An Automated Approach for Software Bug Classification

By
NEELOFAR
2008-NUST-MS PhD-CSE (E)-21
MS-8 (SE)

Submitted to the Department of Computer Engineering
In fulfillment of the requirements for the degree of

Master of Science
in
Computer Software Engineering

Thesis Advisor
Brig. Dr. Muhammad Younus Javed

College of Electrical & Mechanical Engineering
National University of Sciences & Technology

2012

In the name of Allah, the most Beneficent and the most Merciful

DECLARATION

I hereby declare that I have developed this thesis entirely on the basis of my personal efforts under the sincere guidance of my supervisor (Name of Your Supervisor). All the sources used in this thesis have been cited and the contents of this thesis have not been plagiarized. No portion of the work presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

Signature__________________

Neelofar

ACKNOWLEDGEMENTS

I am very lucky to have met my advisor Dr. Younas Javed. He offered me advice on potential research topics and helped me through my research. He was kind and always encouraging. His great talent in understanding my ideas helped direct me along the right path. He provided me with advice for, not only my research, but personal issues as well. Whenever I had issues in my life, I didn't hesitate to bring them to him. Without his guidance, it would not have been possible to perform my research and finish my M.S.

My Mom has been with me throughout my entire graduate career. She provided me with a tremendous amount of love and support. Whenever I was downhearted, she encouraged and provided me with the power to overcome my hardships.

My Friend Hufsa and my sister Huma provided me full support during the entire period of my research. With the love and cooperation of all these people I am today able to submit this dissertation of mine.

ABSTRACT

Open source projects for example eclipse and fire fox have open source bug repositories. User reports bugs to these repositories. Users of these repositories are usually non-technical and cannot assign correct class to these bugs. Triaging of bugs to developer to fix them is a tedious and time consuming task. Developers are usually expert in some particular area. For example few developers are expert in GUI and others are in pure java functionality. Assigning a particular bug to relevant developer could save time as well as would help to maintain the interest level of developers by assigning those bugs according to their interest. However Assigning right bug to right developer is quite difficult for tri-ager without knowing the actual class a bug belongs to. In this research, I have classified the bugs in different labels on the basis of summary of the bug. I have used multinomial Naïve Bayes text classifier for Classification purpose. For feature selection Chi-Square and TFIDF algorithms were used. Using Naïve Bayes and chi- square we get average of 83 % accuracy.

CHAPTER 1: INTRODUCTION TO TEXT MINING 6

1.1 INTRODUCTION 6
1.2 CLASSIFICATION 6
1.3 DIFFERENCE BETWEEN CLASSIFICATION AND CLUSTERING 7
1.4 APPLICATION DOMAINS OF CLASSIFICATION 8
1.5 CLASSIFICATION ALGORITHMS 9
1.5.1 DECISION TREE 9
1.5.2 NAIVE BAYES 10
1.5.3 GENERALIZED LINEAR MODELS 10
1.5.4 SUPPORT VECTOR MACHINE 11
1.6 CLASSIFICATION- A TWO STEP PROCESS 11
1.6.1 MODEL CONSTRUCTION AND TRAINING 11
1.6.2 PREDICTION THROUGH TRAINED MODEL 12
1.7 SUPERVISED VS. UNSUPERVISED CLASSIFICATION 12

CHAPTER 2: INTRODUCTION TO MINING SOFTWARE REPOSITORIES 14

2.1 INTRODUCTION 14
2.2 HISTORY OF MINING SOFTWARE REPOSITORIES 14
2.3 EFFECT OF OPEN SOURCE SOFTWARE 15
2.4 OTHER EARLY USES 16
2.5 MSR WITHIN SOFTWARE ENGINEERING AND PROCESS 17
2.5.1 MSR AND SOFTWARE ENGINEERING 17
2.5.2 SOFTWARE MAINTENANCE: 18
2.5.3 OTHER USES IN SOFTWARE EVOLUTION 18

CHAPTER 3: LITERATURE REVIEW 20

3.1 PREVIOUS RESEARCH 20
3.2 A FRAMEWORK FOR AUTOMATIC ASSIGNMENT OF BUGS 20
3.3 A DYNAMIC APPROACH TO SOFTWARE BUG ESTIMATION 20
3.4 BUG CLASSIFICATION IN WEB BASED APPLICATIONS 21
3.5 AUTOMATED DUPLICATE BUGS DETECTION 21
3.6 SOFTWARE ESCALATION PREDICTION WITH DATA MINING 21
3.7 AUTOMATIC BUG TRIAGE USING TEXT CATEGORIZATION 22
3.8 IF YOUR BUG DATABASE COULD TALK 24
3.9 ADAPTIVE BUG PREDICTION BY ANALYZING PROJECT HISTORY 24
3.10 MINING IN SOFTWARE ARCHIVES TO DETECT HOW DEVELOPERS WORK TOGETHER 25
3.11 FINDING CO-EVOLUTION OF PRODUCTION & TEST CODE USING MSR 25

3.12 Challenges in software evolution 26

Chapter 4: Design and Implementation 29

4.1 The Lifecycle of a Bug Report 29
4.2 Bug reports 30
4.3 Pre-processing 32
4.3.1 Stop Words Removal 33
4.3.2 Stemming 33
4.3.3 Manual Extraction 33
4.4 Feature Selection 34
4.4.1 Chi-Square Testing: 34
4.4.2 TF-IDF: 35
4.5 Classifier Modeling and Training 35
4.5.1 Naïve Bayes Training 36
4.5.2 Training steps 36
4.5.3 Model testing 38
4.6 Dry Run 39
4.6.1 Pre-Processing 40
4.6.2 System Training 42

Chapter 5: Testing and Experimental Results 49

5.1 Testing 49
5.2 Test Cases 49
5.2.1 Test Case1 49
5.2.2 Test Case 2 51
5.2.3 Test Case 3 53
5.2.4 Test Case 4 54
5.2.5 Test Case 5 55
5.3 Experimental Results 55
5.3.1 Classes having varying number of records: 56
5.3.2 Classes having small training set 58
5.3.3 Classes having refined data obtained by improved pre processing 59
5.4 Comparison with Other Systems 63
5.4.1 Classification system proposed by Lai Xu 63
5.4.2 Comparison with John Anvik, Lyndon Hiew and Gail C. Murphy 65

Chapter 6: Conclusion and Future Work 66

6.1 Conclusion 66

6.2 FUTURE WORK 67

CHAPTER 7: REFERENCES 68

APPENDIX A 71

List Of Figures

Figure 1.1 Decision tree for churn analysis of Cellular Telephony Network ___________________ 100
Figure 1.2 Model construction and training ____ 111
Figure 1.3 Prediction using trained model ____ 122
Fig. 3.1 Escalation Prediction Solution Architecture _____________________________________ 222
Figure 3.2 Architecture of JDF Tool ____ 288
Figure 4.1 Bug classification system____ 29
Figure 4.2 bug report obtained from Bugzilla bug repository _______________________________ 31
Figure 4.3 Information contained in bug reports____ 32
Figure 4.4 Data preprocessing steps and techniques_______________________________________ 33
Figure 4.5 Feature selection using Chi Square and TFID ____________________________________ 35
Figure 4.6 Naïve Bayes Training ____ 38
Figure 4.7 Flow chart for Naïve Bayes Training ____ 39
Figure4.8 Naïve Bayes testing ____ 39
Figure 4.9 Flow chart for Naïve Bayes Testing ____ 400
Figure 4.10 Tokens and their frequencies of class "build" _____________________________ __ _42
Figure 4.11 Tokens and their frequencies of class "intro"___________________________________42
Figure 4.12 frequency of each token of vocabulary V in vocabulary Textc_______________________44
Figure 5.1 Snapshot of Test Case 1____ 500
Figure 5.2 Snapshot of Test Case 1____ 511
Figure 5.3 Snapshot of Test Case 2____ 522
Figure 5.4: Snapshot of Test Case 3 ____ 533
Figure 5.5: Model building by finding probabilities of features against a class _________________ 535
Figure 5.6 Accuracy with change in Training to testing ratio _________________________________ 61
Figure 5.7 Training to testing ratio versus execution time___________________________________ 622
Figure 5.8: Comparison of Chi-Square and TFIDF____ 633
Figure 5.9 Accuracy Comparison of proposed system and one proposed by Lian Yu ____________ 644
Figure 5.10 Comparison of Naïve Bayes (2) and SVM (1) on the basis of processing time _______ 655

List of Tables

Table 4.1 Bugs and their classes ____ 39

Table 4.2 Conditional probability of all the tokens belonging to class “build” ____ 45

Table 4.3 Conditional probability of all the tokens belonging to class “intro” ____ 46

Table 5.1 Classes in training data with number of bugs ____ 57

Table 5.2 Experimental results with classes having varying number of data ____ 58

Table 5.3 Training classes with number of bugs ____ 59

Table 5.4 Experimental results with classes having small difference in record set ____ 59

Table 5.5 Accuracy with change in training to testing ratio ____ 60

CHAPTER 1: Introduction to Text Mining

1.1 Introduction

Data mining is the process of extracting useful information through data analysis. It is also known as knowledge discovery. Useful knowledge obtained as a result of data mining can be use to cuts costs, increase revenues or both. Target data for mining purpose is categorical and numerical having data types like integer, decimal, float, char, varchar2 etc.

In case one wants to mine numerical or categorical data then what should be the technique? Some examples of data other than numerical or categorical are product specifications, emails, sound files, web documents, document libraries, digital images and power point presentations etc. How to perform mining if data is descriptive such as comment fields in reports, product descriptions or call centre notes?

Data mining techniques cannot be applied to data that is not numerical or categorical. 85% of enterprise data falls in the category of non numerical or non categorical [2]. For the success of business, knowledge extraction from this unstructured data can be critical.

Unstructured data is processed using text mining techniques so that it can be processed by data mining algorithms and techniques. Techniques from information extraction, information retrieval and natural language processing are used by text mining [2]. Text mining is not a separate field from data mining but an extension/specialization of it. The source of information in data mining is numeric or categorical data which is easy to use for knowledge extraction. But in text mining, data is in the form of documents/text which requires a lot of pre-processing to make it analyzable for knowledge extraction algorithms.

1.2 Classification

Classification is a function of data mining to assign classes/categories to items in a collection. Basic goal of classification is the accurate prediction of target class for each case in data. For example, loan applications can be classified into high, medium or low risks on the basis of classification model.

For classification a dataset is required in which classes are already known. For instance, classification model for credit risk prediction would require observed data for loan applications collected over a long time. Other attributes like employment history, number of investments, type of investments, home ownership, time of residence etc are also tracked along with historical credit rating. In this example all the attributes will be the predictors, the target would be the credit rating and each customer data would be the case.

Classifications apply to discrete data. Order does not get implied in classification. Floating point, continuous values are numerical rather than categorical. Such models use regression algorithm rather than classification algorithms for prediction.

Binary classification is simplest type of classification. In this classification there are only two target values. In loan risk example, for example there are only two target values: high credit rating and low credit rating. In multiclass classification there is more than one target value like unknown, high, medium or low credit rating.

During training process (also known as model build process), classification algorithm finds the relationship between target values and predictors. Different techniques are used by different classification algorithms for finding relationship. The trained model is then applied to data set in which class assignments are unknown for classification purpose.

During testing phase, results which are predicted by classification models are compared with known target values in data set. Classification data is typically divided into two sets for classification purpose: one for the training purpose (model building) and other for testing the model.

1.3 Difference between classification and clustering

Usually clustering and classification are considered as same, but actually they are different analytical approaches. In order to differentiate between the two consider some data having customer records, where customer's attribute is represented by each record. There can be number of identifiers included like demographic information, name, financial attributes like revenue spent and income, address, age and gender etc. Clustering techniques are used to group together the related records. The grouping is done on the basis of records having similar attribute values. Clustering is often used as an exploratory process because for the analyst/end-user may not necessarily specify ahead of time how records should be related together. Basic objective of clustering is in fact to discover clusters and then examine the values and attributes that define segments or clusters. Such analysis can be used to drive promotion strategies and marketing to target specific types of customers.

There are large number of clustering algorithms available, all of which work on the process of assigning records to a cluster, calculating similarity and/or distinctiveness measure, and records reassignment to clusters until measure get stable showing that a stable segment is obtained by converging the process. Clusters are created by placing similar records in a same cluster. These records are more similar to each other and more different from the records of other clusters. Different similarity measures are used for clustering e.g. statistical variability, spatial distance etc. Overall goal of all these approaches is to converge to groups of related records.

Although classification is different technique than clustering but still there are some similarities between the two approaches. Like clustering, classification also segment records into distinct segments called classes. But unlike clustering, in classification analysis, how classes are defined is known ahead of time. For example, in customer loan applicants example discussed above the default classes would be (Yes/No). In classification training data is mandatory to build model. Training data have already defined classes (already classified). Classification is not as exploratory as clustering because each record has a value for the attribute used to define the classes. Main objective of classification is to decide how new records should be classified instead of explore the data to discover interesting segments [1].

1.4 Application Domains of Classification

Classification has number of applications in drug and biomedical response modeling, business modeling, customer segmentation, credit analysis and marketing. In some of these applications it is used as a function of data mining while in others it’s used as statistical modeling. Some of the common applications of classification are:

a. Drug discovery and development
    1. Quantitative structure-activity relationship
    2. Toxicogenomics
b. Computer vision
    1. Optical character recognition
    2. Medical imaging and medical image analysis
    3. Video tracking
c. Handwriting recognition
d. Geostatistics
e. Biometric identification
f. Speech recognition
g. Biological classification
h. Document classification
i. Statistical natural language processing
j. Internet search engines
k. Pattern recognition
l. Credit scoring

1.5 Classification Algorithms

1.5.1 Decision Tree

Decision trees are classification models in which leaves represent dataset partitions/classes while branches represent classification question. Some known algorithms of decision tree are ID3, C4.5, Cart. Figure 1.2 shows a decision tree for churn analysis of a telephonic company.

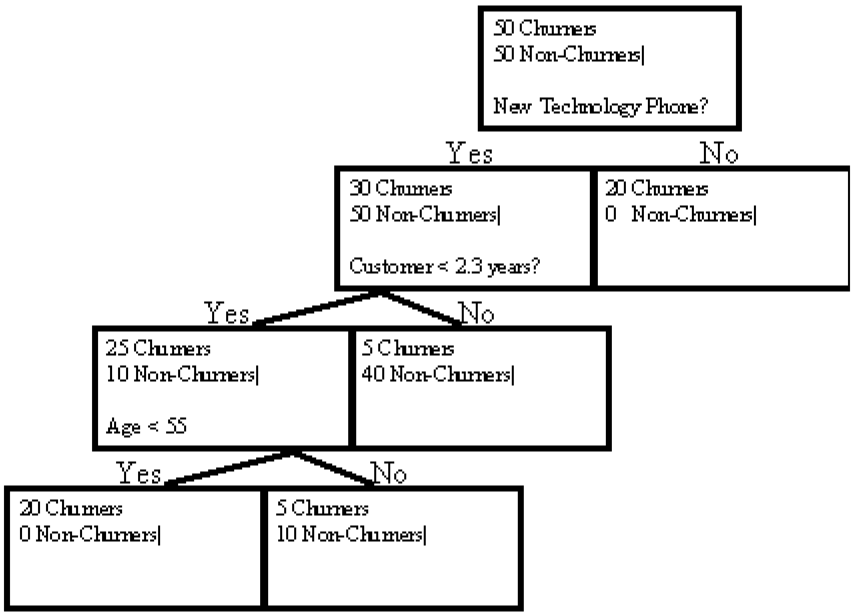


Figure 1.1 Decision tree for churn analysis of Cellular Telephony Network [30]

1.5.2 Naive Bayes

Naïve Bayes classifier is a probabilistic classification model based on Bayesian theorem with independent assumptions. “Independent feature model” would be the more descriptive form term for this model.

Independent assumption means the presence or absence of a feature of a class is independent of presence or absence of another feature. For example, an apple has features like it is red, about 4” in diameter and round. Naïve bayes classifier marks all these features to be equally contributed to the probability that this fruit is an apple.

1.5.3 Generalized Linear Models

Generalized linear model is a generalized version of linear regression that generalizes the linear model by relating it to the response variable through some link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

1.5.4 Support Vector Machine

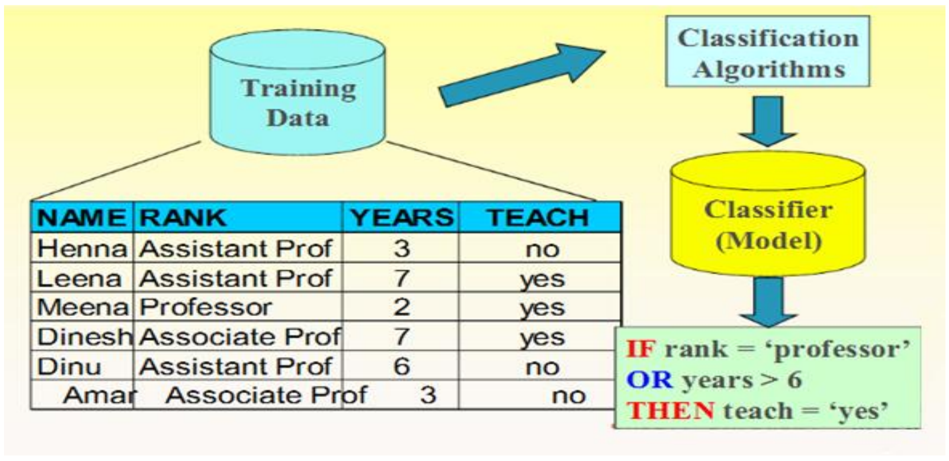
Support vector machine (SVM) analyzes data to recognize patterns which are then use for regression analysis and classification. SVM is a supervised learning method that takes a set of input data and using this it predicts the possible class against each input. Standard SVM is a non-probabilistic binary linear classifier. Given training data, in which each record belong to one of the two categories, an SVM training algorithm builds a classification model that can categorize the new unclassified records into one of the two classes.

SVM uses vector representation in which examples are mapped as points in space in such a way that examples belonging to separate categories are divided by a clear gap that is as wide as possible. Using the same space new examples are mapped and predicted to belong to one of the two categories.

1.6 Classification- A two Step Process

1.6.1 Model construction and training

Classifier model is constructed in this step using training data and classification algorithm.

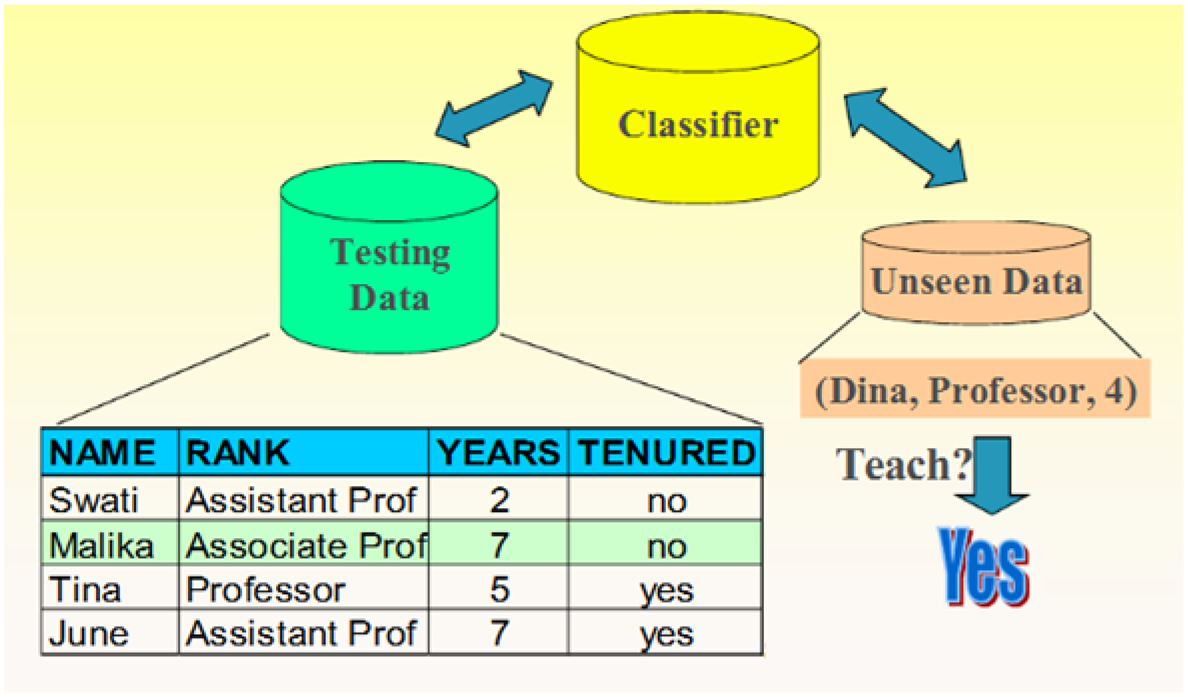


NAME	RANK	YEARS	TEACH
Henna	Assistant Prof	3	no
Leena	Assistant Prof	7	yes
Meena	Professor	2	yes
Dinesh	Associate Prof	7	yes
Dinu	Assistant Prof	6	no
Amar	Associate Prof	3	no

Figure 1.2 Model construction and training [31]

1.6.2 Prediction through trained model

Trained model is applied on data without labels. Labels/classes of data is predicted using the trained model.



NAME	RANK	YEARS	TENURED
Swati	Assistant Prof	2	no
Malika	Associate Prof	7	no
Tina	Professor	5	yes
June	Assistant Prof	7	yes

Figure 1.3 Prediction using trained model [31]

1.7 Supervised vs. Unsupervised Classification

Machine learning algorithms are divided into two categories: supervised and unsupervised. The difference is on the basis on how classification model classifies data. In supervised learning algorithms, classes are predefined. In other words certain segment of data will be labeled using such classification models. Machine learner's task is pattern searching and mathematical model construction. Accuracy of model is then measured by evaluating the prediction accuracy of model. Some examples of supervised learning algorithms are decision tree, Naïve Bayes etc.

In unsupervised learning labels are not predefined. Basic task of unsupervised learning is automatic classification labels development. Classification is done on the basis of similarity between records in such a way that they can be assigned in a group. These groups are knows as clusters.

Unsupervised learning is also known as cluster analysis. In cluster analysis prediction model is not told how texts/records are to be grouped. It’s the task of clustering algorithm to arrive at some group/cluster. In some unsupervised algorithms (K-means e.g.), number of clusters to be created is told to machine in advanced.

Chapter 2: Introduction to Mining Software Repositories

2.1 Introduction

To understand constantly evolving software systems is a very daunting task. Software systems have history of how they come to be and this history is maintained in software repositories. Software repositories are the artifacts that document the evolution of software systems. Software repositories often contain data from years of development of a software project. [4]

Examples of software repositories are:

a) Runtime Repositories: Example of runtime repositories are deployment logs that contain useful information about application usage on deployment sites and its execution.
b) Historical Repositories: Examples of historical repositories are bug repositories, source code repositories and archived communication logs.
c) Code Repositories: Examples of code repositories are Google code and codeforge.net that store source code of various open source projects. [3]

By referring these repositories, one can easily understand a piece of code. Although these repositories are a huge treasure of information about software system and software project but to extract useful knowledge from these repositories is a mess. So, the idea behind mining software repositories is to devise tools to access this wealth of information and to extract useful knowledge by analyzing them.

MSR is the process of software repositories analysis to discover meaningful and interesting information hidden in these repositories.

2.2 History of mining software repositories

Meir Lehman’s studies of IBM operating system software introduced the concept of software evolution. He discovered that software evolution was largely the result of feedback system rather than individual management decisions, and that studying the feedback process could lead to more effective software management. He identified defect reports, advances in technology, changes resulting from installation and operation of the system and changing user needs as some of the feedback mechanisms [5]. He found that it is important to study the input and output entities of feedback systems both individually and jointly.

While Lehman's studies of IBM sparked interest in investigations of software repositories, research was limited by lack of access to repositories that contained long-term rich data worth investigating. Commercial software developers, the most valuable source of data, were, in most cases, unwilling to make their repositories public. A few commercial enterprises, however, did allow researchers to access their information. [4].

Basili and Perricone [6] used data from a NASA software project to analyze the relationship between software development errors and environmental factors such as module complexity, developer familiarity with software, and whether the module was new or modified. Mockus, Weiss, and Yang [7] used MSR with version control and problem tracking repositories from Avaya Labs to predict how much effort would be needed to repair problems during development or after release of a software project. ATT, Nortel, Nokia, and Mitel also made their software databases available to MSR researchers. However, it was the advent of open source software that resulted in access to the large, varied, and long-term software repositories needed for productive MSR research. [4]

2.3 Effect of Open Source Software

Two of the most popular open source version control systems are CVS (Concurrent Versioning System) and SVN (Subversion). These two repositories are basically the source code repositories that track changes in source code and maintain such information as well like who made this change, when was the change made, at which revision a particular file is committed etc. Bugzilla and jira are bug repositories. Each bug report contains a summary description of the bug, when it was reported, its severity and priority, operating system in which it was reported, its current status etc. Communication repositories include project emails, chats and other archived communications between developers working on a project. Runtime repositories containing deployment and execution audit data, source forge and

Google code that are centralized repositories are some other open source repositories available to MSR.

Open source software repositories offer a wealth of information but to extract meaningful information from them is not easy due to lack of integration. Code of the project might move to SVN, the developers of the same project discuss this code with other developers via email and bug tracking tools contain the bug reports of the same project. Although all the necessary information of the project is stored in these repositories but these repositories are not integrated to each other and there is no way to retrieve all of developer's activities or to track artifacts across different projects.

The Hipikat tool, developed by Cubranic et al [8], is one attempt to overcome the problem of non-integrated repositories in MSR research. Cubranic et al addressed the problem that arises in a situation when a new software developer joins a team that is already working on any project and new developer must be trained to speed up the project. When members of the development team are in the same location, a senior developer can explain the intricacies of the project to the new developer and provide feedback and advice. However, when the team is a virtual team, this type of support is not available. Hipikat is an MSR tool that accesses the entire project memory source code, bug tracking, communications, and project documents by establishing relationships between the artifacts. Hipikat establishes these relationships by combining information from different sources, or by inferring relationships based on meta-information contained within the databases. A major advantage of Hipikat is that it builds the project memory automatically and does not require any significant changes in existing work practices. [8]

2.4 Other Early Uses

Some other uses of MSR which were worked out in its early life were bug identification and prediction, and finding the code that can be reused by developers for their need. CP-Miner is a tool, developed by Li and Lu that can identify copy-paste code and copy-paste bugs in large software systems. [9] Copy-paste approach is used in large software applications to reuse code and thus to reduce development effort but this strategy might be the source of bugs in the system because developers forget to modify the identifiers in such code. Mandelin et al developed the Prospector tool to help developers search repositories to find specific code for reuse. The user submits a simple query describing the desired input and output, and Prospector automatically returns a candidate list of code snippets. [10]

Although the early uses of MSR showed valuable contributions of this field but still the need of the time was further research in this field to create tools and techniques to help analyzing these repositories and extracting meaningful information from them. A one day international Workshop on MSR was held in Edinburg, Scotland in 2004. It continued to be an annual event expanding to a Workshop Conference in 2008.

2.5 MSR within Software Engineering and Process

2.5.1 MSR and Software Engineering

There is a huge Software Engineering data over the course of time. MSR picks this data, processes and analyzes it, and detects patterns in this data. MSR is an open field, both in what can be mined and what one can learn from the practice. Any software repository can be mined not necessarily the code, bug or archived communication repositories. One prize winning study in MSR analyzed the use of IRC channels used as a source of meetings between developers developing open source project. In that project, chat logs were mined to find the information such as when was the meeting held, which developers attended the meeting, what was discussed and most important whether the meeting was useful or not.

Another important source of information is archive communication repositories such as emails which were used as a source of communication between developers. Similarly, bug repositories can be used along with the source code repositories having change logs, for instance, comments in CVS and summary given by the user while reporting bug can be used for source code changes categorization as an attribute of corrective maintenance activity. Another major repository to be mined is source code repository. Much creative information can be mined from this repository, for example, using source code as a communication tool, one analyst conducted a research in which he used the comments (e.g. TODO tag) as a source of communication. Such comments are termed task comments. Although this is not a good source of information like communication archives but at least it shows a company it’s high time to have a communication tool as code comments are hardly an ideal way to communicate.

2.5.2 Software Maintenance:

These repositories track software changes by managing software evolution. Software change is defined as addition, deletion, or modification in software artifacts. Most important artifact in software life cycle is source code change. All other artifacts are maintained to manage and track these code changes [11]. These repositories grow larger and larger with the evolution of software and thus these are the best part where MSR can be applied. Software maintenance is the most expensive part of software life cycle and thus MSR techniques prove to be extremely valuable by benefiting this piece of process. Code cloning is an inexpensive way to reuse existing code. However, analyzing the software repositories which use this technique, it is found that using this technique makes the maintenance of these software products quite complicated and thus too much clone code is a risk.

MSR techniques are very effective in predicting faults. Gall worked on common semantic dependencies between source code classes due to modification or addition of a class, on the basis of its version history. Thus, by using the repositories, one can see the coupling and dependency between classes or other piece of codes and thus help in maintenance of system by seeing what other components might affect by changing the particular component. In fact, fault finding and predicting is probably the largest application of MSR. A number of tools are created for this purpose.

2.5.3 Other Uses in Software Evolution

By using MSR, developers and architects can have an insight into the design of the system that would be impossible otherwise. To create a software using visual representation is one of the most confusing and difficult tasks in software engineering. This is crucial when dealing with legacy systems or during reverse engineering. Mining repositories can help with this issue in a number of ways.

Eclipse repositories were mined to develop author topic models to help developers find the experts on the given part of the system. Author-Topic modeling (AT) captures the relationship of author and topic. Of course, general reverse engineering would be a lofty goal. Some information is not available in software repositories such as architecture models and designs. They are required to be reproduced using reverse engineering to support MSR questions. This sort of information is essential to software evolution as component and subsystem reuse is seen by many to be the best way to decrease the time required to bring a software product to completion. By mining the repositories available, one can certainly gain valuable architectural insights though.

Chapter 3: Literature Review

3.1 Previous Research

An extensive research has been conducted to learn the techniques and algorithms which are already being used in field of MSR. Some of these techniques are mentioned in this section.

3.2 A framework for automatic assignment of bugs

Manual bugs triaging i.e. bugs assignment to individual developers for fixation is a time consuming and tedious task. Micheal W. Godfrey, Olga Baysal and Robin Cohen presented a framework for automatic assignment of bugs to developers for fixation [12]. The approach presented by them employs preference elicitation to learn developer predilections in fixing bugs. The knowledge about developer's expertise is inferred by analyzing the bugs history fixed by the developer. When a new bug report arrives, using vector space model, system automatically assigns it to the appropriate developer considering developer's expertise, preference and workload. He addresses the task allocation problem by proposing a set of heuristics that support accurate assignment of bug reports to the developers.

3.3 A Dynamic Approach to Software Bug Estimation

Outsourcing is a common trend in today's software market. Project development in globally distributed environment is increasing day by day. However, to manage these globally distributed projects and their resources is much harder. Hemant Josh, Chuanlei Zhouang, Oskum Bayrak presented a methodology to predict future bugs using history data. This information can be used for support management and resource planning in distributed projects. Their algorithm works in a two step analysis mode: Global and Local. Global analysis, for each component, finds the counts of bug over time while local analysis analyzes the past history of a bug. Results were compared by eclipse software data and bug prediction was very close to the actual bug count [13].

3.4 Bug Classification in Web Based Applications

Popularity of web based applications is increasing day by day and they are changing life styles of people. Dependency of people on web based applications is an increasing trend. However, web based applications development is still a very challenging task, depicted by hundreds and thousands of bugs reported daily in bug reporting and tracking tools. Lei Xu, Lian Yu, Jingtao Zhao, Changzhu Kong, and HuiHui Zhang proposed an algorithm using data mining techniques that automatically classifies the bugs of web-based applications by predicting their bug type. They further proposed debug strategy association rules which find the relationship between bug types and bug fixing solutions. Debug strategy is built based on what was the bug, and what was the most effective solution given by the developer to fix it [14].

3.5 Automated Duplicate Bugs Detection

In software evolution, bug tracking tool is very important to record the software maintenance activities and bugs and problems in a system. However, in open source bug tracking systems like mentis and bugzilla, a number of duplicates are reported which hamper the utility of these bug tracking systems. Sometimes, as many as one fourth of all the reports in a project are duplicates. Triagers and developers identify duplicate bug reports manually which is a time consuming and high cost process in terms of project management and maintenance. Nicholas Jalbert and Westley Weimer proposed a system that automatically indicates whether an arriving bug report is original or duplicate of an already existing report. It saves developer's time. To predict bug duplication, system uses textual semantics, graph clustering and surface features. In their experiments, they used bug data from Mozilla bug repository and included almost 29,000 bug reports. Experimental results show that the system was able to filter out 8% of duplicate bugs and thus reduces the development cost [15].

3.6 Software Escalation Prediction with Data Mining

Defect escalation is a term used for significant impact of a defect on customer's operations. Defect escalation has a very poor impact on software user in terms of software quality. These defects are then tried to be fixed as quickly as possible, at a high cost, outside the general product release engineering cycle. Even if the defects are reported by software vendors and customers before they are escalated, to prioritize them quickly and accurately for resolution is not always possible. Inaccurate prioritization may lead to escalation of defects, even previously known and reported. Apart from unknown defects, escalation of known defects amounts to millions of dollars per year, labor cost along with the loss of reputation, loyalty, satisfaction and repeat revenue due to inaccurate prioritization. Tilmann Bruckhaus provided a technique for Escalation Prediction (EP) to avoid escalations by predicting the defects that have high escalation risk and then by resolving them proactively[16].

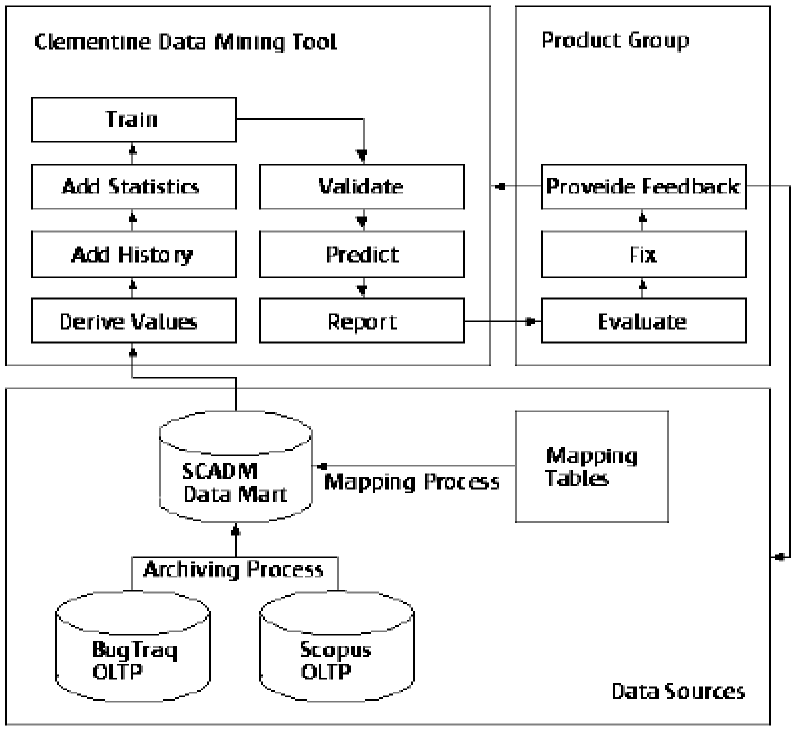


Fig. 3.1 Escalation Prediction Solution Architecture

3.7 Automatic bug triage using text categorization

For the management of bug reports and resources to fix these bugs, a bug tracking system is required by large software development projects. One example of such system is Bugzilla, which was first introduced in the development of Mozilla as an open source bug tracking system but now used in a number of other projects as well. In open sources software development projects, team members are usually dispersed around the world. Developers, project managers and other resources rarely see each other. Bug tracking systems are especially important in management of such open source large software projects. In such projects, bug tracking system is not just a tool for reporting and tracking bugs but also to coordinate work among developers.

An important section in bug reports is “Additional Comments”. Most bug tracking systems provide the facility to add comments to bug reports. This feature is really helpful in geographically and time dispersed software projects and can help to fill a slot for issue specific, focused discussion. For implementation details these comments serve as forum. All the team members like developers who due to their expertise and insight can help in design deliberations and all the stakeholders whose code will be affected by the modifications which are proposed are included into discussion through these bug reports. Users having interest in quick fix of the bug can also join in.

To deal with new bug reports and their fixation as quickly as possible is very important. If the developers ignore the bugs reported by a user or new features demanded by customers, it can kill the overall repute and turn the users away. In large open source software projects triaging bug to developers is a difficult task both due to a very large number of bugs reported daily and duplicate bugs. One who can best identify the bug-whether it’s a real bug or duplicate is the developer. However, it would be too much time consuming and would be a burden for a developer. Therefore, in large open source projects such as Eclipse and Mozilla a separate team member is dedicated to triaging. It’s not an ideal solution as it would introduce more delays and potential errors in case of wrong decision by triager of assigning which report to which developer. Davor and Gail presented a technique using machine learning, and in particular text categorization, to “cut out the triageman” and automatically assigns bugs to developers based on the description of the bug as entered by the bug’s submitter. They used multinomial Naïve Bayes algorithm for the prediction of duplicate/Real bug. Prediction accuracy was 30% when training to testing ratio was 9:1 [17].

3.8 If Your Bug Database Could Talk

Bug databases list all the bugs and problems that arose during software development life cycle and thus are most consistent sources for failure information. But the bug databases are not too descriptive to record how the problem arose, what the impact area was and who fixed it. Code repositories, archived communications, deployment logs etc. contain this information. Using these entire databases, one can find the relationship between bug and fixes. Fixes are related to locations. Using this relation Thomas Zimmerman, Adrian Schroter, Andeas Zmmeller and Rahul Premraj determined the density of defects in a component by counting the applied fixes. They used Eclipse programming environment code base and worked on the bugs that were reported in first six months of project development [18]. The research questions illustrated by them are:

a) Is it possible to use code complexity to predict failure proneness?
b) Is there any relationship between bugs' number after release and during testing?
c) Are there more errors in the code of some developers than others?

Using the code repositories and bug repositories one can find the change in the code that introduces a bug and one that fixes it. Using this information one can predict the future bugs.

3.9 Adaptive bug prediction by analyzing project history

Sunghun Kim presented two bug prediction algorithms to analyze a project's change history: Bug cache and Change classification. Bugs cache approach works on the assumption that bugs do not occur in isolation rather in a burst of group of related bugs. A developer can find the more error/bug prone areas of the project by using the bug cache. It would help in allocating more resources to more error prone areas of a software system. Sunghum Kim used two machine learning algorithms in his research, Support vector Machine and Naïve Bayes and used 10% of the code files from seven open source projects.

The change classification approach classifies bug changes with 65% buggy change recall and 78% accuracy. Both approaches can be used to find locations of bugs by leveraging project history and learning the unique bug patterns. This information can help to reduce software development cost and to increase software quality [19].

3.10 Mining in software archives to detect how developers work together

Complete history of open source projects is available in open source software repositories. SUBVERSION and CVS store all the committed versions of code files that have existed during software development. It contains the commit revision numbers of files in addition to information like which developer has committed the file and when. This information is very important in open source projects as developers and other team members are locally separated.

Peter Weigerber, Mathias Pohl and Michael Burch examined the artifacts changed by developers, which developers and when. They searched the following questions in their research

a. Do files or modules are worked on by just one developer or a number of developers work on single module or file?
b. What's the hierarchy of developers in a big project? Is there a main developer and other helper developers or rather work is equally distributed among all the developers?
c. Are there phases during the evolution, when there is a very active development and ones when there is hardly any development? [20]

3.11 Finding Co-Evolution of Production & Test Code using MSR

Number of artifacts are created and maintained during software systems engineering. Bart Van Rompaey, Arie Van and Andy Zaidman investigated whether there is some correlation between test code and production code by using the information contained in versioning systems, size metrics and code coverage reports. Main purpose of the research was to create mutual consciousness among managers and developers about testing process.

They evaluate their results both with the help of log-messages and the original developers of the software system [21].

3.12 Challenges in software evolution

Software evolution is necessary and an ongoing activity to keep it up to date otherwise it becomes ineffective. Most important piece of work that needs to be considered in evolution is source code but it is not the only thing that matters. Software is multidimensional and also the development process behind it. To develop quality source code, other artifacts like tests, specifications, constraints, documentations are needed as well [22].

Bart Van Rompaey, Arie van Deursen, Andy Zaidman and Serge Demeyer worked on two dimensions i.e. source code and test. They analyzed the evolution of tests as source code evolves. To analyze the co-evolution of production and test code, they used the data stored in versioning control system. But for using VCS for this purpose, it was necessary that source code and test code must be committed. Therefore, their main focus is the co-evolution of production code against constant software tests i.e. integration test and unit tests. They understood that, to get best result, production code and test code should be developed and maintained side by side, for at least two reasons:

a. New functionality should be tested as soon as soon as it gets completed e.g. via unit testing [23].
b. When changes or up gradation is applied, the old code preservation (in terms of its behavior) should be checked.

Here, Moonen et al. have shown that refactoring itself does not change the behavior of the system but, they invalidate tests [24]. Elbaum et al. concluded that even very small changes in source code can have serious effect on test reporting or the fraction of production code tested by the test suite [25]. These observations support the argue that updated source code and test code need to co-evolve. This leads to the inconsistent situation where tests are necessary for the success of the software (and its evolution), while also an important consideration during maintenance. It is exactly this inconsistency that enforced them to study the co-evolution of test code and production. They proposed to use lightweight techniques and visualizations, which are common to the field of studying software evolution [26].

For this study, their major concern was: How does testing occur in open-source software systems? In order to guide their research, they improved this question into other supplementary questions:

a) Does co-evolution occur phrased of synchronously?
b) When should the test writing effort be increased? Right before release or some other phase of SDLC?
c) Is it possible to detect test strategies, test-driven development for instance [27]?
d) What is the relation, if any, between test coverage and test writing effort?

To conclude, they performed an experiment to study the co-evolution of test code and production of two open source softwares. They evaluated their research internally and externally, by studying log messages (written during development) and by sharing their findings with the developers and recording their remarks.

Both users and the developers submit bug report to repository. These reports play important role in revealing defects and improving software quality. Increase in number of bug reports in repository will potentially increase the number of duplicate bug reports. Detecting duplicate reports will reduce effort in fixing that particular bug or defect. However, it is a big challenge to detect all duplicate bug reports out of a very large number of existing bug report. Tao Xie, Lu Zhang , Xiaoyin Wang , Yoonki Song and Hong Mei presented ‘Jazz Duplicate Finder’ , a tool that helps to identify duplicate or repeated bug reports on Jazz, which is a teamwork stage for software development and management. Natural language and Execution information was used to find duplicates in bug report. [30]

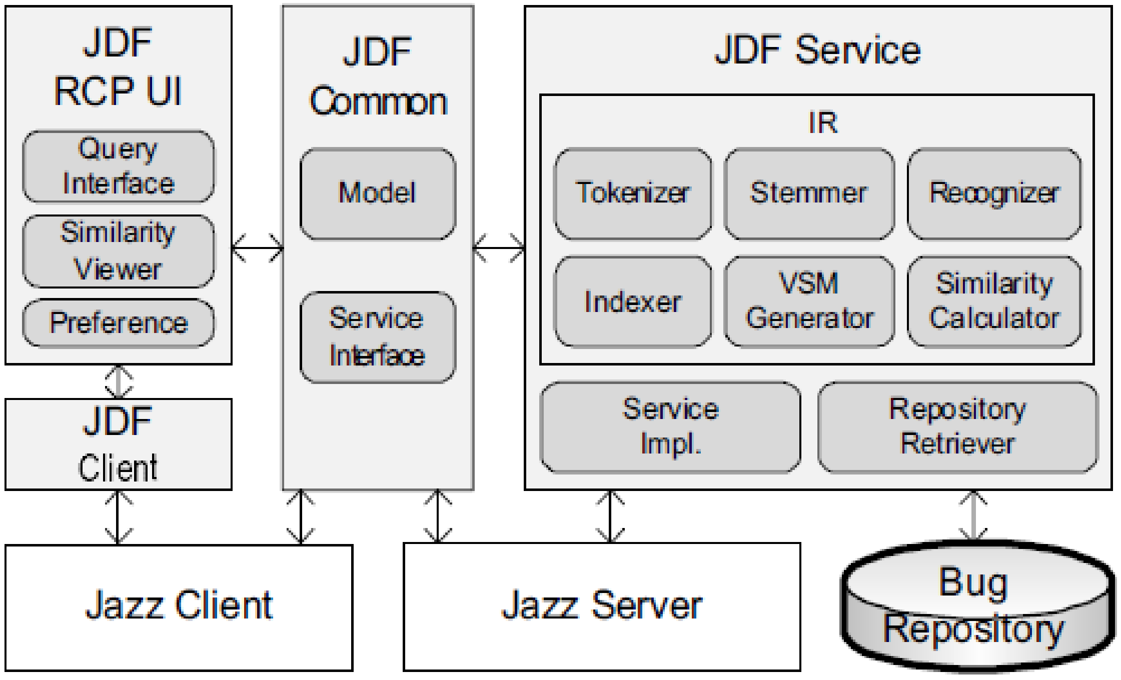


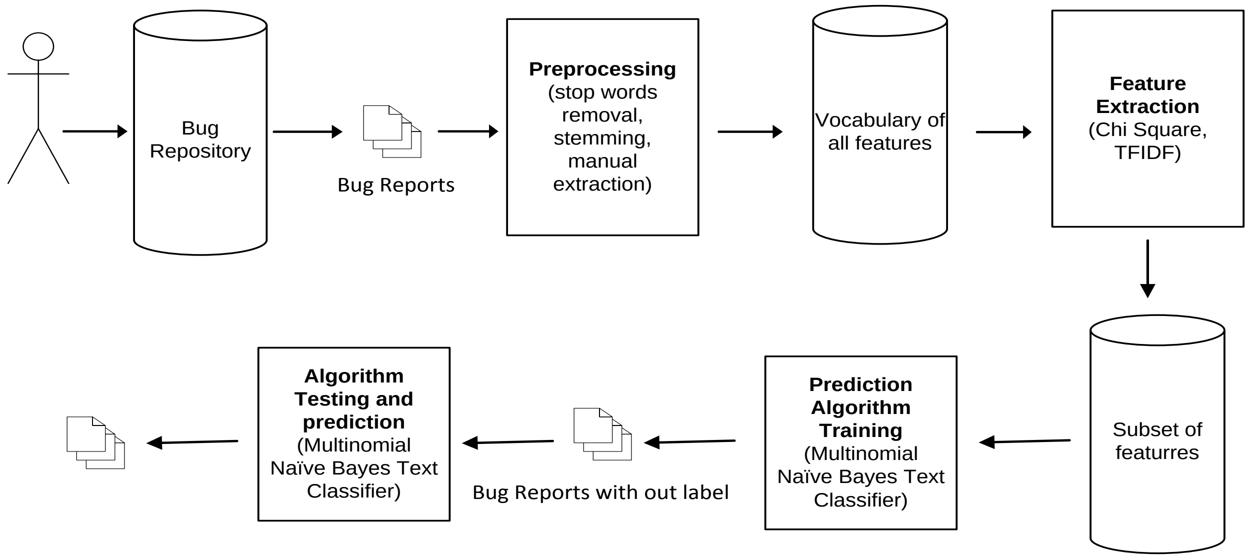
Figure 3.2 Architecture of JDF Tool

Chapter 4: Design and Implementation

4.1 The Lifecycle of a Bug Report

There are a number of states of bugs before they get fixed. When the report is first submitted, its state is NEW. The bug is then assigned to developer to fix it. Its status changes to ASSIGNED. Developer fixes the bug and it is then marked as RESOLVED. There are a number of ways to resolve a report; if the resolution results in a change in the code, its status becomes RESOLVED otherwise DUPLICATE. If the developer cannot reproduce the bug, it is marked as “WORKSFORME”. Report is marked as INVALID or WONTFIX if it’s not an actual bug. Report can be reopened that was formerly closed and its status changes to REOPENED.

When the bug is first reported to repository, it is submitted to our proposed system as shown in Figure 4.1. System extracts all the terms in these reports using bag of words approach. The vocabulary is of a very high dimensionality and thus numbers of features are reduced by using chi-square algorithm. These features are used for training of classification algorithm which is then used for classification of bug reports. The classification algorithm used in proposed system is multinomial Naïve Bayes. Details of all these modules and processes used in proposed classification system are given in sub sections below.



Bug Repository
Bug Reports
Preprocessing (stop words removal, stemming, manual extraction)
Vocabulary of all features
Feature Extraction (Chi Square, TFIDF)
Subset of featurres
Prediction Algorithm Training (Multinomial Naïve Bayes Text Classifier)
Bug Reports with out label
Algorithm Testing and prediction (Multinomial Naïve Bayes Text Classifier)

Figure 4.1 Bug classification system

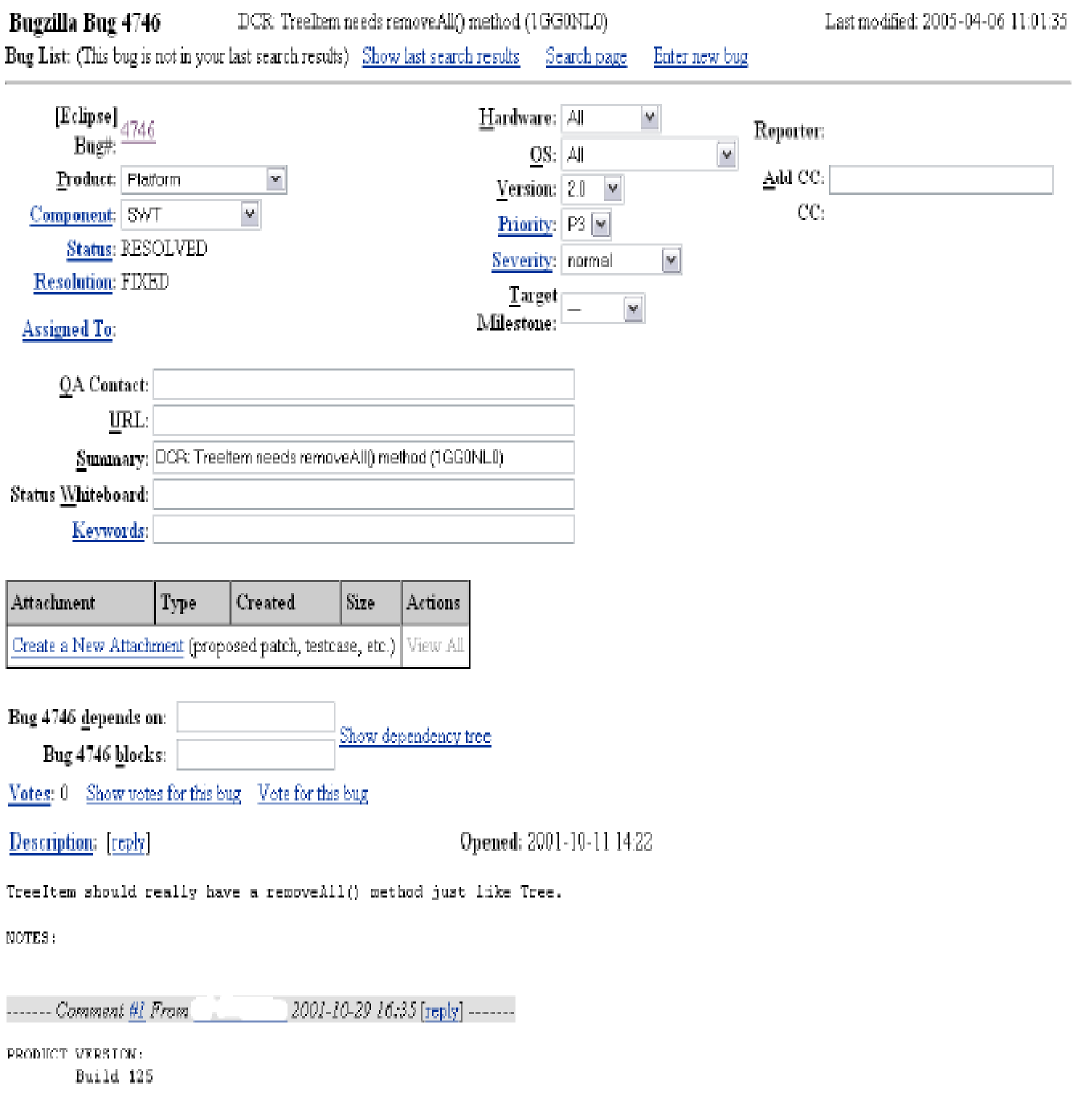
4.2 Bug reports

Eclipse bug repository is used in this research to get bug data. Figure 4.2 shows the information contained in these bug reports.

a. **Bug_id**: unique identifier assigned to each bug
b. **Component:** the component of the product in which this bug is being reported like ant, CVS etc.
c. **Severity:** severity of bug like normal, major, enhancement etc.
d. **Status:** current status of the bug like open, closed, fixed etc.
e. **Operating system:** operating system in which this bug is reported like windows, fedora etc.
f. **Summary:** brief summary of the bug explained by reporter.

The most important attribute of bug report is SUMMARY. Reporter submits the bug by giving its brief summary. Summary briefly discusses the scenario in which this bug reproduces. It also describes impact areas of the bug. The data from summary of the bug is taken and algorithm is trained over it. Whenever a new bug arrives, proposed system classifies it using this summary field.

The challenge associated with using the summary to classify bugs of open bug repositories is that the bugs are submitted by ordinary users of products, not the technical people. For instance, user of Mozilla firefox might be a non technical person who cannot report the bug using some predefined technical terms to specify the bug. Classifying these bug reports using summary data is a challenging task that requires data to be thoroughly pre processed first. Classification algorithms cannot differentiate between “browser”, “firefox” or “Mozilla firefox”. For this purpose, synonym dictionary might be helpful and could improve the results but it would increase the scope of this research and thus synonym dictionary integration in proposed system is one of the proposed future work.



Bugzilla Bug 4746 DCR: TreeItem needs removeAll() method (1GG0NL0) Last modified: 2005-04-06 11:01:35

Bug List: (This bug is not in your last search results) Show last search results Search page Enter new bug

[Eclipse] Bug#:	4746	Hardware:	All	Reporter:	
Product:	Platform	OS:	All	Add CC:	
Component:	SWT	Version:	2.0	CC:	
Status:	RESOLVED	Priority:	P3		
Resolution:	FIXED	Severity:	normal		
Assigned To:		Target Milestone:	---		

QA Contact:

URL:

Summary: DCR: TreeItem needs removeAll() method (1GG0NL0)

Status Whiteboard:

Keywords:

Attachment	Type	Created	Size	Actions
Create a New Attachment (proposed patch, testcase, etc.)				View All

Bug 4746 depends on: Show dependency tree

Bug 4746 blocks:

Votes: 0 Show votes for this bug Vote for this bug

Description: [reply] Opened: 2001-10-11 14:22

```
TreeItem should really have a removeAll() method just like Tree.

NOTES:
```

------- Comment #1 From 2001-10-29 16:35 [reply] -------

```
PRODUCT VERSION:
        Build 125
```

Figure 4.2 bug report obtained from Bugzilla bug repository

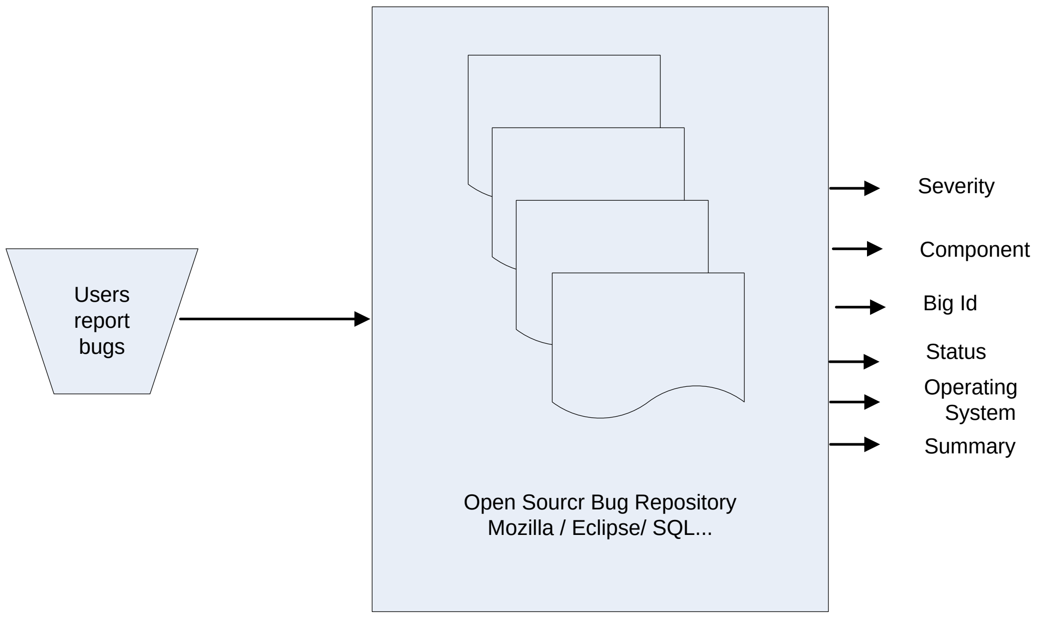


Figure 4.3 Information contained in bug reports

4.3 Pre-processing

Data pre-processing is the most important step of data mining. Data obtained from bug repositories is in raw form and cannot be directly used for training the classification algorithm. The data is first pre-processed to make it useful for training purpose. Data pre-processing is the most time consuming step of data mining and most important as well. I used stop-words dictionary and regular expression rules to filter useless words and filter the punctuations respectively. I applied porter stemming algorithm to stem the vocabulary but stemming did not prove to be of any worth in case of textual data. It in fact further deteriorated the results and decreased the accuracy.

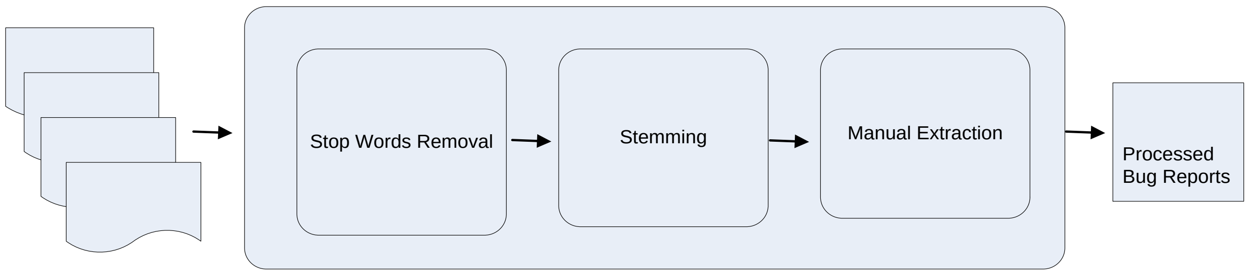


Figure 4.4 Data preprocessing steps and techniques

4.3.1 Stop Words Removal

A deep study of the stop words dictionary available on internet revealed that stop words removal is not generic but it is actually domain specific. Some terms are considered to be stop words in one domain but they carry useful information in another domain. In this research, the reporters of bugs are non technical users. Users report the bug using simple non technical English words. So, the stop words dictionary used contains language dependent as well as domain dependent stop words.

List of stop words that are being used in this research is given in appendix A.

4.3.2 Stemming

Stemming is defined as the process of reducing words to their base/stem in linguistic morphology. Stemming programs are commonly known as stemmers or stemming algorithms.

In this research, porter stemming algorithm is used. Source code for porter stemmer is available on its official website. [29]

4.3.3 Manual Extraction

Proposed system uses multinomial Naïve Bayes algorithm for classification. Naïve Bayes algorithm is based on probability. Final forecasting of a bug class is performed on the basis of product of posterior probability as well as the prior probability of class. Use of classes having very large data difference (difference on the basis of number of records) is not feasible for algorithm training. Prediction will tilt towards class having largest record set. To avoid this condition, manual selection of classes-not having very large difference in record set- is performed for training data.

4.4 Feature Selection

The vocabulary obtained after applying “bag of words” approach on data has very large dimensionality. Most of these dimensions are not related to text categorization and thus result in reducing the performance of the classifier. To decrease the dimensionality, the process of feature selection is used which takes the best k terms out of the whole vocabulary which contribute to accuracy and efficiency. Feature selection has two main advantages:

a) Algorithm training becomes more efficient due to reduction of dimensionality of vocabulary.
b) By reduction of rare terms classification accuracy increases.

There are a number of feature selection techniques such as Chi-Square Testing, Information Gain (IG), Term Frequency Inverse Document Frequency (TFIDF), and Document Frequency (DF). In this research, chi-square and TFIDF algorithms are used for feature selection.

4.4.1 Chi-Square Testing:

Chi-Square test for independence is used for feature selection. It is used to determine the relationship between two variables. “Independence” means that the two variables are not related to each other. Chi-Square is defined as

$$x^2(t,c)=\frac{N(AD-CB)^2}{(A+C)(B+D)(A+B)(C+D)}$$

Equation 4.1 chi square test for feature extraction

In equation 4.1 A, the total number of documents in class c containing term t, B is the total number of documents not belonging to class c but containing term t, C is the total number of documents belonging to class c containing term t, D is the number of documents not in c not containing t.

4.4.2 TF-IDF:

TFIDF-Term Frequency-Inverse Document Frequency is a term weighting technique that is used to evaluate the importance of word in a collection of corpus. It assigns the weight to a term in the document given by

$$tf - idf_{t,d} = tf_{t,d} \times idf_{t-}$$

Equation 4.2 TF-IDF technique for feature selection

Thus TF-IDF assigns weight to a term in a document. Weight is

1. Highest when term occurs frequently within a small number of documents.
2. Lower when the term occurs fewer times in a small number of document, or occurs in a lot of documents.
3. Lowest if the term occurs in virtually all documents.

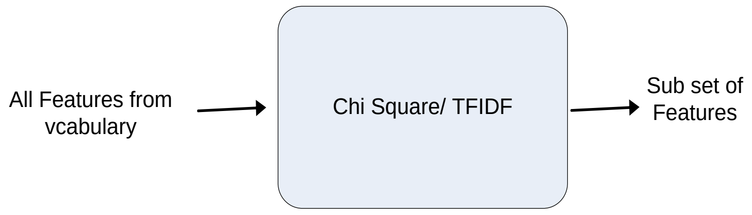


Figure 4.5 Feature selection using Chi Square and TFID

4.5 Classifier Modeling and Training

Text classification is an automated process of finding some metadata about a document (in this research the term “document” is alternatively used for “bug”). Text classification is used in various areas like document indexing by suggesting its categories in a content management system, spam filtering, automatically sorting help desk requests etc.

Naïve Bayes text classifier is used in this research for bug classification. Naïve Bayes classifier is based on Bayes' theorem with independent assumption and is a probabilistic classifier. INDEPENDENCE means the classifier assumes that any feature of a class is unrelated to the presence or absence of any other feature.

4.5.1 Naïve Bayes Training

The probability of a document d being in class c is computed as

$$P\left(c\middle|d \propto P(c)\, \pi_{1\le k\le n_d}\, P(t_k\,|C)\right)$$

Equation 4.3 Conditional probability of term in given document

P(t_k|c) is a measure of how much evidence t_k contributes that c is the correct class. P(t_k|c) is the conditional probability of term t occurring in a document of class c. P(c) is the prior probability of a document occurring in class c.

$$P(c) = Nc$$

$$\hat{P} = \frac{N}{\acute{N}}$$

Equation 4.4 Prior probability of a class

If a document's terms do not provide clear evidence for one class versus another, we choose the one that has a higher prior probability.

4.5.2 Training steps

a) Features from all the documents are extracted and a vocabulary is created.
b) Count the total number of documents in training set. (Steps c to h are repeated for all the classes).
c) Count the number of documents in given class.
d) Find the prior probability of a class by dividing documents in a given class to the total number of documents in the whole training set.
e) Create another vocabulary having the terms of this class only. (Repeat step f to h for every term in vocabulary).
f) Find the frequency of term in vocabulary created in step e.

g) Find the conditional probability of each term using equation 4.3.

h) Return vocabulary obtained in step a, prior probability of class and conditional probability of term

```
                 $TRAIN\boldsymbol{M}ULTINOMIAL\boldsymbol{NB}(\boldsymbol{C},\boldsymbol{D})$
1 . $V \leftarrow EXTRACTVOCABULARY(D)$
2.  $N \leftarrow COUNTDOCTS(D)$
3. for each c $\in C$
4. do $N_c \leftarrow COUNTDOCSINCLASS(D,C)$
5.   prior[c] $\leftarrow N_c \ / N$
6.   $text_c \leftarrow CONCATENATETEXTOFALLDOCSINCLASS\ (D,C)$
7.   foreach $t \in V$
8.   do $T_{ct} \leftarrow COUNTTOKENOFTERMS\ (text_c, t)$
9.   foreach $t \in V$
10.    do $condprob[t][c] \leftarrow \frac{\boldsymbol{t_{ct}+1}}{\boldsymbol{\epsilon(t_{ct}+1)}}$
11.    return $V, prior, condprob$
```

Figure 4.6 Naïve Bayes Training

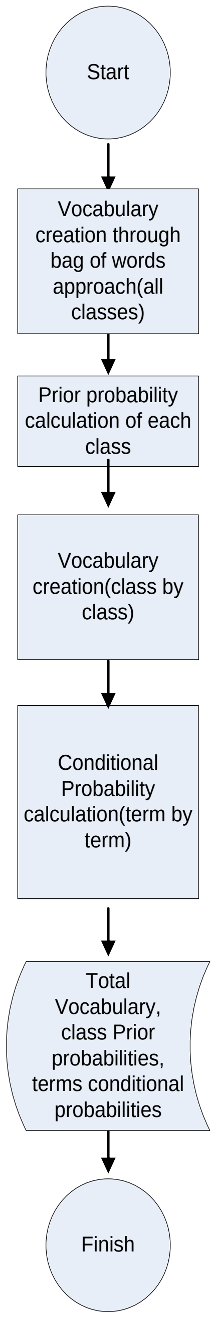

Start
Vocabulary creation through bag of words approach(all classes)
Prior probability calculation of each class
Vocabulary creation(class by class)
Conditional Probability calculation(term by term)
Total Vocabulary, class Prior probabilities, terms conditional probabilities
Finish

Figure 4.7 Flow chart for Naïve Bayes Training

4.5.3 Model testing

In order to predict the class of a bug/ document, following steps are applied during prediction phase of classifier.

a) Tokens/features from bug are extracted and maintained in a vocabulary.

(Repeat steps b to d for all the classes).

b) Find the log of prior probability of the class.

(Repeat steps c to d for all terms obtained from document)

c) Find the conditional probability of term to be a part of this class.

d) Score the class on the basis of product of probabilities obtained in step b and c.

```
$APPLY\boldsymbol{MULTINOMIALNB}(\boldsymbol{C}, \boldsymbol{V}, \boldsymbol{prior}, \boldsymbol{condprob}, \boldsymbol{d})$
1 . $W \leftarrow EXTRACTTOKENSFROMDOC(V, d)$
2. for each $c \in \boldsymbol{C}$
3. do $score[c] \leftarrow \log prior[c]$
4.     for each $t \in W$
5.     do $score[c] += \log condprob[t][c]$
6.   return $argmax_{c \in \boldsymbol{C}}, score[c]$
```

Figure4.8 Naïve Bayes testing

4.6 Dry Run

Algorithm is trained on two classes having three bug reports each. Bug reports and their classes are given in table 4.1

Bug Summary	**Class/label**
1. Can't cancel build project from progress view.	Build
2. Could create build path error in case of invalid external JAR format.	Build
3. Incremental build involving a resource filter fails to produce expected subdirectory of the output folder.	Build
4. Intro crashes when uses a custom JAXP parser.	Intro
5. Welcome leaks handles.	Intro
6. Welcome view shows nothing when opened via ctrl+3	Intro

Table 4.1 bugs and their classes

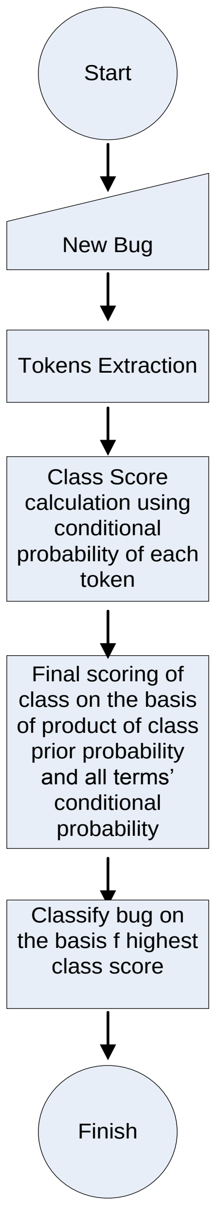

Start
New Bug
Tokens Extraction
Class Score calculation using conditional probability of each token
Final scoring of class on the basis of product of class prior probability and all terms' conditional probability
Classify bug on the basis f highest class score
Finish

Figure 4.9 Flow chart for Naïve Bayes Testing

4.6.1 Pre-Processing

4.6.1.1 Stop Words Removal

Following words are considered as stop words and are removed from vocabulary.

a. Can't

b. From
c. Could
d. In
e. Of
f. To
g. The
h. Each
i. I
j. When
k. A
l. Via
m. In

4.6.1.2 Stemming

Following words are stemmed to their root using porter stemming algorithm

a.	Incremental	→	increment
b.	Involving	→	involve
c.	Fails	→	fail
d.	Crashes	→	crash
e.	Leaks	→	leak
f.	Handles	→	handle
g.	Shows	→	show

4.6.1.3 Tokenization and frequency calculation

Bug summaries belonging to both classes are tokenized and frequency of each word/token is calculated against each class. List of tokens and their frequencies of bug summaries belonging to build and intro class are given in figure 4.5 and figure 4.6

Token	Cancel	build	project	View	create	Path
Frequency	1	3	1	1	1	1

Token	Error	Case	Invalid	External	Jar	Format

Frequency	1	1	1	1	1	1

Token	Increment	Involve	Resource	Filter	Fail	Produce
Frequency	1	1	1	1	1	1

Token	Expected	Sub-directory	Output	Folder
Frequency	1	1	1	1

Figure 4.10 Tokens and their frequencies of class "build"

Token	Ctrl+3	Intro	Crash	Custom	JAXP	Parser
Frequency	1	1	1	1	1	1

Token	Welcome	Leak	Handle	View	Show	Nothing	Opened
Frequency	2	1	1	1	1	1	1

Figure 4.11 Tokens and their frequencies of class "build"

4.6.2 System Training

Step by step process of system training is given below.

a. Step 1

V ← ExtractVocabulary(C,D)

C is the set of classes/labels in training data. In the current example, C has two classes "build" and "intro". V is the vocabulary obtained by extracting tokens of all the classes.

V ← { cancel build project progress view create path error case invalid external JAR format increment build involving resource filter fail produce expected subdirectory output folder intro crash custom JAXP parser welcome leak handle shows nothing opened ctrl+3 }

b. Step II

N 🡐 CountDocs(D)

D is the set of all the documents in training data. In this example, there are three bug reports belonging to "Build" and three belonging to "intro". So value of N = 6.

c. Step III

All the sub steps in step III are repeated for all the classes included in training.

<u>Class "build"</u>

i. **N_c 🡐 CountDocsInClass(D,c)**

   N_c is the number of documents in class c. c is "build" and N_c = 3

ii. **Prior[c] 🡐 N_c / N**

   Prior[c] is the prior probability of class "build". N_c = 6 and N = 3. So Prior[c] = 0.5

iii. **$Text_c$ 🡐 ConcatenateTextOfAllDocsInClass(D,c)**
   a. $Text_c$ is the subset of vocabulary V and is obtained by concatenating all the tokens of given class.
   b. $Text_c$ 🡐 {intro crash custom JAXP parser welcome leak handle shows nothing opened ctrl+3}

iv. **T_{ct} 🡐 CountTokensOfTerm($text_c$, t)**
   a. For all the tokens in vocabulary V, find the frequency of each token in vocabulary $Text_c$. Figure 4.7 shows the frequency of each token of vocabulary V in vocabulary $Text_c$.

Token	Cancel	build	Project	View	Create	Path
Frequency	1	3	1	1	1	1

Token	Error	Case	Invalid	External	Jar	Format
Frequency	1	1	1	1	1	1

Token	increment	Involve	Resource	Filter	Fail	Produce
Frequency	1	1	1	1	1	1

Token	Ctrl+3	Intro	Crash	Custom	JAXP	Parser
Frequency	0	0	0	0	0	0

Token	Welcome	Leak	Handle	View	Show	Nothing	Opened
Frequency	0	0	0	1	0	0	0

Token	Expected	Sub-directory	Output	Folder
Frequency	1	1	1	1

Figure 4.12 frequency of each token of vocabulary V in vocabulary Text$_c$.

v. **CondProb[t][c] ← Tct + 1 / $\sum_{t'}$ ($T_{ct'}$ + 1)**

For each token from vocabulary V find the conditional probability of this token to be in class c (build). Table 4.2 shows conditional probability of all the tokens from vocabulary V.

Token	**Conditional**	**Token**	**Conditional**

	Probability		**Probability**
cancel	0.0334	ctrl+3	0.0167
build	0.0667	Intro	0.0167
project	0.0334	Crash	0.0167
view	0.0334	Custom	0.0167
create	0.0334	JAXP	0.0167
path	0.0667	Parser	0.0167
error	0.0334	Welcome	0.0167
case	0.0334	Leak	0.0167
invalid	0.0334	Handle	0.0167
external	0.0334	View	0.0334
jar	0.0334	Show	0.0167
format	0.0334	Nothing	0.0167
increment	0.0334	Opened	0.0167
involve	0.0334	Expected	0.0167
resource	0.0334	Subdirectory	0.0334
filter	0.0334	Output	0.0334
fail	0.0334	Folder	0.0334
produce	0.0334		

Table 4.2 Conditional probability of all the tokens belonging to class "build" from vocabulary V

Steps from i to v are repeated for second class "intro" and the conditional probabilities obtained are given in table 4.3

Token	**Conditional Probability**	**Token**	**Conditional Probability**

cancel	0.0204	ctrl+3	0.0408
build	0.0204	Intro	0.0408
project	0.0204	Crash	0.0408
view	0.0408	Custom	0.0408
create	0.0204	JAXP	0.0408
path	0.0204	Parser	0.0408
error	0.0204	Welcome	0.0612
case	0.0204	Leak	0.0408
invalid	0.0204	Handle	0.0408
external	0.0204	View	0.0408
jar	0.0204	Show	0.0408
format	0.0204	Nothing	0.0408
increment	0.0204	Opened	0.0408
involve	0.0204	Expected	0.0408
resource	0.0204	Subdirectory	0.0204
filter	0.0204	Output	0.0204
fail	0.0204	Folder	0.0204
produce	0.0204		

Table 4.3 Conditional probability of all the tokens belonging to class "intro" from vocabulary V

4.6.3 System Testing/Forecasting

Let there be two bug reports which are submitted to system to predict their labels. Summaries of these bug reports are

a. Incremental build of jdt.ui each time I start up
b. Hover bug welcome page

Let's start with the first bug report "Incremental build of jdt.ui each time I start up". Algorithm will check the probability of this bug report to be in one of the classes - "build, intro"- on the basis of product of prior probabilities of classes and posterior probabilities of all the terms in reported bug.

Let's check the probability of given bug to be in class "build".

Posterior probability of build = 0.5

Conditional probability of increment to be in class build = 0.0334

Conditional probability of build to be in class build = 0.0667

Conditional probability of jdt.ui to be in class build = 0.0167

Conditional probability of time to be in class build = 0.0167

Conditional probability of startup to be in class build = 0.0167

Score[build] = (0.5)(0.0334)(0.0667)(0.0167) (0.0167) (0.0167) = 0.000000005187

Now let's find Probability of the given bug to be in class "intro"

Posterior probability of intro = 0.5

Conditional probability of increment to be in class intro = 0.0204

Conditional probability of build to be in class intro = 0.0204

Conditional probability of jdt.ui to be in class intro = 0.0204

Conditional probability of time to be in class intro = 0.0204

Conditional probability of startup to be in class intro = 0.0204

Score[intro] = (0.5)(0.0204**)(** 0.0204**)(** 0.0204**)** **(**0.0204**)** **(**0.0204**) = 0.000000001766**

Result: Build has higher score so the bug will be assigned to class "build"

The second bug which is reported is "hover bug in welcome page"

Let’s find the probability of this bug to be in class “build”

Posterior probability of build = 0.5

Conditional probability of hover to be in class build = 0.0167

Conditional probability of bug to be in class build = 0.0167

Conditional probability of welcome to be in class build = 0.0167

Conditional probability of page to be in class build = 0.0167

Score [build] = (0.5) (0.0667)(0.0167) (0.0167) (0.0167) = 0.00000003889

Now let’s find the probability of the given bug to be in class “intro”

Posterior probability of intro = 0.5

Conditional probability of hover to be in class build = 0.0204

Conditional probability of bug to be in class build = 0.0204

Conditional probability of welcome to be in class build = 0.0612

Conditional probability of page to be in class build = 0.0204

Score [intro] = (0.5)(0.0612**) (** 0.0204**) (**0.0204**) (**0.0204**) = 0.0000002597**

Result: Intro has higher score so the bug will be assigned to class “intro”

Chapter 5: Testing and Experimental Results

5.1 Testing

Testing is the process of executing software to verify that it satisfies the specified requirement. It is the strategy that recovers as many defects as possible. Since no program or system design is perfect at the final implementation, therefore, testing is an essential requirement. The proposed system is tested by giving datasets as input to the system. The test cases for this purpose are discussed below.

5.2 Test Cases

Test cases are the self generated input patterns given to the system to verify the system's output against expected input. A few test cases used for the proposed software are as follows.

5.2.1 Test Case1

Package: preprocess

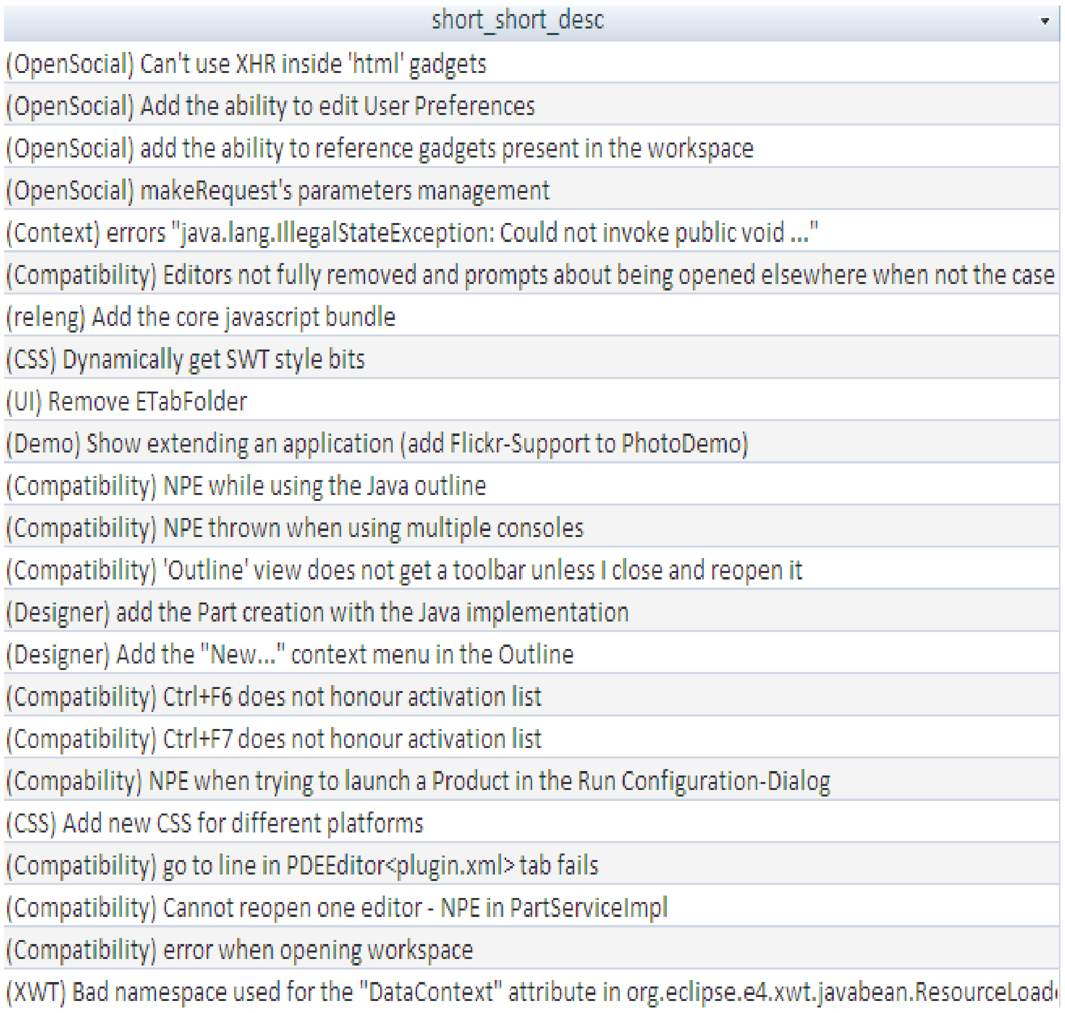
Class Name: FilterSummary

Input: short_short_desc (column containing summary)

Output: filtered summary

Result: Test succeeded

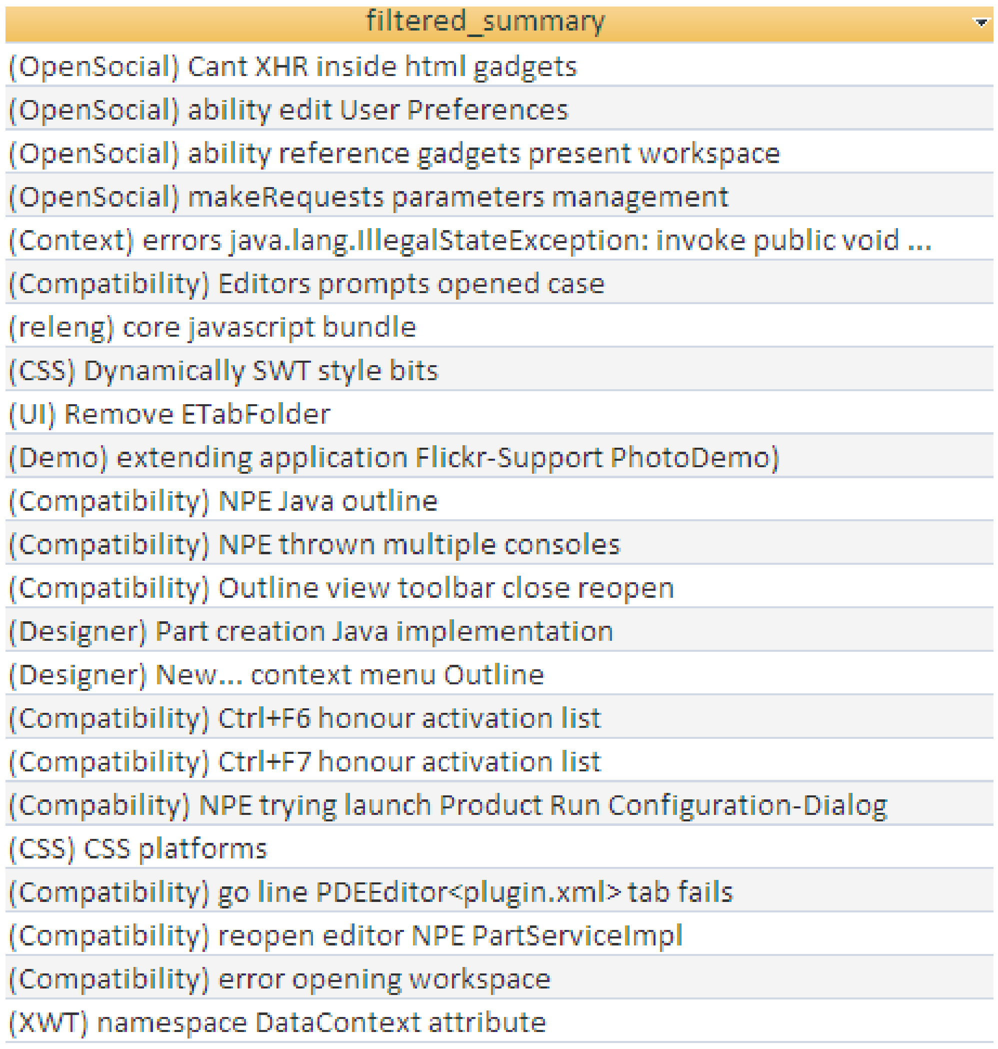
Original data



short_short_desc
(OpenSocial) Can't use XHR inside 'html' gadgets
(OpenSocial) Add the ability to edit User Preferences
(OpenSocial) add the ability to reference gadgets present in the workspace
(OpenSocial) makeRequest's parameters management
(Context) errors "java.lang.IllegalStateException: Could not invoke public void ..."
(Compatibility) Editors not fully removed and prompts about being opened elsewhere when not the case
(releng) Add the core javascript bundle
(CSS) Dynamically get SWT style bits
(UI) Remove ETabFolder
(Demo) Show extending an application (add Flickr-Support to PhotoDemo)
(Compatibility) NPE while using the Java outline
(Compatibility) NPE thrown when using multiple consoles
(Compatibility) 'Outline' view does not get a toolbar unless I close and reopen it
(Designer) add the Part creation with the Java implementation
(Designer) Add the "New..." context menu in the Outline
(Compatibility) Ctrl+F6 does not honour activation list
(Compatibility) Ctrl+F7 does not honour activation list
(Compability) NPE when trying to launch a Product in the Run Configuration-Dialog
(CSS) Add new CSS for different platforms
(Compatibility) go to line in PDEEditor<plugin.xml> tab fails
(Compatibility) Cannot reopen one editor - NPE in PartServiceImpl
(Compatibility) error when opening workspace
(XWT) Bad namespace used for the "DataContext" attribute in org.eclipse.e4.xwt.javabean.ResourceLoad

Figure 5.1 Snapshot of Test Case 1

This test case is designed to check if the summary is filtered for stop words and punctuations. It takes short_short_desc, column containing summary, as input (Figure 5.1) and produces filtered summary as shown in Figure 5.2. Summary is actually the small description of bugs having keywords enclosed in brackets e.g. *(UI)*. As these bugs are reported in free format English language so it contains lot of punctuations and stop words and hence is not fit for direct use in bug prediction and needs some filtering. The output of the filtering step is shown in Figure 4.2 where each bug is filtered for the stop words and punctuations and the resultant summary is stored as filtered summary.



filtered_summary
(OpenSocial) Cant XHR inside html gadgets
(OpenSocial) ability edit User Preferences
(OpenSocial) ability reference gadgets present workspace
(OpenSocial) makeRequests parameters management
(Context) errors java.lang.IllegalStateException: invoke public void ...
(Compatibility) Editors prompts opened case
(releng) core javascript bundle
(CSS) Dynamically SWT style bits
(UI) Remove ETabFolder
(Demo) extending application Flickr-Support PhotoDemo)
(Compatibility) NPE Java outline
(Compatibility) NPE thrown multiple consoles
(Compatibility) Outline view toolbar close reopen
(Designer) Part creation Java implementation
(Designer) New... context menu Outline
(Compatibility) Ctrl+F6 honour activation list
(Compatibility) Ctrl+F7 honour activation list
(Compability) NPE trying launch Product Run Configuration-Dialog
(CSS) CSS platforms
(Compatibility) go line PDEEditor<plugin.xml> tab fails
(Compatibility) reopen editor NPE PartServiceImpl
(Compatibility) error opening workspace
(XWT) namespace DataContext attribute

Figure 5.2 Snapshot of Test Case 1

5.2.2 Test Case 2

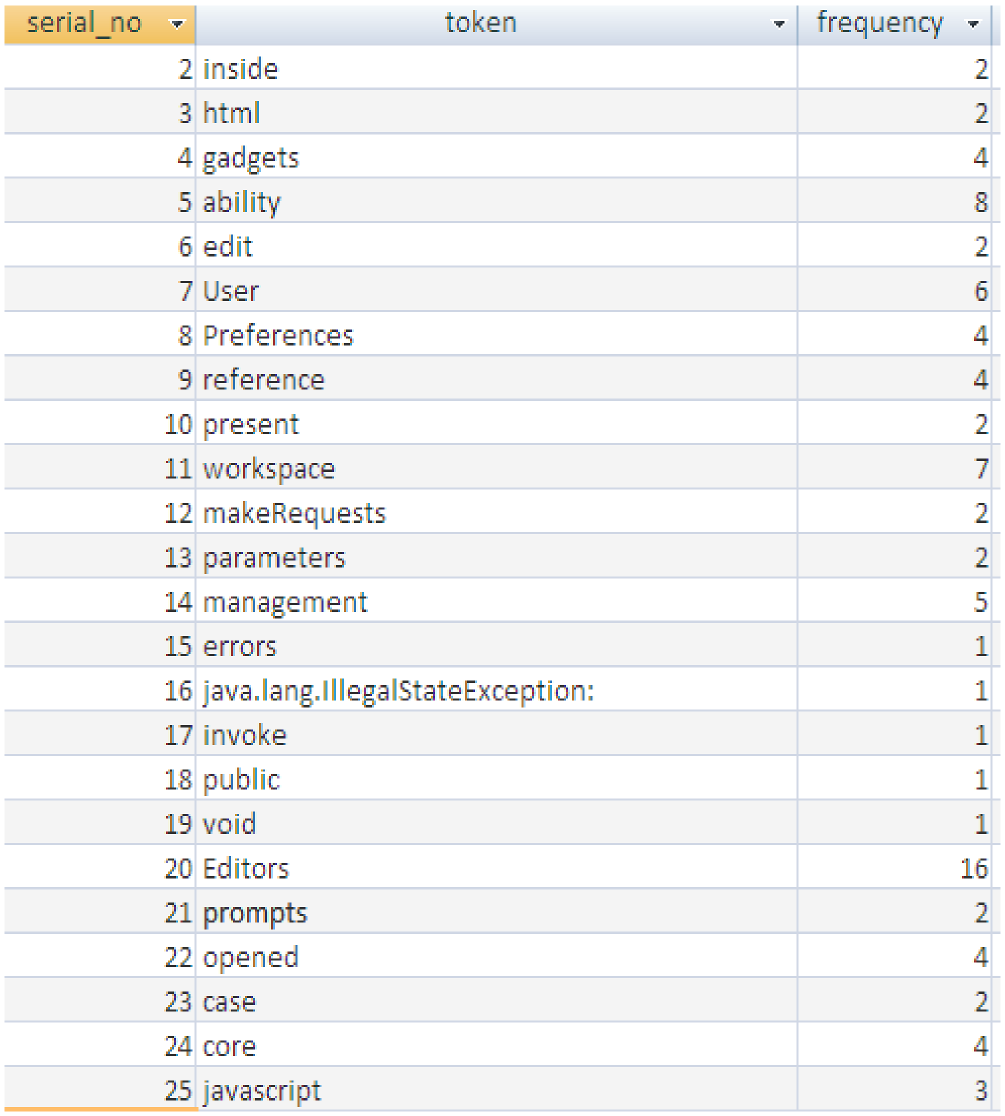
Class Name: Tokenizer

Input: filtered Summary

Output: Tokens, frequency (table containing tokens of filtered summary and frequency)

Result: Test succeeded

Output



serial_no	token	frequency
2	inside	2
3	html	2
4	gadgets	4
5	ability	8
6	edit	2
7	User	6
8	Preferences	4
9	reference	4
10	present	2
11	workspace	7
12	makeRequests	2
13	parameters	2
14	management	5
15	errors	1
16	java.lang.IllegalStateException:	1
17	invoke	1
18	public	1
19	void	1
20	Editors	16
21	prompts	2
22	opened	4
23	case	2
24	core	4
25	javascript	3

Figure 5.3 Snapshot of Test Case 2

To check that tokens with their frequency are created, test case 2 is designed that takes filtered summary- output of test case 1- as input and produces tokens of the summary and calculates frequency against each token. Results are shown in Figure 5.3. To precede bug prediction, the first step is to read each record of filtered summary word by word. To accomplish this task, tokens for the whole filtered summary are produced and stored in the column *tokens* as shown in Figure 5.3. Another requirement for bug prediction is to know the number of occurrences of each token in the bug records, so that the tokens with highest occurrence could decide the class of a particular bug. Therefore, test case 2 also checks that frequency of each token is obtained successfully. Tokens and frequency are stored in the same table as shown in Figure 5.3.

5.2.3 Test Case 3

Package: featureextraction

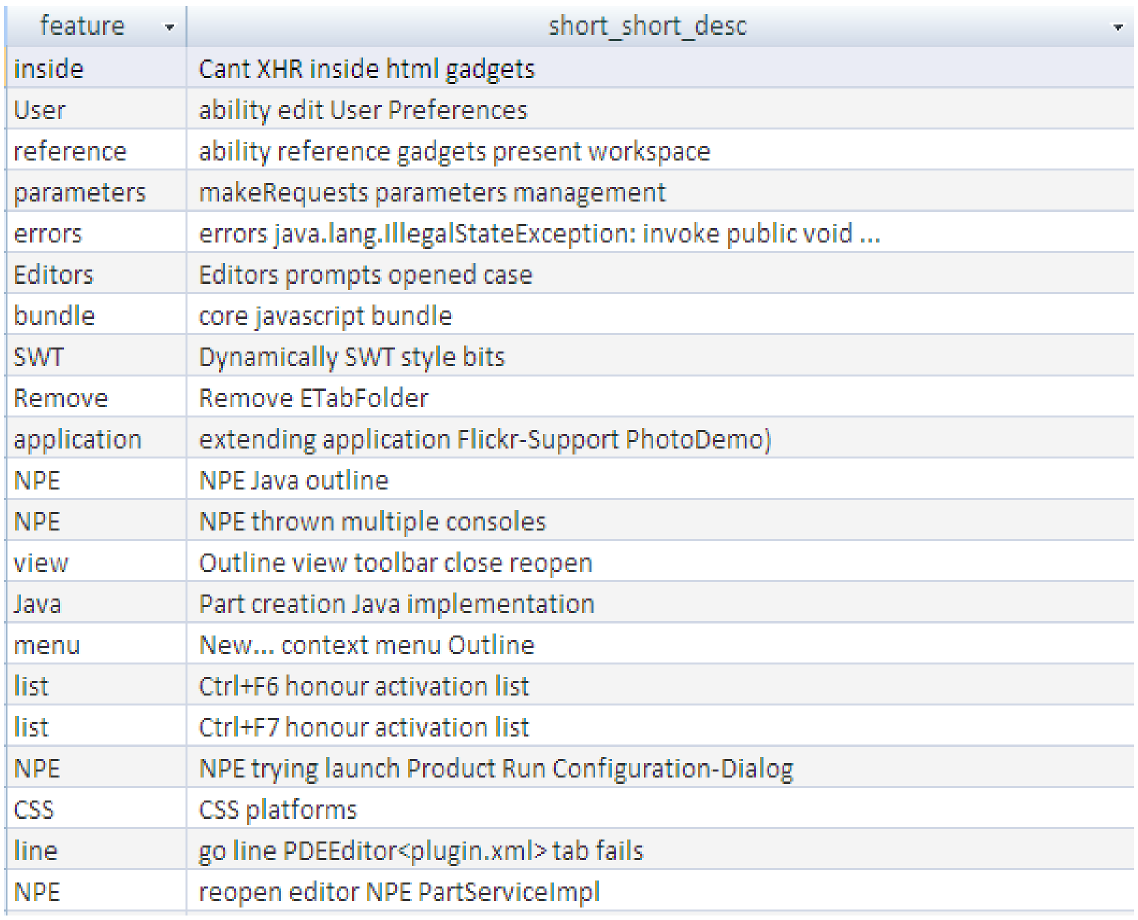
Class Name: FeatureExtraction

Input: *token, frequency* from each class table

Output: features of each bug

Result: Test succeeded

Output File



feature	short_short_desc
inside	Cant XHR inside html gadgets
User	ability edit User Preferences
reference	ability reference gadgets present workspace
parameters	makeRequests parameters management
errors	errors java.lang.IllegalStateException: invoke public void ...
Editors	Editors prompts opened case
bundle	core javascript bundle
SWT	Dynamically SWT style bits
Remove	Remove ETabFolder
application	extending application Flickr-Support PhotoDemo)
NPE	NPE Java outline
NPE	NPE thrown multiple consoles
view	Outline view toolbar close reopen
Java	Part creation Java implementation
menu	New... context menu Outline
list	Ctrl+F6 honour activation list
list	Ctrl+F7 honour activation list
NPE	NPE trying launch Product Run Configuration-Dialog
CSS	CSS platforms
line	go line PDEEditor<plugin.xml> tab fails
NPE	reopen editor NPE PartServiceImpl

Figure 5.4: Snapshot of Test Case 3

This test verifies that a bug feature is extracted from each record containing bug. This feature is extracted using Chi Square feature extraction algorithm. It takes as input, the *token* and *frequency* column of each class and produces a feature as output, shown in Figure 5.4.

5.2.4 Test Case 4

Package: modeltraining

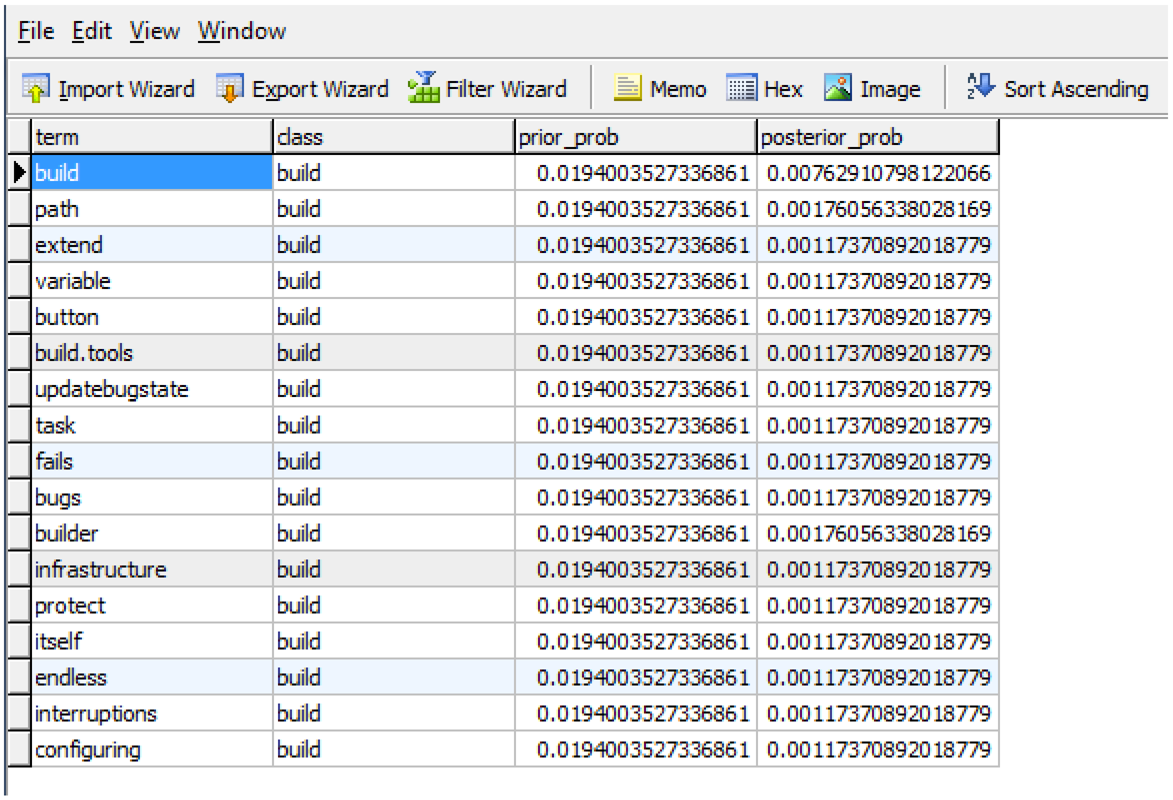
Class Name: ModelTraining

Input: filtered summary, features obtained in test case 3

Output: model trained on examples

Result: Test succeeded

Classification model (Naïve Bayes) is trained on training data that is obtained after preprocessing and feature selection. Classification model is actually the set of probabilities that each feature contribute towards a class. Figure 4.5 shows the prior probability and posterior probability of each feature against a class.


File Edit View Window
Import Wizard Export Wizard Filter Wizard Memo Hex Image Sort Ascending

term	class	prior_prob	posterior_prob
build	build	0.0194003527336861	0.00762910798122066
path	build	0.0194003527336861	0.00176056338028169
extend	build	0.0194003527336861	0.00117370892018779
variable	build	0.0194003527336861	0.00117370892018779
button	build	0.0194003527336861	0.00117370892018779
build.tools	build	0.0194003527336861	0.00117370892018779
updatebugstate	build	0.0194003527336861	0.00117370892018779
task	build	0.0194003527336861	0.00117370892018779
fails	build	0.0194003527336861	0.00117370892018779
bugs	build	0.0194003527336861	0.00117370892018779
builder	build	0.0194003527336861	0.00176056338028169
infrastructure	build	0.0194003527336861	0.00117370892018779
protect	build	0.0194003527336861	0.00117370892018779
itself	build	0.0194003527336861	0.00117370892018779
endless	build	0.0194003527336861	0.00117370892018779
interruptions	build	0.0194003527336861	0.00117370892018779
configuring	build	0.0194003527336861	0.00117370892018779

Figure 5.5 Model building by finding probabilities of features against a class

5.2.5 Test Case 5

Package: modeltesting

Class Name: ModelTesting

Input: records with unknown classes/labels

Output: records with known classes/labels

Result: A prediction accuracy of 89% is obtained with testing to training ratio of 1:10

5.3 Experimental Results

This section is based on the output and results generated after applying the implemented system on bug data set obtained from Bugzilla bug repository. Bugzilla maintains the bug reports of number of projects like Mozilla, Eclipse, SQL etc. For this research bugs reports from Eclipse bug repository are used.

Results are obtained on the basis of prediction accuracy. Prediction Accuracy is defined as

"**Ratio of the number of bug reports with correct types classified to the total number of bug reports to be classified.**"[14]

Prediction accuracy was not good during the first few experiments. In order to improve accuracy, some processes in preprocessing steps were improved.

a. It was found that still there are some punctuations and stop words in bug summaries which were decreasing the prediction accuracy. So, these stop words were appended in stop words dictionary. Stop words dictionary which is used contained almost 450 words.
b. In this research, data of an open bug repository is used in which end users of a product report the bugs. Users most of the time are not very technical and use the terminology that confuses the prediction model instead of helping in its training. Such reports were manually removed from the dataset. Although it's a time consuming task but it increases the accuracy.

c. Naïve Bayes is affected by class prior probabilities. If one takes data set having one class containing a very large number of records and other having very small, model automatically will twist the prediction towards class having large data. So, classes having much difference in record set cannot be used.

Experiments are conducted by changing the training to testing ratio, classes/labels and feature extraction algorithms.

5.3.1 Classes having varying number of records:

In an experiment, algorithm was trained on classes having very large difference in number of records. Some classes like misc, navigat etc. have a very large number of records (almost 500 records) while others have a very few e.g. build, marker (less than 100 records). Forecasting results were deviated towards classes having more records. Table 5.1 shows the classes included in the training phase and the number of records in each class.

Class/Label	**Number of records**
Build	82
Externaltools	90
Markers	97
Javadoc	151
Mode	194
Launch	198
Intro	257
Navigat	357
Pref	398
Misc	498

Table 5.1 Classes in training data with number of bugs

5.3.1.1 Experimental Results

Table 5.2 shows the experimental results obtained by training the algorithm on classes mentioned in table 5.1. Prediction accuracy was very low due to the large difference in number of records in classes on which proposed system was trained. Further more data was not much refined in initial set of experiments that resulted in low prediction accuracy.

Training to testing ratio that was used in this experiment was 4:1. In first set of experiments system was trained on data without using any feature selection algorithm. System was trained on 3725 bug reports and testing was performed on 914 bug reports. 46.93% Prediction accuracy was obtained. Accuracy was poor because algorithm was trained on classes having varied number of records and system forecasted most of the bug reports belonging to the classes having more data in training.

Same experiment was repeated with TFIDF as feature selection algorithm. Term Frequency Inverse Document Frequency is usually used as feature selection in categorization of documents like emails, news, documents on World Wide Web etc. For bug classification, this algorithm was not very fruitful because data from Bugzilla is used which is an open source bug tracking system and unlike formal documents mentioned before, the terminology used by users to report bugs is not as technical and formal to apply TFIDF. Accuracy was reduced to 39% using TFIDF.

Another experiment was conducted with same data but now using Chi Square algorithm for feature selection. Training and testing data was same as used in experiment without any feature selection algorithm and the prediction accuracy obtained was 51%.

5.3.1.2 Summary

Best results were obtained using Chi Square Feature selection algorithm. Results were further improved in preceding experiments by increasing training to testing ratio and by refining the training data in preprocessing step.

Feature Selection Algorithm	**Training Data**	**Testing Data**	**Training/Tesitng ratio**	**Prediction Accuracy**
No	3725	914	4:01	46.93
TFIDF	3725	914	4:01	39
Chi Square	3725	914	4:01	51

Table 5.2 experimental results with classes having varying number of data

5.3.2 Classes having small training set

In another set of experiments, system was trained over very small data. All the classes used in training have data range from 41 to 56. List of data and number of bugs in each class is given in table 5.3.

Class Labels	**Number of bugs**
SSH	50
ChangeSet	50
runtime	50
Linked	48
reconciling	51
PDE	45
Workbench	41

Table 5.3 Training classes with number of bugs

5.3.2.1 Experimental Results

As already discovered in the first set of experiments that TFIDF was of no use in increasing the prediction accuracy. Second set of experiments was performed using Chi Square feature selection algorithm. Although prediction accuracy increased as compared to first experiments due to small difference in a number of bugs in each class but still accuracy was low as data was not well pre-processed. Table 5.4 shows the accuracy obtained as a result of experiments performed by training algorithm using classes having small difference in number of bugs.

Training/Testing Ratio	**Total Data**	**Training Data**	**Testing Data**	**Accuracy (%)**
1:01	820	421	399	57
1:05	820	665	155	58
1:07	820	715	105	63
1:10	820	748	72	66

Table 5.4 Experimental results with classes having small difference in record set

Using same set of data, experiment is repeated by using porter stemmer algorithm during pre processing. Porter stemmer is used to minimize a word to its root. The algorithm was not of any worth in increasing the accuracy because the bugs are reported by non technical users and do not contain proper terms which can be pre processed by the techniques used in pre processing of documents like emails, world wide web documents etc. Porter stemmer decreased the prediction accuracy from 57% to 45% using 1:5 testing to training ratio.

5.3.2.2 Summary

Prediction accuracy increases by decreasing the difference in number of records in classes used in system training. Porter stemmer decreased the prediction accuracy.

5.3.3 Classes having refined data obtained by improved pre processing

Table 5.5 shows the effect of changing the training to testing ratio on prediction accuracy. Table data is plotted in figure 5.1. Graph clearly shows that prediction accuracy increases as training to testing ratio increases. Highest accuracy is obtained when this ratio is 1:11. However, we cannot increase this ratio to a very large extend as it might over fit the training data and training over a very huge vocabulary is very time consuming and thus decreases the efficiency of algorithm. In figure 5.6, accuracy starts decreasing if the testing to training ratio increases from 1:11.

testing/training ratio	**Accuracy**
1:01	61
1:02	66
1:03	68
1:04	69
1:05	70
1:06	76
1:07	80
1:08	82
1:09	83
1:10	84
1:11	89
1:12	88
1:13	86
1:14	86
1:15	86

Table 5.5.Accuracy with change in training to testing ratio

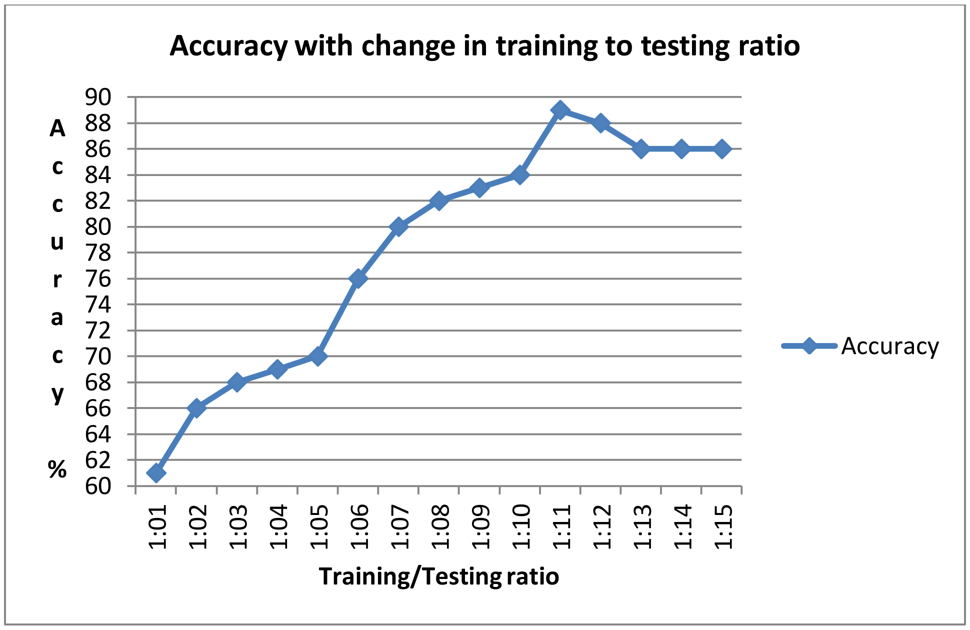


Figure 5.6 Accuracy with change in Training to testing ratio

The vocabulary obtained after applying "bag of words" approach was of very high dimensionality. Model training on such a large vocabulary is very time consuming. Chi- square algorithm is used for feature selection to reduce the number of features in vocabulary. Chi-square algorithm although reduces the dimension of total vocabulary and increases the efficiency of model training, great care must be taken in selecting the ratio of features to be used in training out of total features. Increasing features and thus increasing training to testing ratio although increases the accuracy but execution time of algorithm increases as well. So, increasing the training vocabulary data beyond a certain limit is not feasible in real time applications.

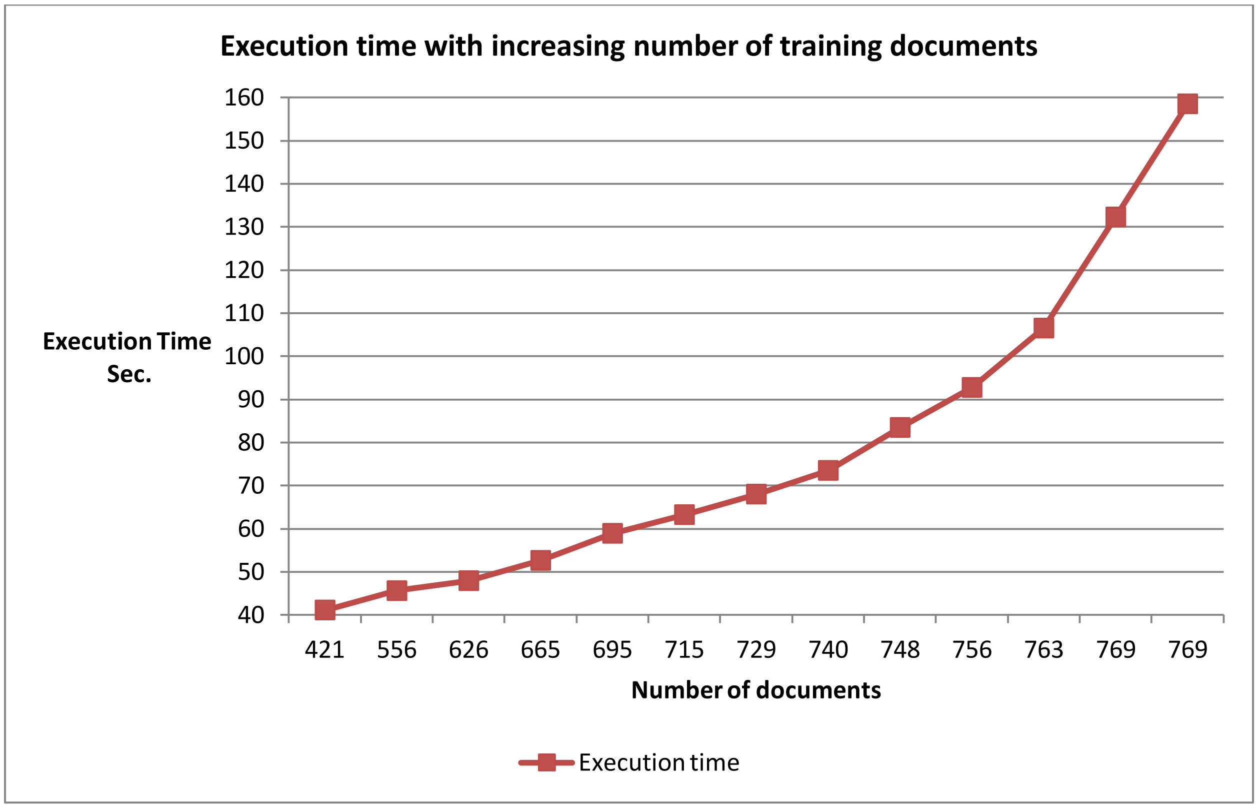


Figure 5.7 Training to testing ratio versus execution time

After observing the results of a number of experiments, it was found that the best results were obtained when the features having Chi value greater than 0.5 are taken and used in training. Therefore, 0.5 chi value was used as a threshold in model training. There were almost 1200 features with chi threshold 0.5

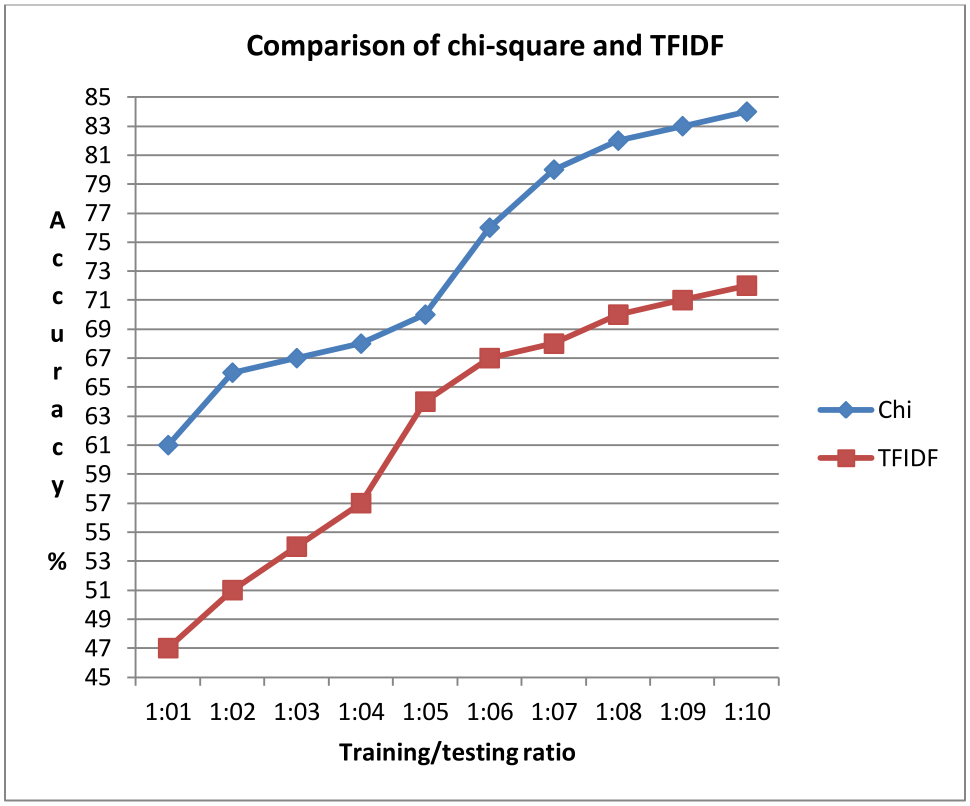


Figure 5.8: Comparison of Chi-Square and TFIDF

There are a number of feature selection algorithms available. Two of these techniques are used in this research- Chi Square test and TFIDF. Figure 5.8 shows that Chi Square algorithm proved to be more effective in this case and the features extracted through this algorithm gave better prediction accuracy.

5.4 Comparison with Other Systems

5.4.1 Classification system proposed by Lai Xu

Proposed system using Naïve Bayes classifier is a probability based approach that works on the prior probability of classes and conditional probability of features in the classes. Another important model for text classification is support vector machine. Changzhu Kong,, Lian Yu, Lei Xu, HuiHui Zhang and Jingtao Zhao used SVM for bug classification. Proposed technique using naïve Bayes text classifier has following advantages over this system:

a) When training data is small, proposed system performs better than SVM based system of Lei Xu. Training curve for SVM is much greater than Naïve Bayes and when enough training set is not given it does not perform well.

b) As far as processing time is concerned Lei system is in a disadvantage. Processing time is much higher from the other proposed technique and it grows quadratically as the number of documents increases in training set.

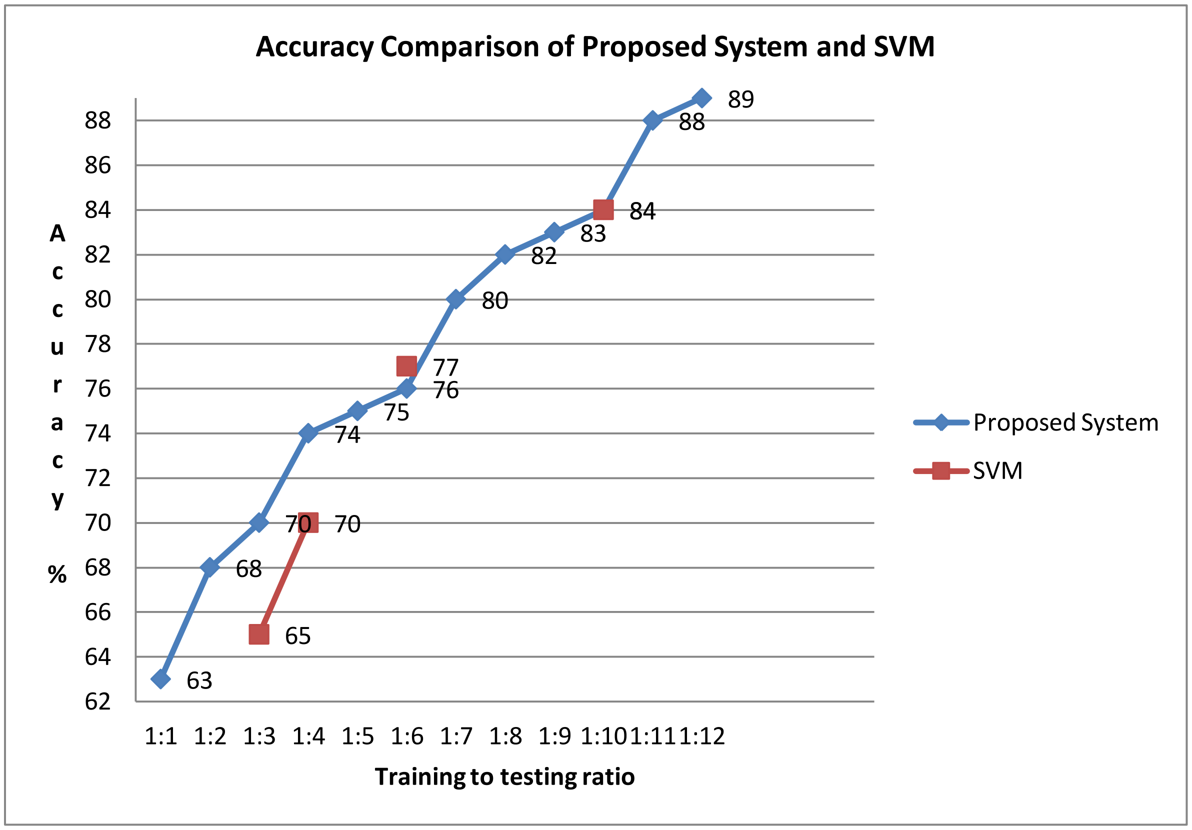


Figure 5.9 Accuracy Comparison of proposed system and one proposed by Lian Yu

a) Proposed system starts with an advantage when a small number of documents are used in the training set, but then as the number of documents increases, the difference diminishes. As far as processing time is concerned Lian Yu system is in a disadvantage. Processing time is much higher from the other text classification techniques and it grows quadratically as the number of documents increases in training set.

Fabrice Colas and Pavel Brazdil gave the comparative analysis of Naïve Bayes and SVM in their research on the basis of processing time. Figure 5.5 shows that Naïve Bayes out performs SVM. Processing time of both algorithms is almost the same when the number of documents/number of features is small. However, if the number of documents in training data increases, the processing time of SVM increases quadratically.

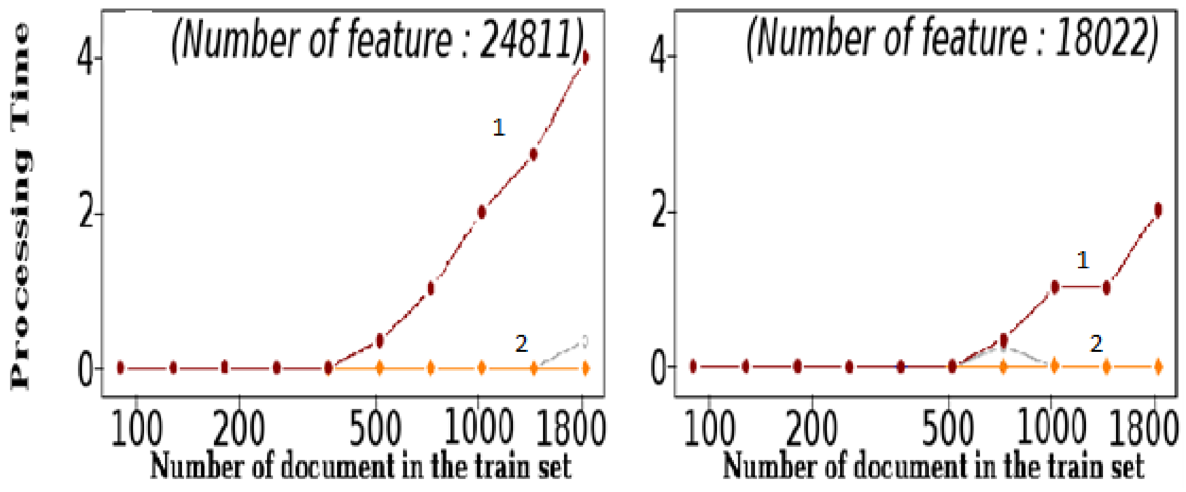


Figure 5.10 Comparison of Naïve Bayes (2) and SVM (1) on the basis of processing time

5.4.2 Comparison with John Anvik, Lyndon Hiew and Gail C. Murphy Technique

John Anvik, Lyndon Hiew and Gail C. Murphy presented a bug classification technique for bug assignment to developers using semi automated technique. The approached used a supervised machine learning algorithm. 64% of maximum precision was obtained for firefox and eclipse data.

Chapter 6: Conclusion and Future work

6.1 Conclusion

In open source bug repositories, bugs are reported by users. Triaging of these bugs is a tedious and time consuming task. If some proper class is assigned to these bugs it would be easier to assign these bugs to relevant developers to fix them. However, as reporters of these bugs are mostly non-technical it would not be possible for them to assign correct class to these bugs. In this research an automated system for classifying these bugs is devised, using multinomial Naïve Bayes text classifier. Chi Square and TFIDF are used for feature selection.

Data mining is a process of extracting meaningful information from raw data. Data that is not numerical or categorical is considered unstructured and is not suitable for the purpose of data mining. Extracting meaningful information from this unstructured data is known as text mining. Text mining is not a separate field from data mining but an extension/specialization of it. Text mining is used in applications like analyzing open-ended survey responses, automatic processing of messages and emails, analyzing warranty or insurance claims and diagnostic interviews, investing competitors by crawling their web sites and mining software repositories etc.

Software systems have a history of how they came to be and this history is maintained in software repositories. Examples of software repositories are archive communication, bug repositories, code repositories and deployment logs etc. Although these repositories are a huge treasure of information about software system and software project but to extract useful knowledge from these repositories is a mess. Idea behind mining software repositories is to devise tools to access the wealth of information in these software repositories to extract useful knowledge by analyzing them.

In this research a bug classification system is proposed that takes bugs from open bug repositories and classifies them in different labels/classes. Any bug report is first preprocessed. During preprocessing, stop words are removed and stemming is applied to the data. Preprocessing would result in a huge vocabulary of words which is almost infeasible to use as it is. To decrease the dimensionality of vocabulary, feature selection technique is applied that reduces the vocabulary size and increases the efficiency of algorithm training. Chi square and TFIDF are used for feature selection. Chi square gave the best results out of the algorithms. Naïve Bayes classifier is used for classification and maximum of 89% accuracy is obtained on training to testing ratio of 1:10.

Lei Xu,, Lian Yu, Jingtao Zhao, Changzhu Kong, and HuiHui Zhang proposed a classification model based on SVM. Our technique using Naïve Bayes text classifier has following advantages over SVM.

a) When training data is small, Naïve Bayes performs better than SVM. Training curve for SVM is much greater than Naïve Bayes and when enough training set is not given it does not perform well.
b) As far as processing time is concerned SVM is in a disadvantage. Processing time is much higher than the other text classification techniques and it grows quadratically as the number of documents increases in training set.

6.2 Future Work

The system can be further improved by applying feature selection techniques other than Chi-Square and TFIDF.

Synonym dictionary can be used so that system can tackle the issue of understanding the synonyms of the same words. For instance, if a user reports a bug related to firefox, system should consider it firefox whether user uses the word browser or Mozilla firefox for it.

Future work can be devoted to the development of an automated triaging system using our approach that assigns bugs to relevant developers to be fixed. Furthermore, bug repositories can be used in combination with other repositories like code repositories to find more error prone areas of a project.

Chapter 7: References

1. Ahmed E. Hassan. The Road Ahead for Mining Software Repositories. IEEE Computer society, 2008.
2. Andreas Hotho . A Brief Survey of Text Mining. University of Kassel. School of Computer Science Otto-von-Guericke-University Magdeburg, May 13, 2005
3. Stephan Diehl, Harald C. Gall and Ahmed E. Hassan. Special issue on mining software repositories . Empirical Software Engineering An International Journal © Springer Science+Business Media, LLC 2009
4. Hassan, A. E. 2008. The road ahead for mining software repositories. In Frontiers of Software Maintenance. 48–57.
5. Lehman, M. and Fernandez-Ramil, J. C. 2002. Software evolution and feedback: Theory and practice. In Software Evolution. John Wiley and Sons.
6. Basili, V. R. and Perricone, B. 1984. Software errors and complexity: An empirical investigation. In Communications of the ACM, 27(1). 42–52.
7. Mockus, A., Weiss, D. M., and Zhang, P. 2003. Understanding and predicting effort in software projects. In In Proceedings of the 25th International Conference on Software Engineering. 274– 284.
8. Cubranic, D., Murphy, G. C., Singer, J., and Booth, K. S. 2005. Hipikat: A project memory for software development. In IEEE Transactions on Software Engineering, 31(6). 446–465.
9. Li, Z., Lu, S., Myagmar, S., and Zhou, Y. 2006. Cp-miner: Finding copy-paste and related bugs in large-scale software code. In IEEE Transactions on Software Engineering, 32(3). 176–192.
10. Mandelin, D., Xu, L., Bodk, R., and Kimelman, D. 2005. Jungloid mining: Helping to navigate the api jungle. In V. Sarkar and M. W. Hall, editors, PLDI. 48–61.
11. Kagdi, H., Collard, M. L., and Maletic, J. I. 2007b. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J. Softw. Maint. Evol. 19, 77–131.

12. Olga Baysal Michael W. Godfrey Robin Cohen. A Bug You Like: A Framework for Automated Assignment of Bugs. IEEE 17th international conference, 2009.
13. Chuanlei Zhang, Hemant Joshi, Srini Ramaswamy and Coskun Bayrak. A Dynamic Approach to Software Bug Estimation, SpringerLink, 2008.
14. Lian Yu, Changzhu Kong, Lei Xu, Jingtao Zhao, and HuiHui Zhang. Mining Bug Classifier and Debug Strategy Association Rules for Web-Based Applications. SpringerLink, 2008.
15. Nicholas Jalbert, Westley Weimer. Automated Duplicate Detection for Bug Tracking Systems. IEEE computer society ,2008.
16. Tilmann Bruckhaus, Charles X. Ling, Nazim H. Madhavji, Shengli Sheng. Software Escalation Prediction with Data Mining.
17. Davor Cubranic, Gail C. Murphy. Automatic bug triage using text categorization . In Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (June 2004), pp. 92-97.
18. 18. Adrian Schröter , Thomas Zimmermann , Rahul Premraj , Andreas Zeller Saarland University. If Your Bug Database Could Talk. In Proceedings of the 5th International Symposium on Empirical Software Engineering. 2006.
19. Sunghun Kim. Adaptive bug prediction by analyzing project history. ACM, 2006.
20. Peter Weissgerber, Mathias Pohl , Michael Burch. Visual Data Mining in Software Archives To Detect How DevelopersWork Together, Proceedings of the Fourth International Workshop on Mining Software Repositories, 2007.
21. Andy Zaidman, Bart Van Rompaey, Serge Demeyer, and Arie van Deursen. Mining Software Repositories to Study Co-Evolution of Production & Test Code. In Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation, 2008.
22. T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in software evolution. In Proc. of the Int'l Workshop on Principles of Software Evolution (IWPSE), pages 13–22. IEEE, 2005.
23. P. Runeson. A survey of unit testing practices. IEEE Software, 25(4):22–29, July/August 2006.

24. L. Moonen, A. van Deursen, A. Zaidman, and M. Bruntink. Software Evolution, chapter The interplay between software testing and software evolution. Springer, 2008. Editors: T. Mens, and S. Demeyer.

25. S. Elbaum, D. Gable, and G. Rothermel. The impact of software evolution on code coverage information. In Proc. Int'l Conf. on Soft. Maint. (ICSM), pages 170–179. IEEE, 2001.

26. M.-A. Storey, D. Cˇ ubranic´, and D. German. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In Proc. of the Symp. on Soft. Visualization, pages 193–202. ACM, 2005.

27. E. Maximilien and L.Williams. Assessing test-driven development at IBM. In Proc. Int'l Conf. on Software Engineering(ICSE), pages 564–569. IEEE, 2003.

28. Naïve Bayesian Based on Chi Square to Categorize Arabic Data, International Business Information Management Conference (11th IBIMA), 2009.

29. http://tartarus.org/~martin/PorterStemmer/

30. http://www.thearling.com/text/dmtechniques/dmtechniques.htm

31. www.cs.uiuc.edu/homes/hanj/cs412/bk3_slides/08ClassBasic.ppt

Appendix A

"about", "above", "across", "after", "again", "against", "all", "almost", "alone", "along", "already", "also", "although", "always", "among", "an", "and", "another", "any", "anybody", "anyone", "anything", "anywhere", "are", "area", "areas", "around", "as", "ask", "asked", "asking", "asks", "at", "away", "back", "backed", "backing", "backs", "be", "became", "because", "become", "becomes", "been", "before", "began", "behind", "being", "beings", "better", "between", "big", "both", "but", "by", "came", "can", "cannot", "case", "cases", "certain', 'certainly", "clear", "clearly", "come", "could", "did", "differ", "different", "differently", "do", "does", "done", "down", "down", "downed", "downs", "during", "each", "early", "either", "end", "ended", "ending", "ends", "enough", "even", "evenly", "ever", "every", "everybody", "everyone", "everything", "everywhere", "face", "faces", "fact", "facts", "far", "felt", "few", "find", "finds", "irst", "for", "four", "from", "for", "ull", "fully", "further", ", 'urthered", "furthering", "furthers", "gave", "general", "generally", "get", "gets", "ive", "given", "ives", "go", "going", "good", "got", "great", "greater", "greatest", "roup", "grouped", "grouping", "groups", "had", "has", "have", "having", "he", "her", "here", "herself", "high", "high", "high", "highe", "highest", "him", "himself", "his", "how", "however", "if", "important", "in", "interest", "interested", "interesting", "interests", "into", "it", "its", "just", "keep", "keeps", "know", "known", "knows", "large", "largely", "last", "later", "latest", "least", "less", "let", "lets", "like", "likely", "long", "longer", "longest", "made", "make", "making", "man", "many", "may", "me", "member", "members", "", "might", "more", "most", "mostly", "mrs", "much", "must", "my", "myself", "necessary", "need", "needed", "needing", "needs", "never", "new", "new", "newer", "newest", "next", "no", "nobody", "non", "noone", "not", "nothing", "now", "nowhere", "number","numbers", "of", "off", "often", "old", "lder", "oldest", "on", "once", "one", "only", "open", "opened", "opening", "opens", "order", "ordered", "ordering", "orders", "other", "others", "our", "out", "over", "part", "parted", "parting", "parts", "per", "perhaps", "place", "places", "point", "pointed", "pointing", "points", "possible", "resent", "presented", "presenting", "presents", "problem", "problems", "put", "puts", "quite", "rather", "really", "right", "right", "room", "rooms", "said", "same", "saw", "say", "says", "second", "seconds", "see", "seem", "seemed", "seeming", "seems", "sees", "several", "shall", "she", "should", "show", "", "howed", "showing", "shows", "side", "sides", "since", "small", "smaller", "smallest", "so", "some", "somebody", "someone", "something", "somewhere", "state", "states", "still", "still", "such", "sure", "take", "taken", "than", "that", "the", "their", "them", "then", "there", "therefore", "these", "they", "thing", "things", "think", "thinks", "this", "those", "though", "thought", "thoughts", "three", "through", "thus", "to", "today", "together", "too", "took", "toward", "turn", "turns", "two", "u", "under", "until", "up", "upon", "us", "use", "used", "uses", "very", "want", "anted", "wanting", "wants", "was", "way", "ways", "we", "well", "wells", "went", "were", "what", "when", "where", "whether", "which", "while", "who", "whole", "whose", "why", "will", "with", "within", "without", "work", "worked", "working", "works", "would", "year", "ears", "yet", "you", "young", "younger", "youngest", "your", "yours", "is", "a", "the", "or", "doesn’t"