes. Recall is the

Table 1. Confusion matrix

		Predicted Class	
		C1	**C2**
Actual Class	**C1**	True Positives (TP)	False Negatives (FN)
	C2	False Positives (FP)	True Negatives (TN)

percentage of correctly predicted positive out of the total number of positive documents. It covers the effect of false negatives (FN).

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

and

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

F-Measure is calculated to measure the average performance of the trained model to avoid the bias towards precision or recall. The F-Measure is twice of the harmonic mean of precision and recall which can be calculated as follows

$$F - Measure = \frac{2 * Precision * Recall}{(Precision + Recall)}$$

In this, the same weightage has been given to both precision and recall.

DATA COLLECTION AND IDENTIFICATION OF PROJECTS/COMPONENTS

A research was conducted to discuss the ways of teaching software engineering using Free/Libre and Open Source Software (FLOSS) projects and ways to collect and retrieve the relevant data (Stamelos, 2009). Authors also conducted a study on the utility of open source meta-repositories of open source projects which helps researchers in data collection and project selection. A comparison has also been done in FLOSSmole and FLOSSMetrics (Gonzalez-Barahona et al., 2010). Further, author examined the literature to determine the use of email archive in conducting the study in FLOSS (Squire, 2012).

In this paper, NASA's PITS datasets have been selected as closed source which uses its in-house developed bug reporting system. Bug repository of NASA PITS projects have been made available in the PROMISE (Predictor Models in Software Engineering) data repository (Boetticher et al., 2007) as a standard dataset on 20th January, 2008. Available PITS data have been downloaded for six projects namely PitsA, PitsB, PitsC, PitsD, PitsE and PitsF from this repository. The severity levels are defined on five point scales in these datasets. These scales range from one to five, from the fullest to the dullest. The severity wise distribution of reported bugs for the selected projects/components is shown in Table 2. The present study has selected some components of important open source projects namely, Mozilla (http://www.mozilla.org), Eclipse (http://www.eclipse.org) and GNOME (http://www.gnome.org). The open source projects selected for study are very popular in terms of their usage in the industry. Mozilla is one of the popular web browsers among the web users. Eclipse provides an integrated development environment for Java developers. GNOME is a desktop environment for the GNU/UNIX based operating systems. The Mozilla project has been started in 1998 with its first release in 2002. The Eclipse project has started in 1998 with its first release in 2003. The GNOME project has started in 1997 and gets its first release in 1999 as GNOME 1.0. These projects are varying with the applications and its use such as web browser, programming environment and desktop environment which are using bugzilla for bug reporting system. We have taken all the versions for the mentioned components of open source projects.

The Bug reports have been downloaded for Mozilla and GNOME components till 16th July, 2011and Eclipse till 4th August, 2011 from the start date of bug reporting in BugZilla reporting system.

For open source projects, bug reports from bug repositories have been downloaded from the bug reporting system using advanced search facility having resolution either "fixed" or "works for me" because only these types of bug reports contains the meaningful information for building the models or training the models while other types of reports are seems to be

Table 2. Severity wise list of components of various projects

Projects/ Components		Severity wise number of reported bugs						Number of terms
		1	2	3	4	5	Total	
PITS	**A**	0	111	155	34	3	303	697
	B	0	23	523	382	59	987	1001
	C	0	0	132	177	7	316	309
	D	0	1	167	13	1	182	347
	E	0	24	517	243	41	825	1059
	F	0	9	477	209	48	743	999
GNOME	**Contacts**	3	82	50	73	8	216	78
	Mailer	1	199	113	243	20	576	230
Mozilla	**BookMarks**	200	1933	2815	1917	671	7536	1110
	FFGeneral	8	747	1073	553	167	2548	888
	Layout	265	2876	1785	1052	414	6392	1638
Eclipse	**CDTDebug**	25	25	122	53	8	233	114
	EclipseDebug	23	97	213	72	39	444	187
	EclipseJDTUI	23	81	282	281	81	748	317
	EclipseSWT	71	161	298	64	36	630	258
	EclpiseUI	28	124	401	327	109	989	432
	IDEPlatform	23	75	267	148	85	598	333
	JDTUI	1	24	118	204	56	403	196

incomplete or duplicate. The severity levels in these bug reports have been defined in the range from blocker to trivial i.e., 1 to 7. The bug reports related to "enhancement" have been removed from the data set as they may not play any role in severity identification because they specify the inclusion of new feature or improvement in the existing feature not as bug. The reports having the default severity, i.e. normal have also been removed from the dataset. The "Normal" severity level may confuse the classifier because it is the default level and assigned by the users when they are not sure about the severity level of such bugs. In other words, if a user is filling the attributes of a newly occurred bug, one may not choose the severity level due to not having the complete idea about the impact of the bug. The remaining levels of the bug reports contain five categories, i.e. blocker, critical, major, minor and trivial. Blocker, critical and major types of bugs need attention as they report the system failure or crash or stuck up. Minor bug reports are generally used for cosmetic changes required in the software while trivial will not affect the system as they are generally messages or labels. Two components of GNOME, three components of Mozilla and seven components of Eclipse are selected from bug repositories maintained with bugzilla.

In the NASA projects, bug reports are clearly defined and the description of bug reports contains many terms. The distinguished terms are able to define the severity level of the bug reports. We removed those terms whose frequencies are very less or very high because they may not be able to distinguish the severity level of the bug reports. We also applied information gain to select most useful words which are important in defining the bug reports and distinguish the severity levels. We have shown sample bug reports for the projects PITSB in Table 3.

Table 3. Sample bug reports of PITSB

Severity	Description
4	SDN_BSM_LOAD_DUMP_COMMIT Lines 63-68, 107-112, 145-150 These sections of the test scenario do not fit in and need to be removed or more testing information needs to be added. Requirement 2018 is not verified at all by this test scenario. Requirement 2018 deals with the Commit Table With Word Count Command, which is not even used or addressed in the test scenario.
3	SND_BSM_LOAD_DUMP_COMMIT Lines 71-87 Requirements 1214 and 2006 are not properly verified. For requirement 1214, the test scenario needs to verify that current table session was terminated without affecting the onboard versions of the selected table. The test scenario does not include that. For requirement 2006, it is only partially verified.
5	Line 1350 in file pciconfig.c Line 1868 in file boot.c The sections labeled /** Imported Functions **/ are labeled incorrectly according to the branch standards. The section should be labeled /** External Definitions /
2	Array 'os_queue_blocks_array' of size 26 declared at line 138 may use index value 26. This will also cause an array index out of bounds on line 513. osids.h #define OS_MAX_QUEUE_IDS 26 File: \fsw\osapi\rtems\osal\osapi.c Function: OS_QueuePut Line 138: rtems_id os_queue_blocks_array [OS_MAX_QUEUE_IDS] 702 /* 703 ** Check Parameters 704 */ 705 if((queue_id > OS_MAX_QUEUE_IDS) || (os_queue_blocks_array[queue_id] == 0))

The description contains many words but very few words are common among the bugs reports. For example in severity level 2 bug reports, the term "Array" makes it severity level 2 while the term "requirements" make the severity level 3. The terms "standards" or "section" or "definitions" designate the severity level 5.

Sample bug reports of the open source components are shown in the Table 4. The term "Crash" clearly defines the severity level of the concern bug reports as "major" while term "warning" makes the severity level as "minor". The term "lock" appears in bug_Id 283107 makes its severity level critical. The porter stemmer algorithm makes the terms "lock" or "locked" or "locks" are same as "lock". The term "Widgets" makes the severity level as trivial because this feature is mainly used for GUI related.

Data is sparsely distributed as there are many number of terms/features and only very few terms are being considered by users while submitting the bug report. Thus, we have to analyze only those features which are important to the reports with respect to the severity level of the report. This has given us the idea to select top 25, 50, 75 and 100 contributing terms as relevant features. These features are selected

Table 4. Sample bug reports of eclipse (EclipseSWT subcomponent)

Bug_id	Severity	Short_desc (Short Description)
195337	major	Eclipse - SWT - JVM crash
283107	critical	Eclipse completely locked up waiting for the SWT internal lock
289346	trivial	[Widgets] Gtk SWT should set type hint for floating shell windows
293945	major	use of uname -p and uname -i is unreliable; swt is wrongly built on 64bit archs
297662	major	crash in libswt-pi-gtk-3624.so
71679	minor	HTTP POST creates "Security Warning" dialogs on Mozilla impl of SWT Browser
151439	major	vm crash when running swt tests browswer tests on SLE_D 10

using information gain criteria based on entropy based measurement from the collected set of bug reports. This will help in measuring the effect of number of terms in performance measure of various machine learning techniques.

CLASSIFIER BUILDING AND EVALUATION

Preprocessing of the bug reports has been done using the standard process i.e., tokenization, stop words removal, removal of terms having length less than three or greater than 50 characters, stemming, removal of low/high frequent terms based on TF*IDF pruning and feature selection based on information gain (Menzies & Marcus, 2008). Lamkanfi et al. (2010) have conducted a study to show that a summary of the bug report is sufficient to explain the longer description of the bug reports. Later on, Lamkanfi et al. (2011) have confirmed that the developed classifiers are component dependent in predicting the severity of the bug report. This supports our approach to train classifiers on a component basis and used one line summary for open source projects.

The methodology contains two steps to develop the classifier for predicting the severity level. First step is the pre-processing of dataset and followed by the classifier building. In the preprocessing of bug reports, following steps have been followed:

1. Tokenization of bug reports (t)
2. Stop words removal (s)
3. Removal of terms having length less than three and more than fifty characters (r)
4. Stemming of words (s1)
5. Pruning of low frequent as well as high frequent terms through TF*IDF (p)
6. Selection of terms based on Information Gain criteria (i)

The paper analyzes NASA Pits dataset and Eclipse, GNOME and Mozilla using tsrs1p(N1) i(N2) where N1 is the number of remaining terms after the TF*IDF pruning and N2 is the number of terms selected based on Information Gain out of N1 terms. In our study, we have varied the terms for N2 from 25 to 100 in the intervals of 25.

Classification models using NB, SVM, J48, RIPPER and RF have been developed by varying number of terms/features using InfoGain criteria. The learned classifiers except random forest have been evaluated by applying 10-fold cross validation approach using stratified sampling to split data between training and testing set. In each case, a classification model have been trained using 90% of the data and tested on the remaining 10% of the data and performance measures have been computed namely, mean accuracy, mean precision, mean recall, and mean F-Measure. For each data set, the experiments have been repeated on these trained models 10 times, rotating the 10% testing part of the data. The random forest does not need the cross validation because the construction of forest contains self-test. The analysis was carried in RapidMiner (Mierswa et al., 2006) and STATISTICA (Statsoft, 2006) tools.

RESULTS AND DICSUSSIONS

The performance measures namely, accuracy, precision, recall and F-Measure values are obtained for different machine learning techniques with varying number of features/terms. We have measured the performance of different machine learning techniques by considering (i) the value of accuracy and F-Measure for all severity level and (ii) number of best cases at different threshold level of accuracy and F-Measure. We set threshold levels as >40%, >50%, >60%, >70% and >80% for performance measure accuracy and F-Measure.

Research Question 1: In answer to research question 1, we have applied five machine learning techniques on six projects of closed source projects and 12 components of open source projects to determine the bug severity. The value of performance measure accuracy for different machine learning techniques has been shown in Table 5 and Table 6 for closed source and open source projects respectively.

Table 5. Accuracy of various classifiers with varying number of terms for NASA's PITS dataset (closed source projects)

Projects	No. of Terms	NB	SVM	J48	RIPPER	RF
PitsA	25	56.11%	66.34%	62.71%	67.33%	62.38%
	50	60.73%	65.68%	66.34%	66.01%	58.49%
	75	65.02%	65.35%	64.69%	67.66%	62.09%
	100	68.98%	66.34%	68.32%	67.00%	54.79%
PitsB	25	28.98%	57.55%	55.93%	58.66%	52.26%
	50	38.60%	60.28%	57.24%	59.78%	52.46%
	75	49.44%	60.49%	57.14%	59.27%	53.12%
	100	51.17%	60.79%	58.36%	59.78%	53.49%
PitsC	25	77.22%	81.96%	81.33%	81.65%	71.19%
	50	86.71%	81.33%	83.86%	77.85%	69.90%
	75	87.34%	80.06%	83.23%	77.85%	70.80%
	100	87.03%	82.28%	83.23%	79.11%	57.31%
PitsD	25	93.96%	96.70%	96.15%	94.51%	94.44%
	50	92.31%	96.15%	95.60%	95.05%	92.74%
	75	94.51%	96.15%	95.60%	93.96%	92.00%
	100	92.86%	96.15%	95.60%	93.96%	91.94%
PitsE	25	28.85%	69.45%	70.30%	68.97%	66.67%
	50	33.70%	69.58%	70.42%	68.97%	65.61%
	75	38.30%	69.09%	70.42%	69.21%	63.80%
	100	43.88%	69.45%	69.94%	69.21%	61.47%
PitsF	25	42.34%	69.89%	67.34%	66.80%	62.75%
	50	47.45%	71.24%	67.34%	67.07%	63.44%
	75	49.19%	72.58%	67.34%	66.80%	63.53%
	100	51.75%	72.45%	67.34%	66.94%	63.35%

We defined different threshold values for the performance measure accuracy and F-Measure in five groups i.e., >40%, >50%, >60%, >70% and >80%. There are 120 cases i.e. five techniques ran on six closed source projects of PITS data set for four terms sets (25, 50, 75, and 100). Out of 120 cases of closed source, we found the number of cases with respect to different threshold level of accuracy as 115 cases with >40%, 110 cases with >50%, 91 cases with >60%, 44 cases with >70%, and 32 cases with >80% level as shown in Figure 1.

Similarly, there are 240 cases i.e. five techniques ran on twelve projects for four terms sets (25, 50, 75, and 100) of open source projects consisting GNOME, Mozilla and Eclipse. Out of these 240 cases, we found 174 cases of more than 40%, 54 cases of more than 50% and only one 1 cases of more than 60% accuracy as shown in Figure 1. If we further combined the results of these dataset i.e. open and closed source projects, we found that there are 80% and 45% cases having accuracy level >40% and >50% respectively.

Table 6. Accuracy of various classifiers with varying number of terms for open source projects

Projects/ Components	No. of Terms	NB	SVM	J48	RIPPER	RF
GNOME						
Contacts	25	31.94%	51.85%	46.76%	45.37%	23.97%
	50	40.74%	54.63%	43.52%	45.83%	22.97%
	75	43.06%	52.78%	44.91%	45.83%	24.79%
	100	41.67%	52.78%	44.91%	45.83%	24.79%
Mailer	25	26.04%	54.86%	53.82%	56.77%	45.89%
	50	30.90%	56.94%	51.56%	55.90%	41.32%
	75	38.19%	57.47%	50.35%	55.56%	41.10%
	100	42.01%	59.38%	51.22%	54.51%	42.33%
Mozilla						
BookMarks	25	5.57%	38.88%	38.61%	39.46%	37.64%
	50	7.99%	39.21%	38.81%	39.95%	37.61%
	75	9.54%	39.84%	38.99%	39.98%	37.56%
	100	10.72%	40.21%	39.04%	40.15%	37.49%
FFGeneral	25	9.77%	45.49%	43.05%	45.17%	43.60%
	50	15.42%	46.70%	42.97%	45.02%	43.34%
	75	18.49%	46.15%	42.86%	45.49%	43.56%
	100	20.53%	46.70%	42.70%	45.80%	43.59%
Layout	25	18.18%	46.57%	45.96%	45.81%	46.03%
	50	13.49%	47.54%	46.93%	46.67%	45.41%
	75	15.49%	48.34%	47.14%	47.42%	45.28%
	100	17.37%	48.50%	47.20%	47.65%	45.42%
Eclipse						
CDTDebug	25	25.32%	60.94%	54.08%	52.36%	54.32%
	50	34.33%	58.37%	53.22%	51.07%	53.37%
	75	40.77%	58.37%	51.93%	50.21%	51.23%
	100	46.35%	58.37%	52.79%	50.21%	49.69%
EclipseDebug	25	20.05%	48.87%	46.85%	48.20%	50.33%
	50	21.62%	49.32%	47.75%	48.20%	49.17%
	75	24.32%	50.23%	48.20%	46.62%	47.51%
	100	28.15%	50.00%	47.07%	47.30%	46.03%
EclipseJDTUI	25	15.91%	45.99%	45.72%	45.72%	42.86%
	50	20.19%	48.26%	46.93%	45.72%	38.43%
	75	24.87%	49.20%	47.86%	45.86%	38.13%
	100	27.67%	50.13%	47.06%	43.98%	38.52%

continued on following page

Table 6. Continued

Projects/ Components	No. of Terms	NB	SVM	J48	RIPPER	RF
EclipseSWT	25	34.29%	50.95%	49.21%	51.27%	46.46%
	50	23.02%	51.11%	47.94%	50.48%	47.38%
	75	24.29%	52.86%	47.94%	49.68%	46.81%
	100	25.24%	53.49%	46.03%	49.05%	46.96%
EclipseUI	25	14.05%	47.62%	44.49%	47.52%	39.50%
	50	20.02%	47.62%	45.10%	47.72%	39.60%
	75	25.28%	45.30%	44.69%	47.52%	40.29%
	100	27.50%	46.11%	45.10%	47.62%	40.72%
IDEPlatform	25	18.23%	45.99%	50.50%	50.17%	44.47%
	50	24.08%	47.16%	50.67%	49.33%	44.95%
	75	30.10%	48.33%	48.83%	49.33%	44.08%
	100	29.93%	48.49%	46.49%	47.83%	45.18%
JDTUI	25	22.83%	55.09%	54.09%	52.85%	48.54%
	50	33.00%	54.34%	53.85%	2.28%	48.74%
	75	37.97%	52.85%	54.59%	51.12%	47.67%
	100	38.96%	52.85%	51.36%	52.36%	49.08%

We also performed the similar aggregation on F-Measure. The value of F-Measure is obtained for each severity level and corresponding number of cases in each category are shown in Figure 2. There are 480 (i.e., 120 and four severity levels) and 1200 (240 and five severity levels) cases for closed and open source projects respectively. There are 208, 182, 159

Figure 1. Classifiers performance based on accuracy

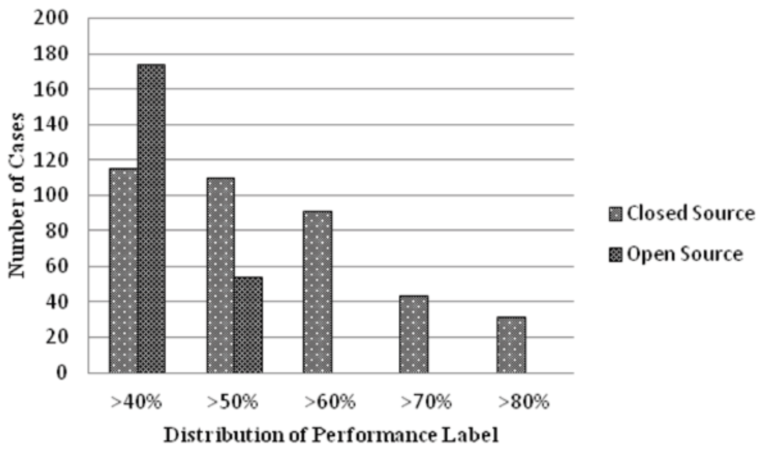
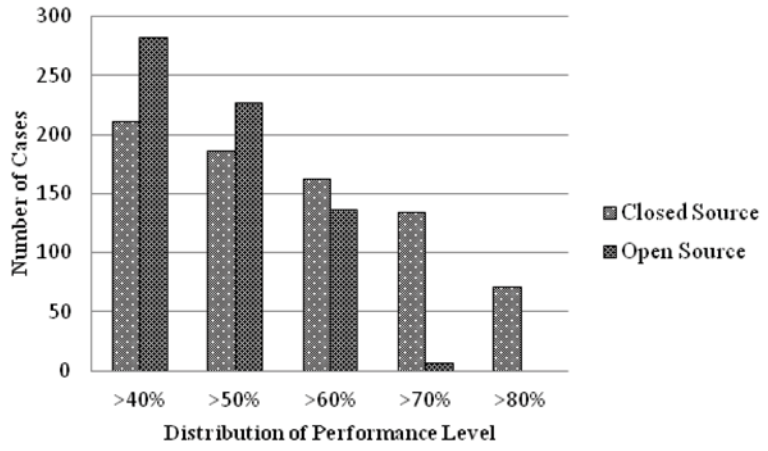


Figure 2. Classifiers performance based on f-measure



and 130 cases for closed source projects shows the threshold level >40%, >50%, >60% and >70% of F-Measure respectively while remaining cases receives lower value of F-Measure. Similarly, there are 283, 227 and 137 number of cases of open source projects with the threshold level of >40%, >50% and >60% respectively.

This analysis clearly shows the applicability of machine learning technique in determining the severity level of the bug report which answers the research question 1.

Research Question 2: In answer to research question 2, we have measured the performance measure F-Measure and accuracy of different machine learning techniques for all severity level and also found the number of best cases at different threshold level of accuracy and F-Measure. A graphical presentation of comparative performance of different techniques has been shown in Figure 3 through Figure 14.

Figure 3. Classifiers performance based on accuracy with respect to different severity levels in closed source projects

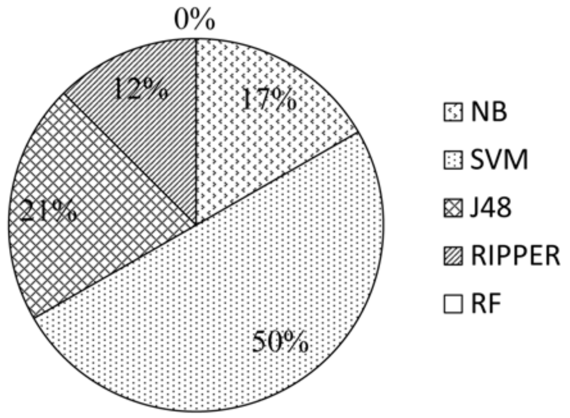
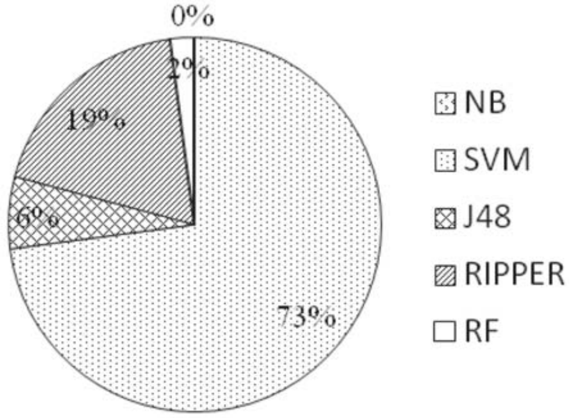


Figure 4. Classifiers performance based on accuracy with respect to different severity levels in open source projects



The aggregated performance of classifier based on accuracy for closed and open source project are shown in Figure 3 and Figure 4. The performance of NB, SVM, J48, RIPPER and RF classifiers are 17%, 50%, 21%, 12%, 0% for closed source and 0%, 73%, 6%, 19%, 2% for open source projects. The combined results of classifiers namely NB, SVM, J48, RIPPER and RF are 6%, 65%, 11%, 17% and 1% as shown in Figure 5.

To further analyze the performance based on F-Measure by considering different severity levels (1-5) with varying number of terms i.e. 25, 50, 75 and 100, we found 228 cases of open source projects and 80 cases of closed source projects. The cases based on top values of F-Measure have been received 41%, 28%, 25%, 6%, and 0% for NB SVM, J48, RIPPER and RF respectively for closed source projects as shown in Figure 6. The cases based on top value of F-Measure has been received 59%, 26%, 5%, 4%, and 6% for NB, SVM, J48, RIPPER and RF respectively for open source projects as shown in Figure 7.

Contribution for both types of projects i.e. closed and open source projects have been found 55%, 26%, 10%, 5%, and 4% for NB

Figure 5. Classifiers performance based on accuracy with respect to different severity levels in both open and closed source projects

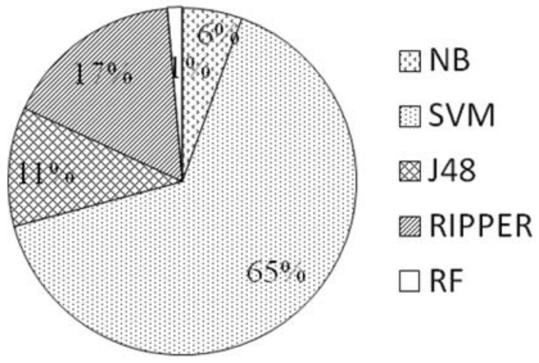
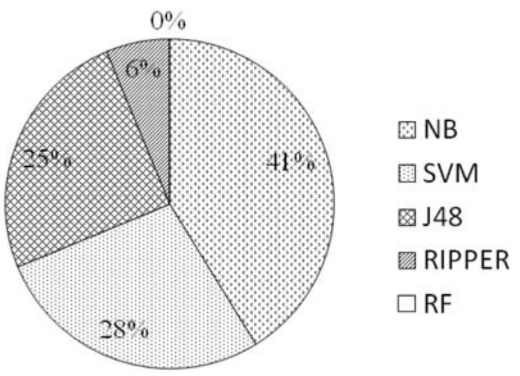


Figure 6. Classifiers performance based on f-measure with respect to different severity levels in closed source projects



SVM, J48, RIPPER and RF respectively as shown in Figure 8.

Figure 9 shows the number of cases for different machine learning techniques at different threshold levels. It is observed from figure that all cases have received accuracy more than 50% except NB. We also found that the number of cases decreases as we increase the threshold level. SVM and J48 techniques have more number of cases in comparison with other techniques for different threshold level.

Figure 10 shows the number of cases for different machine learning techniques at different threshold levels of accuracy. It is observed that SVM technique has more number of cases for different threshold levels. We also observed that at threshold level 40% SVM, J48 and RIPPER performance equally.

If we combine the performance of both open and closed source projects, we found SVM has more number of cases for different threshold level of accuracy, as presented in Figure 11.

The similar categorization for F-Measure has been performed and it is shown in Figure 12 and Figure 13 for closed and open source projects respectively. In case of closed source projects, the number of cases decreases significantly with increase in threshold level i.e. from >40% to > 60%. SVM shows significant improvement over NB, J48, RIPPER and RF

Figure 7. Classifiers performance based on f-measure with respect to different severity levels in open source projects

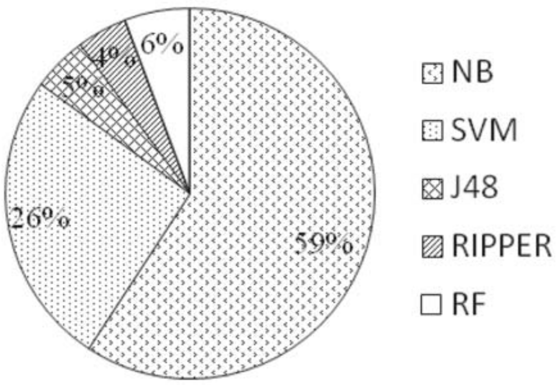
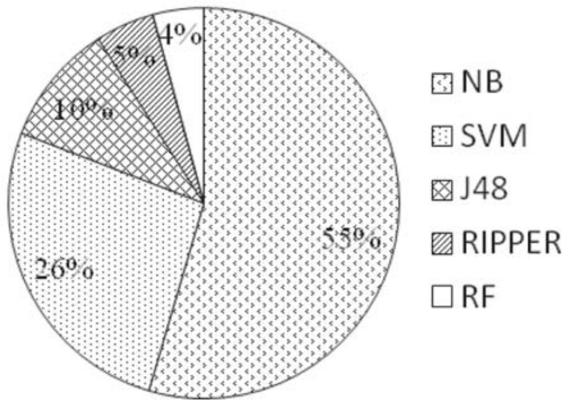


Figure 8. Classifiers performance based on f-measure with respect to different severity levels in both open and closed source projects



classifiers. There is only one instance where J48 performs better than SVM in case of threshold level >60%.

In case of open source projects, at all threshold level except 40%, SVM has more number of cases. We have also observed that number of cases decreases as we increase the threshold value like in case of accuracy.

Figure 14 shows the number of cases for different techniques at different threshold levels by considering the combined result of closed source and open source projects. In this case also SVM has got more number of cases at all threshold levels except 40%.

In terms of number of cases observed at different threshold value of F-Measure and Accuracy, the order of applicability in deter-

Figure 9. Classifiers performance based on Accuracy in closed source projects

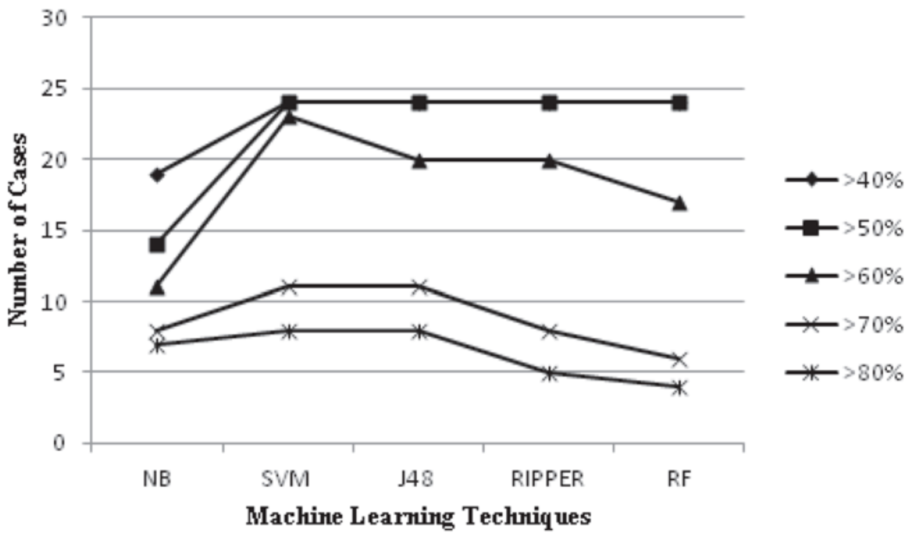
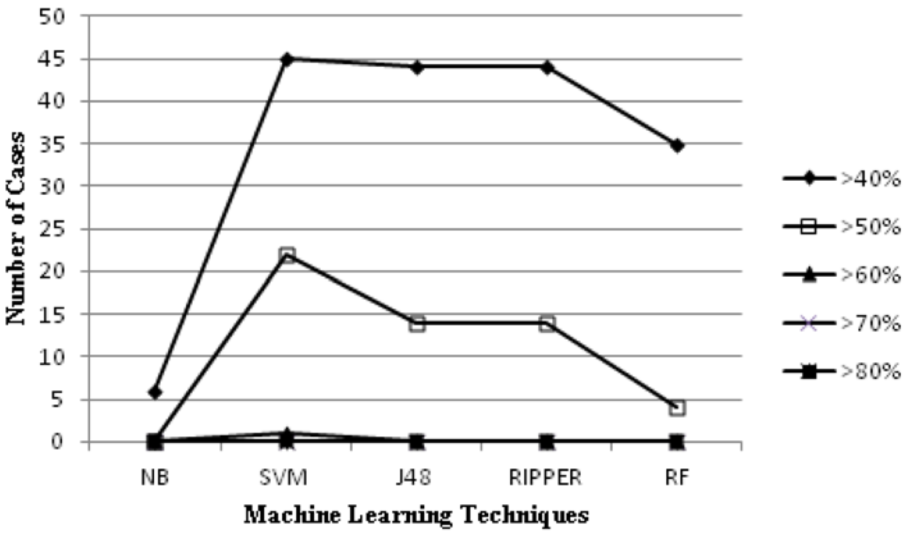


Figure 10. Classifiers performance based on accuracy in open source projects



mining the severity level of bug reports are SVM, J48, RIPPER, RF and NB but on the basis of F-Measure for all severity level it is NB, SVM, J48, RIPPER, and RF. We can say that quality wise NB is better but quantity wise SVM is better.

Research Question 3: In answer to research question 3, we have calculated the frequency distribution of performance measure accuracy as well as F-Measure by varying the number of terms for open and closed source projects.

We have considered the threshold level i.e., >40%, >50%, >60%, >70% and >80% with respect to number of cases by varying the number of terms. The trends for performance

Figure 11. Trends of classifiers performance for combined (closed and open source) projects based on accuracy

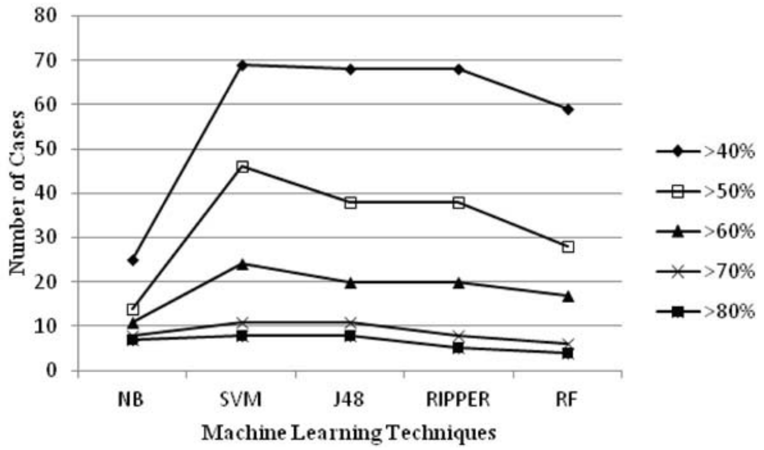
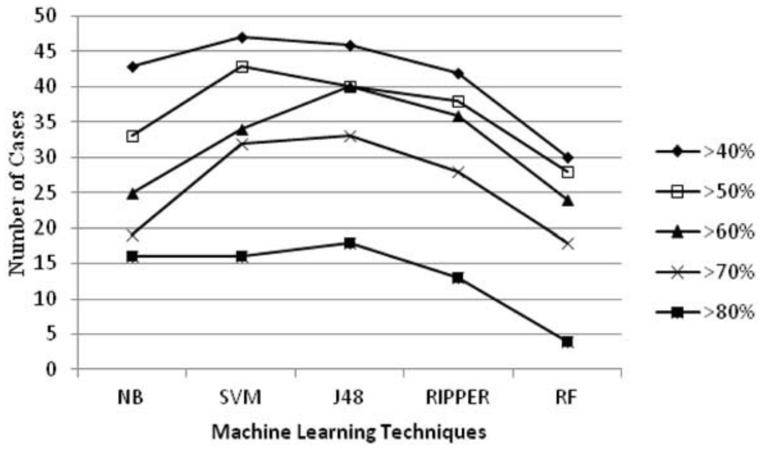


Figure 12. Classifiers performance based on f-measure in closed source projects



measure accuracy are shown in Figure 15 and Figure 16 for closed and open source projects respectively. The trends for performance measure F-Measure are shown in Figure 17 and Figure 18 for closed and open source projects respectively. We observed that number of cases decreases as we increases the value of threshold level but did not find any significant improvement on value of accuracy and F-Measure by increasing the number of terms but in some cases the value fluctuates with minor deviation as shown in Figure 15 to Figure 18.

THREATS TO VALIDITY

Yin (2009) presented the case study research in his recent book with threats to validity of the research conducted by the researchers in the social science perspectives. Runeson and Host (2009) conducted an empirical study for reporting the case study specific to software engineering. Authors have taken the issues from these two studies and organize them into four categories.

Figure 13. Classifiers performance based on f-measure in open source projects

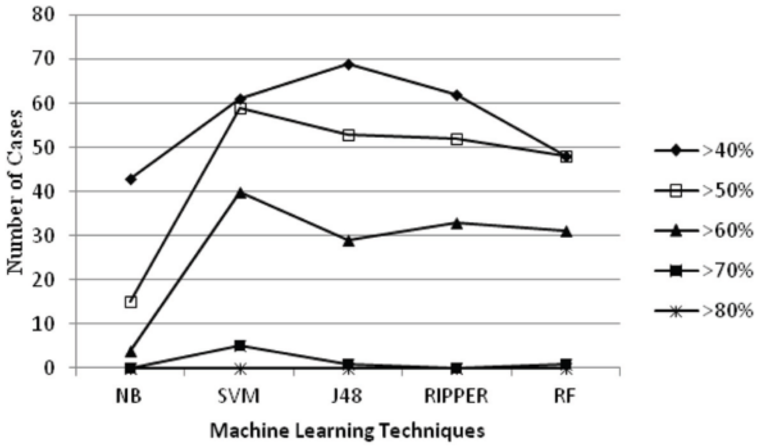
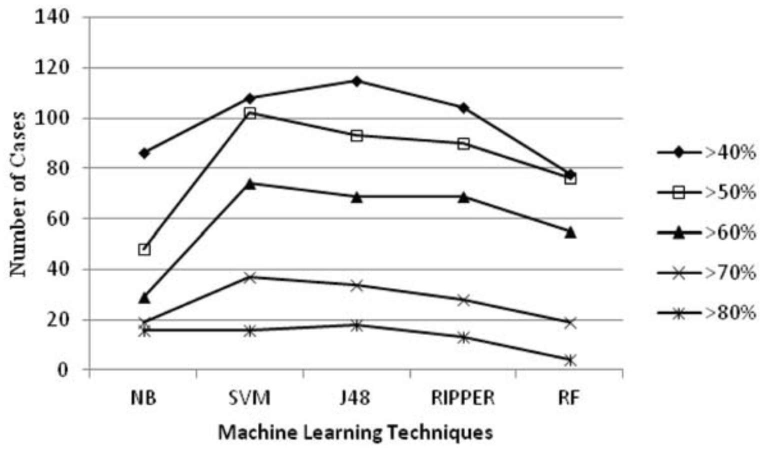


Figure 14. Classifiers performance of combined (closed and open source) projects based on f-measure



- **Construct Validity**: The classifier is trained on bug report data of project specific for the NASA's Pits projects as closed source and component wise for popularly known and used three open source projects. The bug reports are retrieved using the component wise with the suitable terms in the queries. The distribution of the downloaded bug report is sufficient that minimizes the risk towards biased results with a specific category or components or projects.
- **Internal Validity**: We have given equal importance to the every feature/term while developing the models on the training data using 10-fold cross validation. The cross validation approach minimizes the risk of internal validity of the results. We have not considered the problem of data imbalance.

Figure 15. Classifiers performance based on accuracy with respect to number of terms in closed source projects

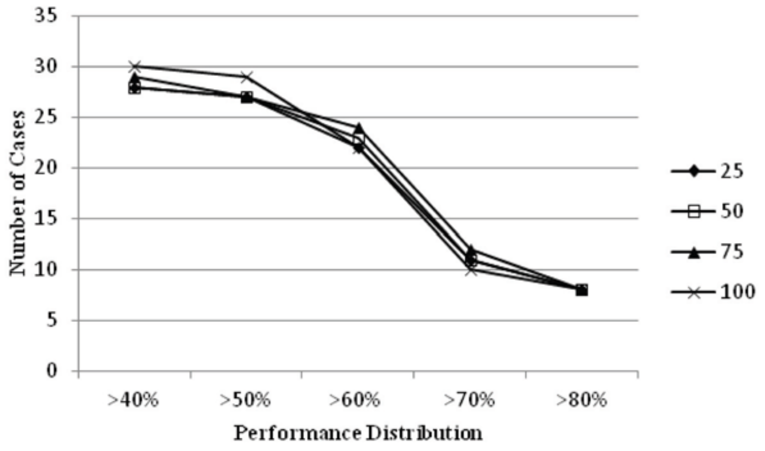
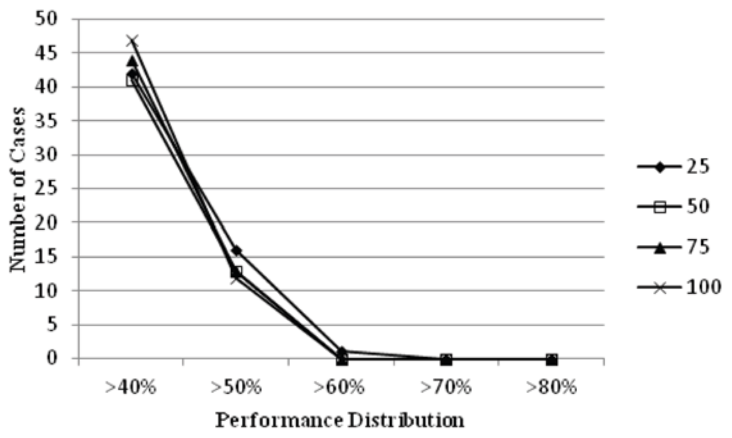


Figure 16. Classifiers performance based on accuracy with respect to number of terms in open source projects



- **External Validity**: We have considered individual components of the popular open source projects and NASA projects as closed source projects. Every bug tracking and reporting system requires the summary, long description and other attributes from the users. We have not considered long description and other attribute of bug report for model building. We have also not validated the predicted severity level of bug report with CVS or SVN.
- **Reliability**: We have made an assumption regarding the true assignment of the severity for the reported bug because we have excluded the reports having the level "Normal" available in the components of open source projects. We have used Rapid-Miner (http://www.rapid-i.com/) tool for data preprocessing, model building and

Figure 17. Classifiers performance based on f-measure with respect to number of terms in closed source projects

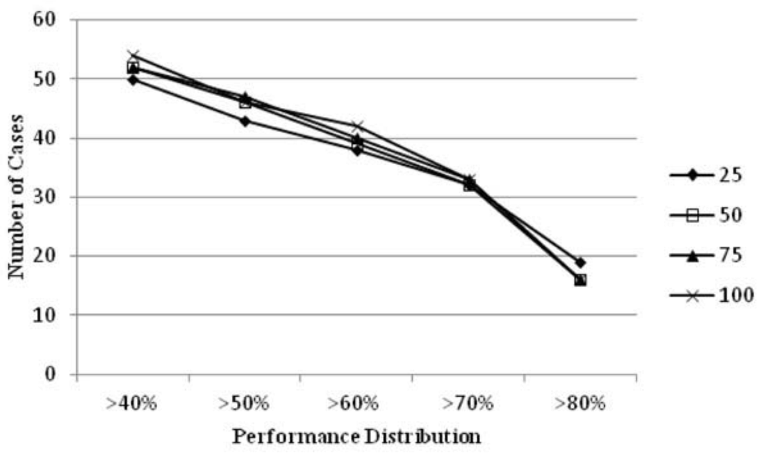
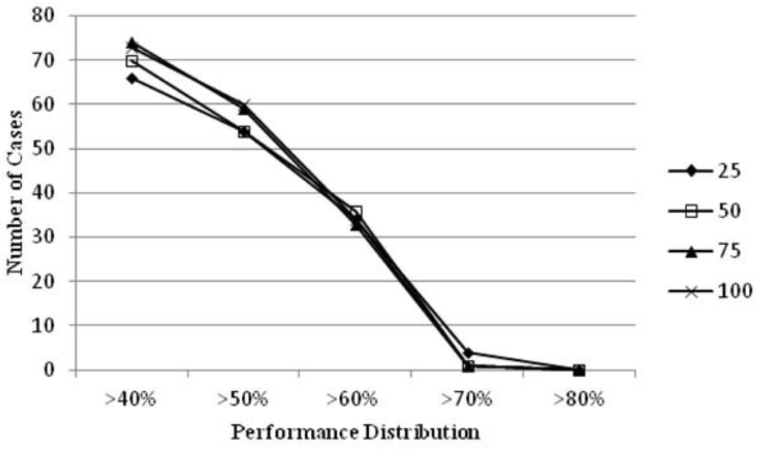


Figure 18. Classifiers performance based on f-measure with respect to number of terms in open source projects



10-fold cross validation for validation of results. The increasing use of RapidMiner tool in data mining community confirms the reliability of the tool.

CONCLUSION

Our study determines the applicability of different Machine Learning techniques namely Naïve Bayes, Support Vector Machine, J48, Random Forest, rule based RIPPER in predicting the severity level of a reported bug (1-5) for closed and open source projects using 10-fold cross validation. We have used the distribution of performance measure criteria namely, accuracy and F-Measure by setting threshold level as >40%, >50%, >60%, >70% and >80% to validate the applicability of these techniques. In this study, the following observations have been made:

- The performance measure on the basis of F-Measure and Accuracy mentioned in Table 5 through Table 6 along with graphical presentation in answer to research question 1 shows the applicability of machine learning techniques in determining the severity level of the bug reports.
- NB, SVM, j48, RIPPER and RF performs in 41%, 28%, 25% 6% and 0% cases for closed source projects and 59%, 26%, 5%,4%, and 6% in open source projects on the basis of F-Measure across all severity level.
- For threshold value of F-Measure with >40%, we found that NB, SVM, J48, RIPPER and RF performs best in 21%, 23%, 22%, 20% and 14% cases for closed source projects and 15%, 22%, 24%, 22%, and 17% in open source projects. If we increase this threshold value as >50%, we observed that NB, SVM, J48, RIPPER and RF perform at top in 18%, 24%, 22%, 21% and 15% cases for closed source projects and 7%, 26%, 23%, 23%, and 21% in open source projects.
- In terms of number of cases observed at different threshold value of F-Measure and Accuracy, the order of applicability in determining the severity level of bug reports are SVM, J48, RIPPER, RF and NB but on the basis of F-Measure for all severity level it is NB, SVM, J48, RIPPER, and RF. We can say that quality wise NB is better but quantity wise SVM is better.

- We have observed that there is no significant difference in the performance of different machine learning techniques as we increase the number of terms.
- We have also observed that prediction of severity levels of the reported bugs of closed source projects is better than open source projects.

This paper will take the research forward in analyzing the short description based feedback/ bug reports/ news captions to classify in categories by classification using the machine learning techniques.

FUTURE RESEARCH AGENDA

In future, the authors will work on the following research agenda:

- This study will be carried out on more projects/components of open source projects to validate and fine tune the parameters to implement the trained model in automatic determination of severity level of the bug report.
- The study will be further carried out in handling of imbalanced data with machine learning techniques.
- The analysis can be further carried out to determine the optimum number of bug reports as well as optimum number of features required to get the best performance.
- Research will also be carried out using clustering of terms by making a global and local dictionary which can help in reducing the number of features.

ACKNOWLEDGMENT

Authors are thankful to the anonymous reviewers for their constructive feedback which has improved the manuscript. We are also thankful to Prof. P.K. Kapur and Dr. (Mrs.) Punam Bedi for their constructive comments in improving the manuscript. Second author acknowledges with thanks the research grants received from Department of Science and Technology (DST), Govt. of India with Grant No. SR/S4/MS: 600/09.

REFERENCES

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., & Gueheneuc, Y. G. (2008). Is it a bug or an enhancement? A text-based approach to classify change requests. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research (CASCON '08)* (pp. 304– 318). ACM Press.

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)* (pp. 361-370). ACM Press.

Baysal, O., Davis, I., & Godfrey, M. W. (2011). A tale of two browser. In *Proceedigns of International Conference on Mining Software Repositories (MSR'11)* (pp. 238-241). ACM Press.

Bettenburg, N., Just, S., Schroter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008). What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (pp. 308–318). ACM Press

Bhattacharya, P., & Neamtiu, I. (2010). Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *Proceedings of 2010 IEEE International Conference on Software Maintenance (ICSM'10)* (pp. 1-10). IEEE Computer Society.

Boetticher, G., Menzies, T., & Ostrand, T. (2007). *PROMISE repository of empirical software engineering data*. West Virginia University, Department of Computer Science, West Virginia. Retrieved from http://promisedata.org

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. doi:10.1023/A:1010933404324.

Burges, C. J. C. (1998). A tutorial on support vector machine for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167. doi:10.1023/A:1009715923555.

Cerrito, P. (2006). *Introduction to data mining: Using SAS® enterprise miner*. SAS Institute, Inc..

Chaturvedi, K. K., & Singh, V. B. (2012). Determining bug severity using machine learning techniques. In *Proceedings of 6th CSI-IEEE International Conference on Software Engineering (CONSEG-2012)* (pp. 378-387). IEEE Explore.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings 12th International Conference on Machine Learning* (pp. 115-123).

Cubranic, D., & Murphy, G. C. (2004). Automatic bug triage using text categorization. In *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering* (pp. 92–97). KSI Press.

Damerau, F., Zhang, T., Weiss, S., & Indurkhya, N. (2004). Text categorization for a comprehensive time-dependent benchmark. *Information Processing & Management*, *40*(2), 209–221. doi:10.1016/S0306-4573(03)00006-2.

Feldman, R., & Sanger, J. (2006). *The text mining handbook: Advanced approaches in analyzing unstructured data*. Cambridge University Press. doi:10.1017/CBO9780511546914.

Gaeul, J., Sunghun, K., & Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE '09)* (pp. 111–120). ACM Press.

Gegick, M., Rotella, P., & Xie, T. (2010). Identifying security bug reports via text mining: An industrial case study. In *Proceedings of the 7th Working Conference on Mining Software Repositories,* Cape Town, South Africa (pp. 11-20).

Gonzalez-Barahona, J. M., Izquierdo-Cortazar, D., & Squire, M. (2010). Repositories with public data about Software development. *International Journal of Open Source Software and Processes*, *2*(2), 1–13. doi:10.4018/jossp.2010040101.

Graham, P. (2003). Better bayesian filtering. In *Proceedings of the 2003 Spam Conference.* Retrieved from http://spamconference.org/proceedings2003.html

Guo, P. J., Zimmermann, T., Nagappan, N., & Murphy, B. (2010). Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (Vol. 1, pp. 495-504).

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutermann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18. doi:10.1145/1656274.1656278.

Han, J., & Kamber, M. (2006). *Data mining: Concepts and techniques*. San Francisco, CA: Morgan Kaufmann Publishers Inc..

Herraiz, I., German, D., Barahona, J. G., & Robles, G. (2008). Towards a simplification of the bug report form in Eclipse. In *Proceedings of 5th International Working Conference on Mining Software Repositories (MSR'08)* (pp. 145-148).

Hooimeijer, P., & Weimer, W. (2007). Modeling bug report quality. In *Proceedings of the twenty-second IEEE/ACM International Conference on Automated Software Engineering (ASE '07)* (pp. 34-43). ACM Press.

Hsu, C. W., & Lin, C. J. (2002). A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, *13*, 415–425. doi:10.1109/72.991427 PMID:18244442.

Joachims, T. (1997). Text categorization with support vector machines: Learning with many relevant features. *University at Dortmund, LS VIII-Report* (*Tech. Rep.*), *23*.

Kanwal, J., & Maqbool, O. (2010). Managing open bug repositories through bug report prioritization using SVMs. In *Proceedings of the International Conference on Open-Source Systems and Technologies*, Lahore, Pakistan.

Kanwal, J., & Maqbool, O. (2012). Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*, *27*(2), 397–412. doi:10.1007/s11390-012-1230-3.

Ko, A. J., Myers, B. A., & Chau, D. H. (2006). A linguistic analysis of how people describe software problems. In *Proceedings of the Visual Languages and Human-Centric Computing (VLHCC '06)* (pp. 127–134). IEEE Computer Society.

Konchady, M. (2006). *Text mining application programming*. Thomson Delmar Learning.

Lamkanfi, A., Demeyer, S., Giger, E., & Goethals, B. (2010). Predicting the severity of a reported bug. In *Proceedings of Mining Software Repositories (MSR'10)* (pp. 1-10).

Lamkanfi, A., Demeyer, S., Soetens, Q. D., & Verdonck, T. (2011). Comparing mining algorithms for predicting the severity of a reported bug. In *Proceedings of Mining Software Repositories (MSR'11)* (pp. 249-258).

Linstead, E., & Baldi, P. (2009). Mining the coherence of GNOME bug reports with statistical topic models. In *Proceedings of 6th IEEE International Working Conference on Mining Software Repositories* (MSR '09) (pp. 99-102). IEEE Computer Society.

Matter, D., Kuhn, A., & Nierstrasz, O. (2009). Assigning bug reports using a vocabulary based expertise model of developers. In *Proceedings of the 2009 6th International Working Conference on Mining Software Repositories(MSR'09)* (pp. 131-140). IEEE Computer Society.

Menzies, T., & Marcus, A. (2008). Automated severity assessment of software defect reports. In *IEEE International Conference on Software Maintenance* (pp. 346–355).

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., & Euler, T. (2006). YALE: Rapid prototyping for complex data mining Tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06).* Retrieved from http://www.rapid-i.com

Mitchell, T. M. (1997). *Machine learning*. McGraw Hill.

Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, *2*(1-2), 1–135. doi:10.1561/1500000011.

Platt, J. C., Cristianini, N., & Shawe-Taylor, J. (2000). Advances of Neural Information Processing Systems: Vol. 12. *Large margin DAGs for multiclass classification* (pp. 547–553). MIT Press.

Porter, M. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130–137. doi:10.1108/eb046814.

Prifti, T., Banerjee, S., & Cukic, B. (2011). Detecting bug duplicate reports through local references. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Ban,Canada (pp. 1-9).

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. doi:10.1007/BF00116251.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann Publishers.

Reddy, S. R., Somayajulu, D. V. L. N., & Dani, A. R. (2010). Classification of movie reviews uUsing complemented Naive Bayesian classifier. [IJICR]. *International Journal of Intelligent Computing Research*, *1*(4), 162–167.

Runeson, P., Alexandersson, M., & Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th International Conference on Software Engineering* (pp. 499-510).

Runeson, P., & Host, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. doi:10.1007/s10664-008-9102-8.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, *24*(5), 513–523. doi:10.1016/0306-4573(88)90021-0.

Sarma, K. (2007). *Predictive modeling with SAS enterprise miner*. SAS Institute, Inc..

SAS Institute Inc. (2004). Getting started with SAS 9.1 text miner. Cary, NC.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, *34*(1), 1–47. doi:10.1145/505282.505283.

Seshasai, S., & Katz sue, W. (2000). Ask Jeeves: AI lab researchers attempt to enforce natural language patent. *The Tech (MIT)*. Retrieved from http://www-tech.mit.edu/V119/N56/

Singh, V. B., & Chaturvedi, K. K. (2011). Bug tracking and reliability assessment system. *International Journal of Software Engineering and Its Applications*, *5*(4), 17–30.

Squire, M. (2012). How the FLOSS research community uses email archives. *International Journal of Open Source Software and Processes*, *4*(1), 37–59. doi:10.4018/jossp.2012010103.

Stamelos, I. (2009). Teaching software engineering with Free/Libre open source projects. *International Journal of Open Source Software and Processes*, *1*(1), 72–90. doi:10.4018/jossp.2009010105.

StatSoft, Inc. (2006). *STATISTICA (data analysis software system),* (Version 7.1). Retrieved from www.statsoft.com

Sureka, A., & Jalote, P. (2010). Detecting duplicate bug report using character N-Gram based features. In *Proceedings of 17th Asia Pacific Software Engineering Conference*, Sydny, Australia (pp. 366-374).

Tamrawi, A., Nguyen, T. T., Al-Kofahi, J. M., & Nguyen, T. N. (2011). Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (pp. 365-375).

Tan, P., Kumar, V., & Steinbach, M. (2006). *Introduction to data mining*. Pearson Education Inc..

Vapnik, V. (1995). *The nature of statistical learning theory*. Berlin, Germany: Springer-Verlag. doi:10.1007/978-1-4757-2440-0.

Wang, X., Zhang, L., Xie, T., Anvik, J., & Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, Leipzig, Germany (pp. 461-470).

Weiss, S., & Verma, N. (2002). A system for real time competitive market intelligence. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 360-365). ACM Press.

Weiss, S., White, B., & Apte, C. (2000a). Lightweight document clustering. [Springer-Verlag.]. *Proceedings of*, *PKDD-2000*, 665–672.

Weiss, S. M., Indurkhya, N., Zhang, T., & Damerau, F. J. (2005). *Text mining: Predictive methods for analyzing unstructured information*. Springer.

Weiss., S., White, B., Apte, C., & Damerau, F. (2000b). Lightweight document matching for helpdesk applications. *IEEE Intelligent Systems and their Applications, 15*(2), 57-61.

Wu, L., Xie, B., Kaiser, G., & Passonneau, R. (2011). BugMiner: Software reliability analysis via data mining of bug reports. In *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering*, Miami, FL (pp. 95-100).

Yin, R. K. (2009). *Case study research: Design and methods* (4th ed.). SAGE Publications.

Yu, L., Tsai, W., Zhao, W., & Wu, F. (2010). Predicting defect priority based on neural networks. In *Proceedings of the 6th International Conference on Advanced Data Mining and Applications*, Wuhan, China (pp. 356-367).

K. K. Chaturvedi is a Scientist, Computer Applications at Indian Agricultural Statistics Research Institute (Indian Council of Agricultural Research), New Delhi, India. He received M.C.A. degree from M. M. M. Engineering College, Gorakhpur, U.P. (India). He involved in the development of Agricultural Data Warehouse, Expert System on Wheat Crop and other Application Software. He has published fifteen papers in repute journals and ten papers in refereed conferences. His research interests include application of Data Warehousing, Data Mining, Mining Software and Bug Repositories and Open Source Software. Currently doing Ph.D. in Computer Science at Department of Computer Science, University of Delhi, Delhi, India.

*V. B. Singh is currently working as Associate Professor in the Department of Computer Science at Delhi College of Arts and Commerce, University of Delhi, Delhi (India). He received his M.C.A. degree from M. M. M. Engineering College, Gorakhpur, U.P. (India) and Ph.D. degree from University of Delhi. He has published more than forty research papers. His research interests include mining software repositories, empirical software engineering and software reliability engineering. He has co-authored a book "*Essentials of Computer Network Database and Internet Technology*" from NAROSA Publishing India. He is on editorial board of* International Journal of Computer Application and Journal of Software. *He has worked in different capacity in* International Conferences. *He has also received an award from Society for Reliability Engineering, Quality and Operations Management for contribution as a promising author and Researcher in the field of computer Science. He has completed major research projects from University Grants Commission and Department of Science and Technology Govt. of India. He is member of Society for Reliability Engineering, Quality and Operations Management and Association of Computing Machinery (ACM).*