8-2013

Categorizing and predicting reopened bug reports to improve software reliability

Rishikesh Gawade

Categorizing and predicting reopened bug reports to improve software reliability

A Thesis

Presented to the

Department of Computer Science

And the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment
Of the Requirements for the Degree

Master of Science

University of Nebraska at Omaha

By

RishikeshGawade

August 2013

Supervisory Committee:

Dr. Harvey Siy

Dr. Robin Gandhi

Dr. ParvathiChundi

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

ABSTRACT

CATEGORIZING AND PREDICTING REOPEN BUG RPORTS TO IMPROVE SOFTWARE RELIABLITY

Rishikesh Gawade, MS

University of Nebraska, 2013

Advisor: Dr Harvey Siy

Software maintenance takes two thirds of the life cycle of the project. Bug fixes are an important part of software maintenance. Bugs are tracked using online tools like Bugzilla. It has been noted that around 10% of fixes are buggy fixes. Many bugs are documented as fixed when they are not actually fixed, thus reducing the reliability of the software. The overlooked bugs are critical as they take more resources to fix when discovered, and since they are not documented, the reality is that defect are still present and reduce reliability of software. There have been very few studies in understanding these bugs. The best way to understand these bugs is to mine software repositories. To generalize findings we need a large number of bug information and a wide category of software projects. To solve the problem, a web crawler collected around a million bug reports from online repositories, and extracted important attributes of the bug reports. We selected four algorithms: Bayesian network, NaiveBayes, C4.5 decision tree, and Alternating decision tree. We achieved a decent amount of accuracy in predicting reopened bugs across a wide range of projects. Using AdaBoost, we analyzed the most important factors responsible for the bugs and categorized them in three categories of reputation of committer, complex units, and insufficient knowledge of defect.

ACKNOWLEDGEMENTS

I thank my advisor Dr. Harvey Siy for guiding me in understanding research problem, selection of resources for research, using software tools and guiding me on every step in field of data mining and empirical software engineering which was new and challenging to me. I thank Dr. Chundi for clearing my knowledge about algorithms and Dr. Gandhi for guiding me through resources of software defects. I thank my family for their encouragement and support without which I should have not able to devote myself fully to thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Problem

Software maintenance takes two thirds of the cost of life cycle of the projects that make up the 70 billion dollar software industry in US (Boehm & Basili, 2001). Fixing bugs is an important part of the maintenance process. However, around 10% of fixes are buggy fixes (Gu, Barr, Hamilton, & Su, 2010). If a system has a high percentage of overlooked fixes, then it could reduce reliability of software, and there has been very little work in the area to broaden the understanding of these bugs (Guo, Zimmermann, Nagappan, & Murphy, 2010). Reopened bugs are critical and wasteful as they consume more resources than the average bug, and the average time fixing them is twice the regular time of the average bug (Shihab E. , Ihara, Kamei, & Ibrahim, 2010). If we are able to understand the root causes and factors responsible for the reopened bugs and predict them in advance it will help in saving resources and set standards to increase the reliability of the software. To understand the causes and factors of the reopened bugs, we need information regarding reopened bugs from software repositories. One of the biggest challenges in getting the data from software repositories is that there is limited access to data, and the difficulty in extracting data is great due to its complex nature (Hassan, 2008). To generalize the root causes of the reopened bugs the data extracted should come from a variety of projects. Also, the number of bug information extracted should be large. Once the data is extracted from the reopened bugs, they should have common root causes and factors that can be used to predict the chances of reopened bugs.

1.2 Objective

The objective of our research is to develop automated data mining techniques for software repositories. Once bug data is collected and processed, the next task is to find out root causes of reopened bugs and categorize them into common patterns. We will apply prediction algorithms to predict the reopening of bugs and to test this method on different categories of projects to achieve a decent amount of accuracy.

1.3 Approach

First step was to find a wide categorical variety of open source projects, and once a category was selected, we had to find a bug tracking systems where data is accessible to the public. We developed novel, automated data extraction methods to extract data from bug repositories and websites by crawling through the systems. We divided bug data into three categories: report data, activity data, and comment data. Once data was extracted and cleaned, an overview of data was shown. The next part of the study was applying machine learning algorithms to create predictions for reopened bugs. We studied and selected two tools, Rattle (Williams, 2009) and Weka (Mark Hall, 2009), for the implementation of algorithms. We tested all machine algorithms in Weka and rattle to find out which ones work most efficiently in predicting reopened bugs from given factors. From this, four algorithms were selected: Bayesian network (Friedman, Geiger, & Goldszmidt, (1997)), NaiveBayes(Bayes, 1763), C 4.5 decision tree (Quinlan, 1986), and Alternating decision tree (Freund & Mason, The alternating decision tree learning algorithm, 1999). For all data sets, we found the most accurate algorithms by measuring precision, recall, and F-measure, which were recorded for both the reopened bugs and

non-reopened. We also identified the most important factors that contribute to reopened bugs using Rattle and the AdaBoost algorithm(Freund & Schapire, Experiments with a new boosting algorithm., 1996).

1.4 Organization of this Thesis

The rest of this thesis is organized as follows: Chapter 2 is background information of the bug fixing process, bug tracking tools, and related work to reopen the bugs. Chapter 3 is to categorize the project selected, commence data extraction methods, and divide the data into three factors: report, activity, and comments. Chapter 4 is an overview of the projects and data. Chapter 5 is choosing techniques for selecting, implementing, and predicting the outcome. Chapter 6 is a summary of total work, limitations of research, and threats to validity of future work.

2. BACKGROUND

Bugs can be defined as a flaw that prevents computer programs from behaving as intended. Bugs can be detected via human review, code analysis tools, component testing, ad hoc testing, system testing, customer reports, and employee input (Guo, Zimmermann, Nagappan, & Murphy, 2010). Whenever bugs are reported during the life cycle of fixing of bug begins. There are numerous questions to be answered when bugs are reported. One question is to whom the bug fixes should be assigned. Another is to whom the task of verifying should be delegated to. Further, who should close the bug? These three steps

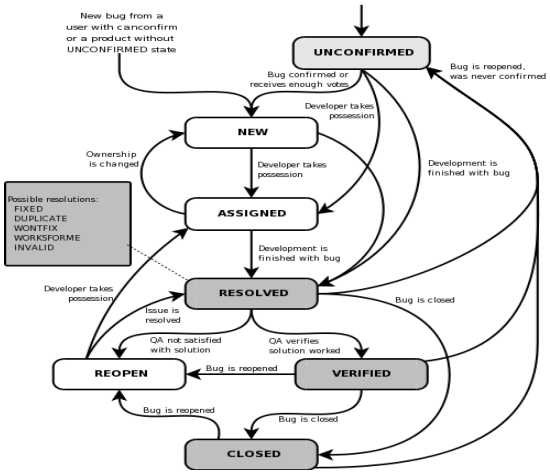

Figure 1: Bug Life cycle in Bugzilla.

are an important in limiting bugs that need reopened, though any oversight in any of the three steps may result in the need to reopen bugs. Most open source projects use a bug tracking tool to report and fix bugs.

There are numerous bug tracking tools. To name few: Mantis (http://www.mantisbt.org/manual/), Jira (https://www.atlassian.com/software/jira), Bugzilla, (http://www.Bugzilla.org/). We have extracted information of the bug reports,

history, and comments from the web-based tool Bugzilla. Bugzilla is used by many open source software systems. The diagram in Figure 1 represents states of a bug from Bugzilla. What we are interested is the reopened state of a bug. The bug reopen state has three incoming paths from resolved, verified, or closed. Reopened bug is reassigned to different person or send it back to fixer. When bug is reopened the fixer has to start the process again there is overhead of time and resources to handle the reopened bug.

There are several scenarios in which bug can be reopened. The bug can be reopened from different forms of resolved state, common scenario is bug is successfully resolved in this case bug state is changed resolved_FIXED, there can be other scenarios in which bug can be invalid, duplicate, and worksforme in this case bug states are resolved_INVALID, resolved_DUPLICATE and resolved_WORKSFORME bug can be reopened from all these forms of resolved state. Once the bug is resolved it moves to verified state asverified_FIXED. The bug can be reopened from verified_FIXED. Finally verification can be successful and bug will be closed, but even when bug is closed still bug can be reopened from closed state.

2.1 Related work

Software repositories consist of version control repositories, bug repositories, archived communication, deployment logs, and code repositories (Hassan, 2008). Software repositories hold invaluable information to understand software evolution. Common patterns of defects, predicted fixes, resources, and required time to complete software activity. Bug repositories can be used to find out bad bug fix patterns, causes, and factors responsible for a reopen state. The study was lagging in mining software

repositories due to limited access to data and difficulty in extracting data due to its complex nature (Hassan, 2008). One of the successful case studies of mining bug repository for defect patterns was by Zimmermann in his study of predicting bugs from history. Zimmermann worked on bug repositories of five Microsoft projects to find out defect patterns. He defined defect density as a number of defects in module to the total number of defects in project, which has complexity metrics as lines of codes, global variables, cyclomatic complexity, read coupling, write coupling, address coupling, fan-in, fan-out, weighted methods per class, depth of inheritance, class coupling, and number of subclasses. He found out defects correlate with complexity metrics. Other parameters for predicting defects were complexity of requirements, problem domain, set of imported classes, number of changes in components, amount of code changed to time taken. He successfully proved defects can be predicted through history of software. Also, knowledge of one project can be applied to other projects.

This Eclipse case study (Shihab E. , Ihara, Kamei, & Ibrahim, 2010)was first to address the factors responsible for reopening of bugs. In the eclipse case study bugs were categorized in four dimensions: work habit, bug report, bug fix, and people. Work habit consisted of time, day, and month at which bugs were fixed. The rationale behind the analysis was that bugs are most likely not to be fixed when they are fixed during certain period of time, and one such time being end of the week, Friday, which produced the most bugs (Sliwerski, Zimmermann, & Zeller, 2005). Factors in bug report dimensions were priority, severity, changes, description of report, and comments. Bug fix factors were time taken to fix a bug, number of files changed, and last status of a bug. His findings on the reopen bugs were: difficult to understand, take more time, increased

reopened bugs with more files changed, and indicated whether the bug would be reopened or not. Shihab defines people dimension as the name of the bug reporter and his experience, and the name of the fixer and his experience. He explains how bug reporting experience helps to write clear concise reports that explain the exact problem. The chance of reopening bugs depends on the number of reports filed, and the fixer's experience results in a decreased chance of a reopen. In the eclipse case study, a researcher used four algorithms to predict whether the bug shall reopen. The names of the algorithms were following Zero-R, NaiveBayes, Logistic Regression, and C4.5. The efficiency and accuracy of each algorithm was calculated by a confusion matrix (Kohavi & Provost, 1998). Due to the percent of reopens being fewer the reopened bugs faced the class minority problem. To solve this problem a re-sampling of training data was done using AdaBoost algorithm (Freund & Schapire, A Short Introduction to Boosting., 1999). However, testing data had same percentage of minority class. The C4.5 was the most efficient algorithm with 62.9% precision and 84.5% recall when predicting whether a bug will be re-opened and 96.8% precision and 89.6% recall when predicting if a bug will not be re-opened. To find out which factors were most responsible for reopens in a C4.5 tree was used in which the most important factors were near root of trees using this analysis of comment text, description text, and the time it took to resolve bug indicated whether the bug will be reopened or not. Threat to validity of findings centered on data that was very limited, and was limited to just one type of project.

The second study we focused on the windows operating system (Guo, Zimmermann, Nagappan, & Murphy, 2010) the study was different from that of the eclipse case study as its goal was not predict each individual bug but try to categorize

reopen bugs in common factors (Shihab E. , Ihara, Kamei, & Ibrahim, 2010). The study used a survey of Microsoft employees to categorize bugs and added a few more factors to that the previous study of Eclipse did not;one of them was a global distribution software team, with a reputation of fixer and reporter. Manual examination of bug reports was added to the survey to categorize reopen bugs. Based on a survey of employees and the manual examination of bug report, initial factors were derived: state, the opener, assignee, severity, component, type, source, and status of bug. Zimmerman derived the following causes: bugs that were difficult to reproduce, developers misunderstood the root cause, insufficient information, priority of bug increased, reputation of assignee, and bug opener related to reopen. Zimmerman's prediction model describes four states for bug probability. One, the bug will not be reopened. Two the bug will be reopened. Three, the bug will be fixed after the reopen. Four, the bug will not be fixed after the reopen. Final factors used for prediction of states were bug source of bug report, reputation of bug opener, reputation of assignee, opened by temporary employee, opener assignee same manger ,opener assignee were in same building, number of editors, number of assignee, number of component, and path changes. Threats to validity to research were restricted to Microsoft employees and the Windows operating system; therefore the results cannot be generalized.

The third study is of reopened bugs in open source software (Shihab E. , et al., 2012). The bug dimensions, factors, and algorithms were of the same as first eclipse study (Shihab E. , Ihara, Kamei, & Ibrahim, 2010)but two more projects were added: Apache and office. The number of bugs studied in eclipse study was less 1530 number of bugs studied of apache and open office was 14359 and 40173 respectively. The results

were more generalized as it was extended to two more projects with a large number of bugs. The precision for eclipse, apache, and open office was 52.1, 52.3, 78.6 and recall was 70.5, 94.1, and 89.3 respectively. Most important factors responsible for reopen varied according to projects comment text and was the most important factor for eclipse and open office while the last status was the most important factor for apache, which was responsible for reopening of the bug.

One more study on bad fixes for the eclipse project was based on bad committers (Jongyindee, Ohira, Ihara, & Matsumoto, 2011). In this bug information was extracted from Bugzilla and version control repositories. The bad fix pattern was defined in three categories. First, a bug was reopened after resolved verified and closed. Second, a bad pattern was bug marked as new and then changed to duplicate. Third, a bug was marked as duplicate but was later changed to new and resolved. Committers were categorized in four categories: developers with high number of commits, developers who support other developers, developers who perform both, developers with low number of commits. Based on this categories sixteen question were answered bad pattern rate, reopen percentage, median value of each committers bug life cycle, number of activities shown in bug tracking system, period of time in project, number of month as committers, time interval between latest bug status to commit in commit log, median review time for verify/close, average review time for verify close, number of bug resolves, number of bug assigned, number of bug fixed ,number of bug reopened, number of bug verified closed, number of time bug status was changed to new, mean bug resolving, average bug resolving time. Findings were reopen bug have longer fixing time, more experience leads to lesser bad commits, there was interrelation in bad pattern if committers performed

badly in one pattern they were more likely perform bad in all pattern the study also found out not all reopens were bad as some reopen took place as they had no knowledge of fix and hence it can be called as bad assigned.

3. DATA DESIGN

3.1 Data Preprocessing

To make our research generalized we needed bug information from projects of different software categories. The number of bug reports required was large in order to get an overall view of each project. We preferred open source systems to acquire data, for availability of data is one of the big challenges (Hassan, 2008).Previous research was limited to commercial projects of Microsoft systems (Guo, Zimmermann, Nagappan, & Murphy, 2010) or from Eclipse where data was scarce. The first task was selecting bug tracking tools. To acquire data, we selected Bugzilla because Bugzilla is an open source web-based, bug-tacking tool, which hosts the bug information of many projects that are open to the public. Its bug information is stored in bug repositories, or is available online through its website. Once we identified the bug-tracking tool, our next goal was the selection of projects that represent deferent categories of software systems. The selection of the project was following Apache in the web server category, and GNU GCC in the compiler category, Mozilla in the browser category, Net beans, and Eclipse in the integrated development category, Open office in the productivity software suite, red hat in the operating system category, and W3C in the standards organization category.

To extract data for our research, our first approach was to mine software bug repositories. We had compressed files from the Eclipse project, and we extracted important factors from thebug report, which were reporter name, fixer name, bug title, version, and priority. Parsing files was done in the Perl programming language. Of the information collected, limitations prevented us from obtaining the name of the person

who verified, closed, reopened, or changed the status from assigned to new. Also we could not extract the information of the comments made on the bugs. The repositories available were limited to a few specific projects and reports. The next approach was to create a web crawler which traveled the link from one link to other by using bug id.

This Pseudo code of our Crawler

```
feed the URL of bug report
begin with Bug id 1
repeat
        combine URL and Bug Report id
        build HTML tree and Parse the page
        get the important attributed of bug report
        replace URL with bug history
        combine bug history with big ID for new URL
        get the important attributes of bug history
        increase the Bug ID
until all bug reports are retrieved
```

Using this approach we got factors needed to categorizing and predicting reopened bugs. Data we received was in tabular form, to clean process data was done in R.Our crawler engine crawled around one million pages. The data was downloaded in an html format. The tags were removed, and the files were converted into clean text. The

variables in the bug reports were organized in a tabular format, separated by commas, to better analyze.

After the data was collected, the next step was to clean the data. We wrote a script in R for cleaning the data. Irrelevant rows were removed. The time at which the bug was reported, resolved, verified, and closed was in an integer format. It was converted into a proper format with the day, hour, and month the task was completed. Machine learning tool Weka was used to convert the csv format files into arfff format.

3.2 Data Factors

To predict and characterize defects, we need to have discrete factors to make models simpler. We have divided the factors to predict the reopen bug in three categories, bug reports, activity, and comment details. All three categories are distinctly separated in Bugzilla, and give us indications of whether or not the bug will be reopened.

3.2.1 Bug Report Factors

The first part of the bug cycle is to report a bug. The bug report is an important factor in understanding defects. Developers can use the bug report to reproduce the bug, thus instant feedback avoids the reopening of a bug. Effectively written bug reports are more likely to result in bugs that don't need reopened. Some reporters are experienced

and give clear, concise descriptions of bugs. Table 1 lists the factors, their abbreviations used in tables, their type, their source, and their description.

ID	Factor	Abbreviation Used in Table	Type	Description
1	Bug Id	Bug_Id	Numeric	Every bug had unique ID
2	Status	Status	Nominal	Status is last state of bug in process.
3	Priority	Priority	Nominal	This is priority assigned by reporter.
4	Product	Product	Nominal	This is name of Product bug was noticed.
5	Component	Component	Nominal	This is name of component bug was noticed.
6	Platform	Platform	Nominal	This is name of Platform bug was noticed.
7	Name Reported	Name_reported	Nominal	This is name of Reporter of bug.
8	Name Modified	Name_Modifie	Nominal	This is last person to modify the bug.
9	Time reported	Time_reported	Numeric	This is time at which bug was reported
10	Time modified	Time_Modifie	Numeric	This is time at which bug was modified.
11	Number of CC	Num_of_cc	Numeric	This is number of cc bug was sent.
12	Month	Month	Numeric	This is month at which bug was resolved.
13	Report Length	Report_len	Numeric	This is length of bug report.
14	Month Day	Mday	Numeric	This is month day at which bug was resolved.
15	Week Day	Wday	Numeric	This is week day at which bug was resolved.
16	Year Day	Yday	Numeric	This is year day at which bug was resolved.

Table 1: Bug Report Factors

3.2.2 Bug Activity Factors

When a bug is reported, the bug goes in to a series of activities. Mainly there are two types of activities: updating the report, changing the status of the bug report. The bug has the following states: new, resolved, verified, and closed. The activity of changing the bug to each of the states is performed by a person. We have selected the name of the person, and the time at which the change of the status was performed.

ID	Factor	Abbreviation	Type	Description
1	Name New	Name_New	Nominal	This is name of person who has changed the status of bug to new.
2	Time New	Time_New	Numeric	This is time at which bug was new.
3	Name Closed	Name_Closed	Nominal	This is name of person who has closed the bug. Certain People when bug is closed reopen rate are high.
4	Time Closed	Time_Closed	Numeric	This is time at which bug was closed.
5	Name Verified	Name_Verifie	Nominal	This is name of person who has verified a bug as fixed.
6	Time Verified	Name_Verifie	Numeric	This is time at which bug was verified.
7	Name Resolved	Name_Resolv	Nominal	This name of person who has resolved bug.
8	Time Resolved	Name_Resolv	Numeric	This is person who has verified a bug as fixed.
9	Time Taken to resolve	Time_Taken_Re	Numeric	This is time gap between resolve and reported.
10	Time Taken to Verifie	Time_Taken_Vr	Numeric	This is time gap between resolve and verified.

Table 2: Bug Activity Factors

3.2.3 Bug Comment Factors

While considering the bug comment factors, we have selected comment factors only before the bug was reopened. Since the bug can have any number of comments, we have tried to consider the last three comments before the bug was closed or resolved.

ID	Factor	Abbreviation	Type	Description
1	Comment Number 1	comm1_num	Numeric	It's the number of comment in bug fixing process.
2	Person of Comment 1	comm1_name	Nominal	Name of the person who made the comment
3	Time of Comment1	comm1_time	Numeric	Time at which comment was made.
4	Length of Comment 1	comm1_length	Numeric	Its length of comment in characters.
5	ResponseTime1	Diff_r_C1	Numeric	It is time between reported bug and first comment was made.
6	Comment Number 2	Comm2_num	Numeric	It's the number of comment in bug fixing process.
7	Person of Comment 2	Comm2_name	Nominal	Name of the person who made the comment
8	Time of Comment2	Comm2_time	Numeric	Time at which comment was made.
9	Length of Comment 2	Comm2_length	Numeric	Its length of comment in characters.
10	Response Time comment1,comment2	respon_TC1C2	Numeric	It is time between second comment and first comment was made.
11	Comment Number 3	comm1_num	Numeric	It's the number of comment in bug fixing process.

12	Person of Comment 3	Comm3_name	Nominal	Name of the person who made the comment
13	Time of Comment3	Comm3_time	Numeric	Time at which comment was made.
14	Length of Comment 3	Comm3_length	Numeric	Its length of comment in characters.
15	Response Time comment2,comment3	respon_TC3C2	Numeric	It is time between second comment and third comment was made.

Table 3: Bug Comment Factors

4. OVERVIEW OF DATA

In this section we have tried to understand background of projects .The number of bugs retrieved, number of reopens, reopen percent by components, reopen percent by products. The global distribution factor .The language the project is coded in .The size of Organization.

4.1 Eclipse Projects

Eclipse is an open source Integrated Development Environment (IDE) for programming languages like Java, Ruby, Perl, etc. It is written in Java programming language .It is globally distributed where bug reporting can be any part of world. Eclipse uses Bugzilla for bug reporting and information regarding bug fixing processes. The total number of bugs we used for the study was 55,336, out of which 6,568 were reopened. The percentage of reopen was around 12%.

	Component	TotalBugs	ReopenBugs	Reopen_Percent
1	UI	14718	1756	12
2	Core	5045	607	12
3	Debug	4062	799	20
4	SWT	3415	385	11
5	Team	2637	247	9
6	Text	2012	460	23
7	Hyades	1778	102	6
8	Resources	1187	164	14
9	cdtcore	1153	71	6
10	Ant	1053	128	12
11	UserAssistance	812	64	8
12	cdtparser	797	49	6
13	CVS	775	122	16
14	jst.j2ee	725	76	10
15	Runtime	672	83	12
16	VE	625	36	6
17	ReportDesigner	524	59	11
18	ReportEngine	483	58	12
19	Compiler	470	41	9
20	Build	462	33	7

Table 4: Reopen percent by components

The above is list of top 20 Eclipse components with highest number of bugs the highest reopen percent is of Textcomponent which is 23% while lowest percent of reopen is Hyades and VEwhich is 6%. We call difference in percentage as variation which is 17%.

4.2 Apache Projects

We have selected Apache as it comes under the server category of reopened bug analysis. The Apache software foundation hosts open source projects. It's known for its server related projects Tomcat and https. Apache products are written in C /C++ language .Its products are globally available. We extracted bug details from Bugzilla's website. The total number of bugs extracted was18, 910 out of which 2,104 were reopened. The percent was around 11 percent. The variation in reopen percent is 17.The variation in

reopen is 16%.

4.3 GNU Projects

The GNU compiler system falls into the compiler category of analysis. It was first developed to handle C programming codes and later it was extended to handle C++ codes. It is written in C++ language and is available globally. The total number of bugs we extracted was 3,663 out of which 76 were reopened and the percent being around 2%.The only product categorized under this GNU system was gcc. The variation in reopen of component is 33%.

4.4 Mozilla Projects

We selected the Mozilla system as it comes under the category of browser. Their products are written in C/C++, java, html and it is globally distributed. The total number of bugs we extracted was41, 790 out of which 5,105 were reopened and the reopened percent being around 12%.Variation shown in reopen % of products is 18%.

4.5 Red hat Projects

Red hat is a Linux based operating system. Linux uses Bugzilla to report bugs. It is written in python and it's available globally. Its bug information is available to the public. Extracting data helped us to understand bugs of operating systems. The total numbers of bugs extracted were 25,810 out of which 1,915 were reopened, and the percent being around 8%.Variation shown product reopen % is 40.

4.6 Net beans Projects

Net Bean is an integrated development environment for java but also works for C, C++and Perl. Its written java and is globally available. We extracted 81,053 bugs out of which 8,543 were reopened and the percent wash around 11. Variation shown in ropen percent is 16.Variation shown in reopen percent is 17%.

4.7 Open Office Projects

Open office is open source productivity suite used for writing documents. It is written in C++ and Java and is globally available. We extracted 42,598 bugs out of which 4,698 were reopened, and percent of reopen being around 11%. Variation shown in reopen % in product and component is around 5 %.

4.8 W3C

W3C is web standards organization. It is written in html, css, JavaScript and is globally available. We extracted 7,954 bugs out of which 629 were reopened, and the reopened percent being around 8%. The variation in reopen percent of product and component was around 16 %.

4.9 Projects Overview

Project Name	Date of first bug used for analysis	Date of last bug used for analysis	Total Bugs	Reopen Bugs	Reopen %	Language	Globally Distributed
Eclipse	2001-10-10	2006-05-06	55,336	6,568	12%	Java	Yes
Apache	2001-01-10	2011-09-19	18,910	2,104	11%	C/C++	Yes
GCC	1999-08-03	2002-07-03	3,663	76	2%	C++	Yes
Mozilla	1998-04-07	2002-09-02	41,790	5,105	12%	C/C++, Java, HTML	Yes
Red hat	1998-11-08	2001-12-04	25,810	1,915	8%	Python	Yes
Net beans	1998-06-29	2007-02-02	81,053	8,543	11%	Java	Yes
Open Office	2003-06-24	2010-03-10	42,598	4,698	11%	Java /C++	Yes
W3c	2002-07-15	2012-09-19	7,954	629	8%	HTML, CSS, JavaScript	Yes

Table 5: Overview of Projects

One of the reopen patterns we found was not matter what project language was written in, what organization it was the reopen % was around 10. The variation in reopen percent of products and components was around 15 %.

5. METHODOLOGY, RESULTS, AND ANALYSIS

5.1 Methods

In this section we summarize methods algorithms used for analysis.

5.1.1K-fold Cross Validation

Since we have the data, the next step is to understand it. To understand the data, we have to select two tools and implement the machine learning algorithms of Weka and Rattle. Machine learning algorithms gain knowledge from training data and implemented their knowledge on test data. There are several types of procedures where the procedure for gaining knowledge from training data and applying rules on testing data .K-fold cross validation is one such method. Below is diagram of 3 fold cross validation method.

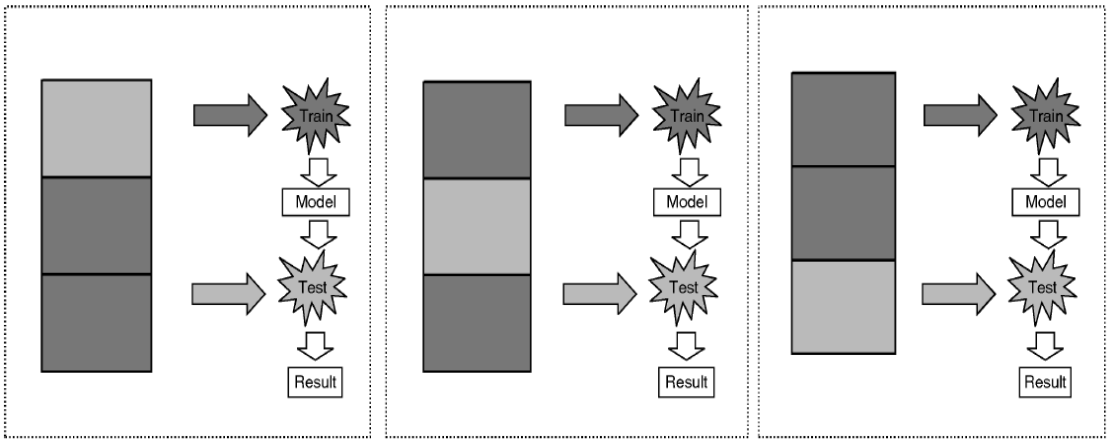

Figure 2:3-fold cross validation (P. Refaeilzadeh, 2009)

The above diagram shows three phases. Each diagram shows three divisions of data, two of which are used by the machine learning program to create a model. The last division of data, depicted in lighter gray above, is set against the model to test the accuracy. The repeated process should now allow the program to able to predict the reopen probability of bugs. We have implemented a 10-fold, cross-validation procedure that is the same as the one above, but instead the data is compartmentalized into ten divisions and the modal and test is repeated ten times. Using the 10-fold cross validation, we have analyzed data from 8 projects using 4 different algorithms.

5.1.2 Confusion Matrix

We have used Decision trees and Bayesian methods to predict whether defects will be reopened or not. The Decision trees used for predictions are the C 4.5 decision tree (Quinlan, 1986), Alternating decision tree (Freund & Mason, The alternating decision tree learning algorithm, 1999), and Bayesian methods. The Bayesian methods used for prediction are NaiveBayes(Bayes, 1763) and Bayesian Network. Predicted results are given in the form of a confusion matrix (Kohavi & Provost, 1998). Its matrix has actual and predicted results.

Actual / Predicted	Not Reopened	Reopened
Not-Reopened	A	B
Reopened	C	D

Table 6: Confusion matrix example

Total number of bugs present were A+B+C+D, and the actual reopened bugs were A+B, and not reopened were C+D. The algorithm predicted A+C as not reopened, and the

prediction was correct for A bugs. Similarly, it predicted B + D as reopened. It was correct for D bugs.

1) Accuracy of prediction = (A+D)/ (A+B+C+D)

2) Reopened precision = D/ (B+D)

3) Reopened recall = D/ (C+D)

4) Not Reopened precision = A/(D+C)

5) Not Reopened recall = A/ (B+A)

5.1.3 C4.5 Decision tree

Decision tress used for predictions are C4.5 and Alternating Decision tree. Every node works as a decision and data is split into multiple classes, or if the node is a leaf node, the decision has been made whether the bug will be reopened or not. In general, to build a decision tree, four terms are required.

1. Attribute value description: Fixed collection of properties.

2. Predefined Target class: Class to be predicted.

3. Discrete Classes: Class with distinguishing features which can help with prediction.

4. Sufficient Data: Set of training examples.

Two common terms are related to selection of a top node: Entropy and information gain. The entropy of each attribute can be defined as the measure of impurity with difference between probabilities of positive to probability of negative (Mitchel, 1997). Formula to

calculate entropy is below. Consider sample data S with probability of positive class $p_p$and probability of negative class pn.

Entropy(S) = - $p_p\log_2(p_p) - p_n\log_2(p_n)$

The information gain, Gain(S,A) of an attribute A,

$$Gain(S, A) = Entropy(S) - \sum_{v=1}^{n} \frac{|S_v|}{|S|} * Entropy(S_v)$$

Attribute with best information gain is selected as root node.

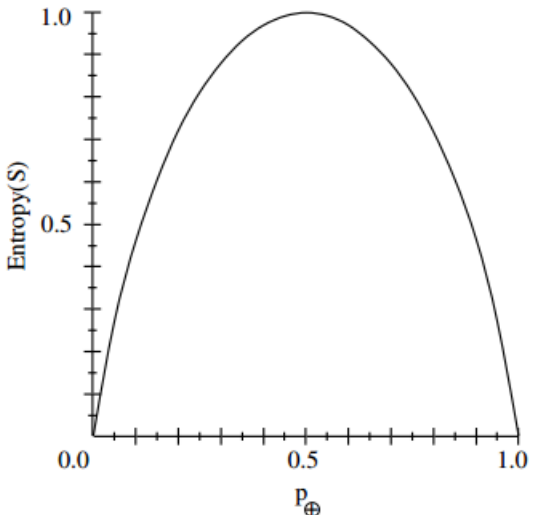

Figure 3: Entropy distribution of Binary class (Mitchel, 1997)

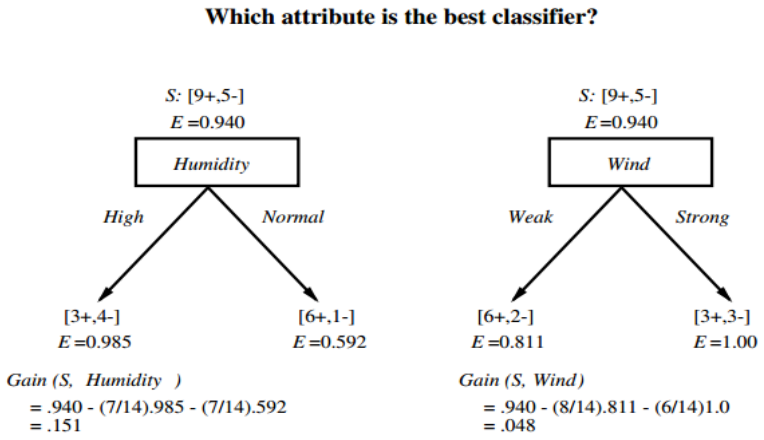

Figure 4: Best Attribute selection (Mitchel, 1997)

Top-Down Induction of Decision Trees ID3C4.5

1. A ← the "best" decision attribute for next node
2. Assign A as decision attribute for node
3. For each value of A create new descendant
4. Sort training examples to leaf node according to the attribute value of the branch
5. If all training examples are perfectly classified stop, else iterate over new leaf nodes

Figure 5is an example of C4.5 decision tree. Based on the highest information gain the variable Time_taken_re is selected as the root node.If the bugs are resolved within 258 days, they are easily understood, and most of them do not get ropened. So the calssifer prdicts them as not reopen.It is correct 2,368 times, and incoreect 116 times.There is high diffrence between postive class and negative class, so it is a root node of a C4.5 decision

tree that is the best attribute to clasify.The next best attribute is the component name "compare." If report length is less than 300 lines, it will be reopened.

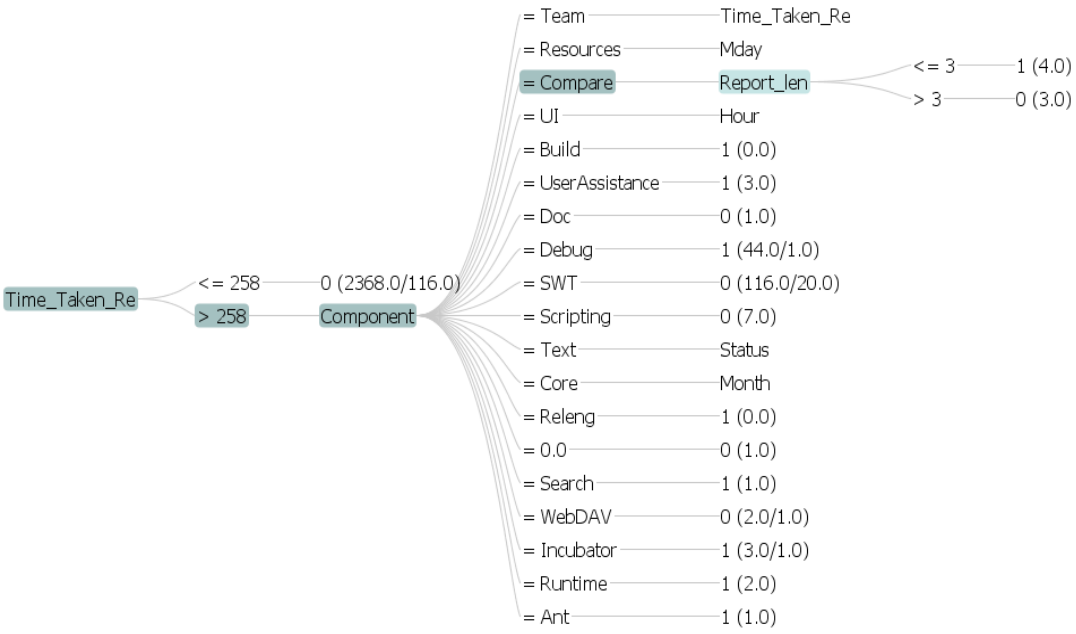

Figure 5 :C4.5 Decision tree for Reopen of eclipse Bugs

5.1.4 NaiveBayes Classifier

Bayesian theorem is popular way of predicting outcomes of events. The Bayesian theorem calculates probabilities of given data and predicts outcomes of a given class with classifier with the highest maximum probability. For instance, a given data "D" and outcome of the class of C and their probabilities as p(D) and p(C).Bayesian theorem can be stated as follows:

Class C can take the value "0" for not reopen and"1" for reopen.

The D data set has six attributes: Status, Product, Component, Platform, Version, and Priority.

The probability of Class C given events Data D is given by Bayes rule.

Bayes'srule: $P(C \mid D) = P(D \mid C) \times P(C)/P(D)$

Bayes classifier is which given data D selects the value of C such that maximizes the value ofP(C | D)

C can be 0 and 1 whichever makes P(C | D) maximize

Bayes classifier can be stated as $argmax\ P(C \mid D) = argmaxP(D \mid C) \times P(C)/ P(D)$

Data can be of several attributes a1 , a2 , ...aN so Bayes classifier can be restated as:

$argmax\ P\ (C \mid a1\ ,\ a2\ ,\ ....aN) = argmax\ P\ (a1\ ,\ a2\ ,\ ....aN \mid C) \times P\ (C)/\ P\ (a1\ ,\ a2\ ,\ ....aN)$

Computation of argmax P(a1 , a2 , ...aN | C) is expensive so in Bayes theorem, class conditional independence is observed.

So Bayes classifier can be restated as:

$argmax\ P(a1\ ,\ a2\ ,\ ....aN \mid C) = P(a1|C)*P(a2|C).....P(aN|C) \times argmax\ P\ (C)$

Denominator is common for every class so it is being ignored.

No.	1: Status Nominal	2: Product Nominal	3: Component Nominal	4: Platform Nominal	5: Priority Nominal	6: **Yes_no_reopened** Nominal
1	VERIFIEDFIXED	Platform	Team	AllAll	P3	0
2	RESOLVEDFIXED	Platform	SWT	AllLinux	P5	0
3	RESOLVEDFIXED	JDT	UI	AllWindows2000	P3	1
4	RESOLVEDFIXED	Platform	UI	AllAll	P5	0
5	VERIFIEDFIXED	JDT	UI	AllAll	P3	0
6	VERIFIEDFIXED	JDT	UI	AllAll	P3	0
7	RESOLVEDWORKSFORME	JDT	Core	AllWindowsNT	P3	1
8	RESOLVEDFIXED	Platform	UI	AllLinux	P2	1
9	RESOLVEDFIXED	JDT	UI	AllWindows2000	P3	1
10	RESOLVEDFIXED	JDT	UI	AllWindows2000	P2	0

Table 7: Learning Data for NaiveBayes Classifier

Consider Data=D= RESOLVEDFIXED, JDT, UI, AllWindows2000, P2

To find whether bug will be reopened or not, we have to calculate previous probabilities of RESOLVEDFIXED, JDT, UI, AllWindows2000, and P2 for class 0 and 1 from training set.

Probabilities are calculated independent whichever class has maximum probability

NaiveBayes will select that class

Let us consider class value 0

Total times class 0 appears is 6 times out of 10 cases so prior probability of class 0 is *P (C=0)* =(6/10) .Out of 6 cases of class 0 RESOLVEDFIXED as status appears 3 times.

*P (RESOLVEDFIXED | C=0) P (C=0) = (3|6) * (6|10)*

Total times class 0 appears is 6 times out of 10 cases so prior probability of class 0 is *P (C=0)* =(6/10) .Out of 6 cases of class 0 *JDT*as productappears 3 times.

*P (JDT | C=0) P (C=0) = (3|6) * (6|10)*

Total times class 0 appears is 6 times out of 10 cases so prior probability of class 0 is *P (C=0)* =(6/10) .Out of 6 cases of class 0 *UI* as componentappears 4 times.

*P (UI | C=0) P (C=0) = (4|6) * (6|10)*

Total times class 0 appears is 6 times out of 10 cases so prior probability of class 0 is *P (C=0)* =(6/10) .Out of 6 cases of class 0 *AllWindows2000* as operating system appears 1 times.

*P (AllWindows2000 | C=0) P (C=0) = (1|6) * (6|10)*

Total times class 0 appears is 6 times out of 10 cases so prior probability of class 0 is *P (C=0)* =(6/10) .Out of 6 cases of class 0 *P2* as priorityappears 1 times.

*P (P2 | C=0) P (C=0) = (1|6) * (6|10)*

Multiplying all probabilities to get P (D | C=0)

*= (3|6) * (6|10)*(3|6) * (6|10)*(4|6) * (6|10)*(1|6) * (6|10)*(1|6) * (6|10)*

*= (3|10) * (3|10) * (4|10) * (1|10) * (1|10)*

Similarly for P (D | C=1)

*P (D | C=1) = (3|10) * (3|10) * (3|10) * (1|10) * (1|10)*

So

P (D | C=0)> P (D | C=1)

argmax P (D | C=0)

It can be seen that when we input value of class=0 that bug will be not reopened. The value of *P (D | C=0)* becomes maximum since we have binary target class. The only other class we have is reopened class=1. Its probability is *P (D | C=1)* so NaiveBayes will compute as not reopened.

5.1.5 Bayesian Network Classifier

The Bayesian Network is a directed acyclic graph defining a joint probability distribution over a set of variables. Each node is a random variable, and a conditional probability distribution is associated with each node defined as *P (N| Parents (N))*.

The Chow-Liu algorithm (Friedman, Geiger, & Goldszmidt, 1997) describes a procedure for constructing a Bayesian network fromthe data. This procedure reduces the

problem to one of constructing a maximum likelihood tree to finding a maximal weighted spanning tree in a graph. The algorithm is as follows:

- Compute probability distribution $I_{Pd}(X_i, X_j)$ between each edge.X_i, X_j. I_{Pd} is the mutual information function.

$$I_P(X,Y) = \sum_{x,y} P(x,y) * log \frac{P(x,y)}{P(x) * P(y)}$$

- Build a complete undirected graph in which the vertices are the variables in X.
- Annotatethe weight of an edge connecting X_i,X_j by I_{Pd}
- Build a maximum weight spanning tree
- Transform the resulting undirected tree to a directed one by choosing a root variable and set the direction of all edges to be outward from it

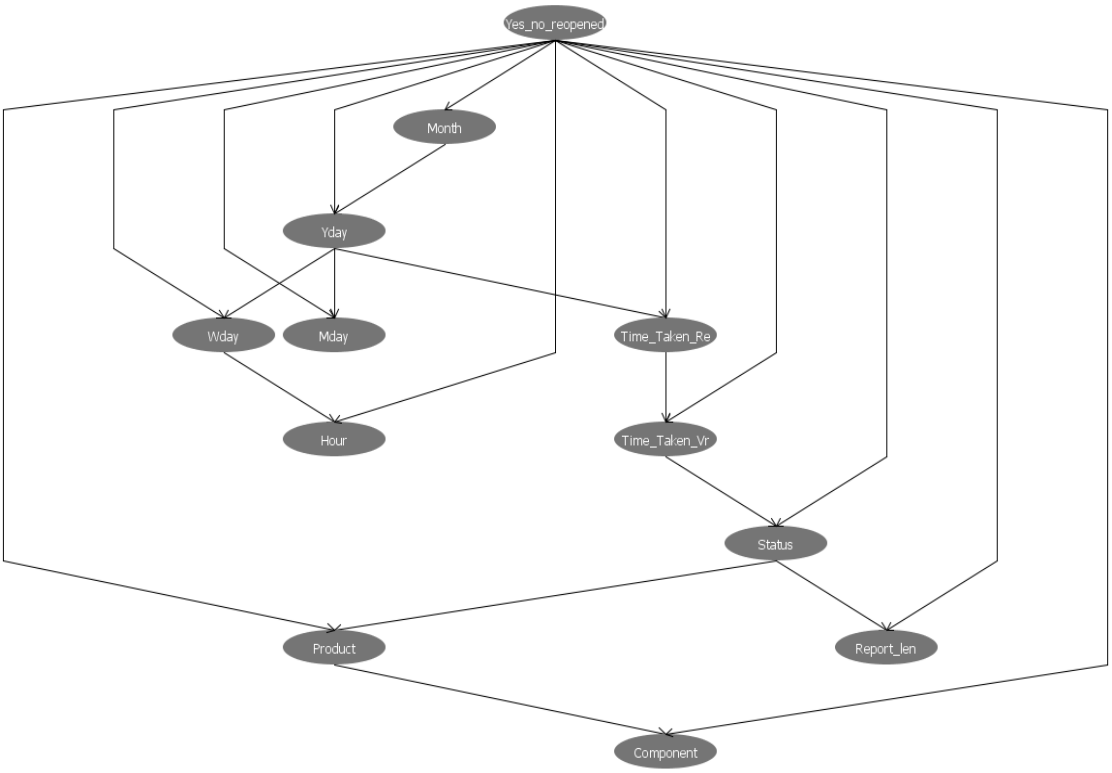

Figure 6: Bayesian Network Generated by Chow-Liu algorithm (Friedman, Geiger, & Goldszmidt, 1997)

Consider Node Product it has two parent nodes Yes_no_reopened and Status

P (Product| Yes_no_reopened , Status). Network is minimum spanning tree generated by Chow-Liu algorithm. The conditional probability distribution among the components is shown in Table 8.

Yes_no_reopened	Status	Platform	PDE	JDT	Equinox
0	CLOSEDFIXED	0.939	0.007	0.047	0.007
0	RESOLVEDFIXED	0.614	0.06	0.324	0.002
0	RESOLVEDWONTFIX	0.588	0.011	0.401	0.001
0	RESOLVEDINVALID	0.903	0.025	0.071	0.001
0	VERIFIEDFIXED	0.103	0.003	0.893	0.001
0	0.0	0.5	0.167	0.167	0.167
0	RESOLVEDWORKSFORME	0.569	0.039	0.391	0.002
0	CLOSEDWONTFIX	0.14	0.06	0.78	0.02
0	CLOSEDINVALID	0.083	0.083	0.75	0.083
0	VERIFIEDWORKSFORME	0.071	0.071	0.786	0.071
0	CLOSEDWORKSFORME	0.1	0.1	0.7	0.1
0	NEW	0.5	0.167	0.167	0.167
0	ASSIGNED	0.167	0.167	0.5	0.167
0	VERIFIEDWONTFIX	0.5	0.167	0.167	0.167
0	VERIFIEDINVALID	0.167	0.167	0.5	0.167
1	CLOSEDFIXED	0.864	0.045	0.045	0.045
1	RESOLVEDFIXED	0.75	0.005	0.241	0.005
1	RESOLVEDWONTFIX	0.762	0.009	0.213	0.015
1	RESOLVEDINVALID	0.836	0.008	0.148	0.008
1	VERIFIEDFIXED	0.235	0.006	0.753	0.006
1	0.0	0.25	0.25	0.25	0.25
1	RESOLVEDWORKSFORME	0.596	0.006	0.391	0.006
1	CLOSEDWONTFIX	0.038	0.038	0.885	0.038
1	CLOSEDINVALID	0.1	0.1	0.7	0.1
1	VERIFIEDWORKSFORME	0.045	0.045	0.864	0.045
1	CLOSEDWORKSFORME	0.167	0.167	0.5	0.167
1	NEW	0.85	0.05	0.05	0.05
1	ASSIGNED	0.577	0.038	0.346	0.038
1	VERIFIEDWONTFIX	0.25	0.25	0.25	0.25
1	VERIFIEDINVALID	0.25	0.25	0.25	0.25

Table 8: Conditional Probability distribution for Node Product for Parents Yes_no_reopened and Status

5.1.6 AdaBoost Classifier

AdaBoost prediction method is developed by (Freund & Schapire, Experiments with a new boosting algorithm., 1996). This method identified important variables for predicting reopened bugs. AdaBoost is based on an ensemble of weak classifiers into strong classifier. Figure 7is the algorithm for AdaBoost(Freund & Schapire, Experiments with a new boosting algorithm., 1996).

Given training data $(x_1, y_1), ..., (x_m, y_m)$
$y_i \in \{-1, +1\}$, $x_i \in X$ is the object or instance, y_i is the classification.

for $t = 1, \ldots, T$
    create distribution D_t on $\{1, \ldots, m\}$
    select weak classifier with smallest error ϵ_t on D_t
        $\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$
        $h_t : X \to \{-1, +1\}$
output single classifier $H_{\text{final}}(x)$

Figure 7 :AdaBoost algorithm

The data is split into training data for each row of training data $x_i \in X$ we have for each of x_i we have predicted output $y_i \in \{1,-1\}$.AdaBoost maintains a probability distribution x_i which can be considered at a point which represent feature in space. If m is the number of attributes, consider $D_t(x_i)$ as probability distribution where t represents iteration. The probability $D_t(1) = 1/m$, and with each iteration, probability distribution is updated. Let the weak classifier be denoted by h_t where t is iteration. The output given by this classifier is predicted class, where the predicted class is denoted $h_t(xi)$. By comparing predicted class $h_t(x_i)$ to actual class y_i we can calculate error rate e_t. The trust in classifier is given by α_t. We calculate α_t by formula $\alpha_t = 1/2\ ln(1-e_t)/\ e_t$. Final classifier H is aggregation of classifier of each iteration. Weighting is set to amount of trust in classifier.

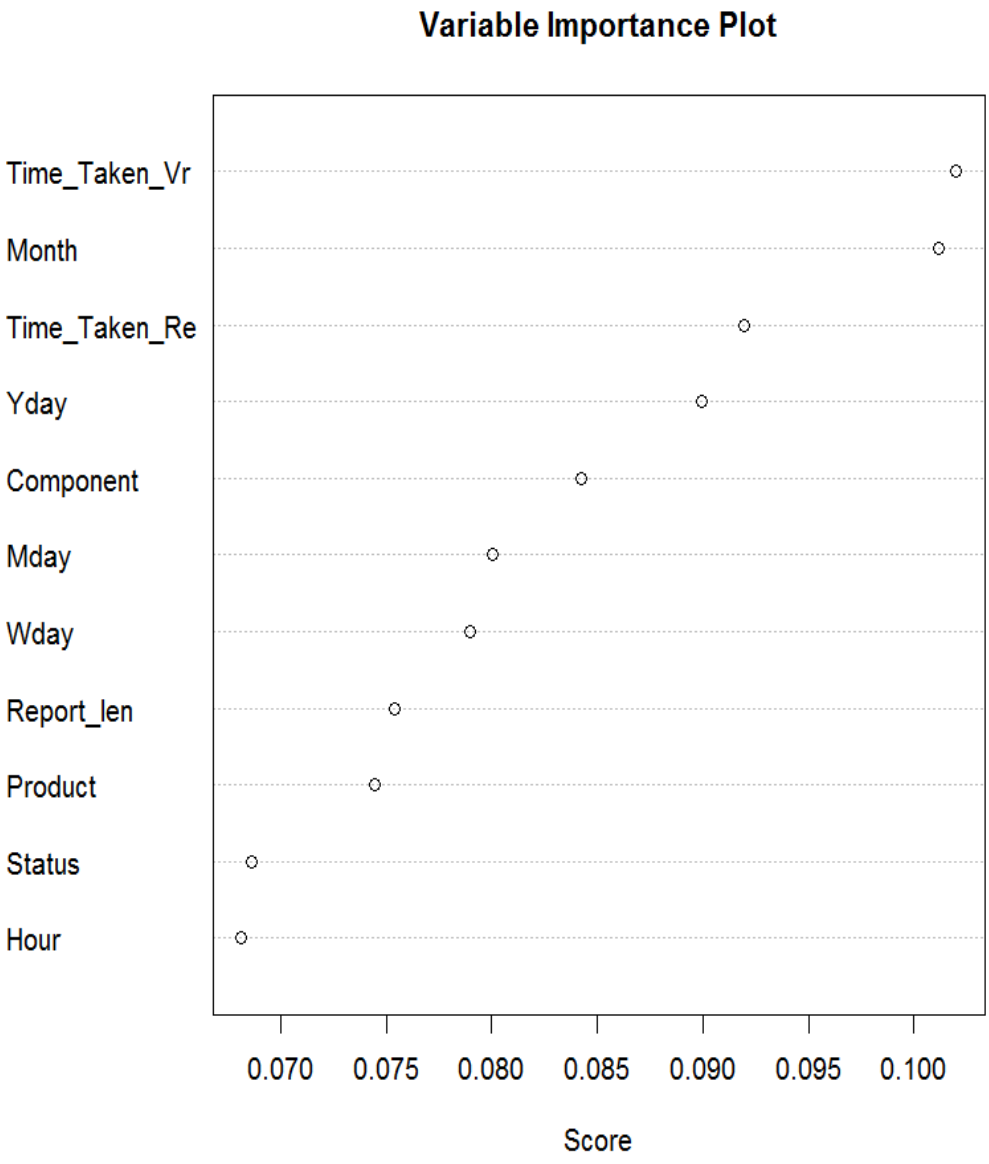

Figure 8 : Variable importance graph generated by AdaBoost algorithm using Rattle

Above is variable importance graph generated by AdaBoost algorithm using Rattle. The Y axis shows the name of variable and higher the value of y is for the variable more important it is .The X axis shows the reduction error rate when the variable is introduced in algorithm. AdaBoost reweights its instances for incorrectly classified instances so it biased towards in correctly classified examples.

5.1.7 Alternating Decision tree Classifier

ADtree(Drauschke, 2008)differs from C4.5 in how it assigns value -α and +α to its decisions.C4.5 has uniform weight to instance while Weight W is associated with each instance.

The ADtree algorithm (Drauschke, 2008) takes the following inputs:

n: Total number of positive and negative instances.

W: 1/n Initial weights at root node.

α(node) : root node, $\alpha(node) = \frac{1}{2} ln \frac{W_+(true)}{W_-(true)}$

$W_+(c)$: sum of all weights of positively classified instances satisfying condition *c*

$W_-(c)$: sum of all weights of negatively classified instances satisfying condition *c*

Data set: Variablesx_j, j =1 to n

Target Class: y_j {+1,-1}j =1 to n

SetofClassifiers:C_j *decision stumps*

h_p: Previous condition of classifier.

$W_+(+h_p)$: sum of all weights where correctly classified positive instance by h_p

$W_-(-h_p)$: *sum* of all weights where correctly classified negative instance by h_p

$W_+(-h_p)$: sum of all weights where incorrectly classified positive instance by h_p

$W_-(+h_p)$: sum of all weights where incorrectly classified negative instance by h_p

$W_*(-h_p)$: sum of all weights where precondition classifies class – 1.

Z_{pj}: condition to select best classifier when precondition is root node.

$$Z_{pj} = 2\left(\sqrt{W_+(+h_p) * W_-(+h_p)} + \sqrt{W_+(-h_p) * W_-(-h_p)}\right) + W_*(-h_p)$$

$W_+(h_p \wedge +c_j)$: is sum of all weights where correctly classified positive instance by c_j which satisfies previous condition h_p

$W_-(h_p \wedge -c_j)$: is sum of all weights where correctly classified negative instance by c_j which satisfies previous condition h_p.

$W_+(h_p \wedge -c_j)$: is sum of all weights where incorrectly classified positive instance by c_j which satisfies previous condition h_p.

$W_-(hp \wedge +cj)$: is sum of all weights where incorrectly classified negative instance by c_j which satisfies previous condition h_p.

Z_{pj} :condition to select best classifier c_j when precondition h_p.

$$Z_{jp} = 2\left(\sqrt{W_+(h_p \wedge +c_j) * W_-(h_p \wedge +c_j)}\right.$$

$$\left. + \sqrt{W_+(h_p \wedge -c_j) * W_-(h_p \wedge -c_j)}\right) + W_*(-h_p)$$

$\mathcal{E}$: is error rate associated is set to $\mathcal{E}$=1.

α_t^+: Classifying power of classifier$c_j$when$W_+(h_p \wedge +c_j)$

$$\alpha_t^+ = \frac{1}{2} ln\left(\frac{W_+(h_p \wedge c_j) + \varepsilon}{W_-(h_p \wedge c_j) + \varepsilon}\right)$$

α_t^-: Classifying power of classifier$c_j$when$W_+(h_p \wedge -c_j)$

$$\alpha_t^- = \frac{1}{2} ln\left(\frac{W_+(h_p \wedge -c_j) + \varepsilon}{W_-(h_p \wedge +c_j) + \varepsilon}\right)$$

$W_{t+1}(n)$: Update of weight

$$W_{t+1}(n) = W_t(n) * e^{-r_t(x_n)y_n}$$

Where

$r_t(x_n) = \alpha_t^+$ if $h_p(x_n) = +1$ and$c_j(x_n) = +1$

$r_t(x_n) = \alpha_t^-$ if $h_p(x_n) = +1$ and$c_j(x_n) = -1$

$r_t(x_n) = 0$, if$h_p(x_n) = -1$.

Algorithm

1. Input (x_n, y_n)

2. Set weights of Instances $W=1/n$

3. Calculate α (node)

4. Repeat for 1 to T

    Select classifier C_j which minimizes Z_{pj}

    Update weights of instances $W_{t+1}(n)$

5. $class(x) = sign(\sum_{t=1}^{T} r_t(x))$

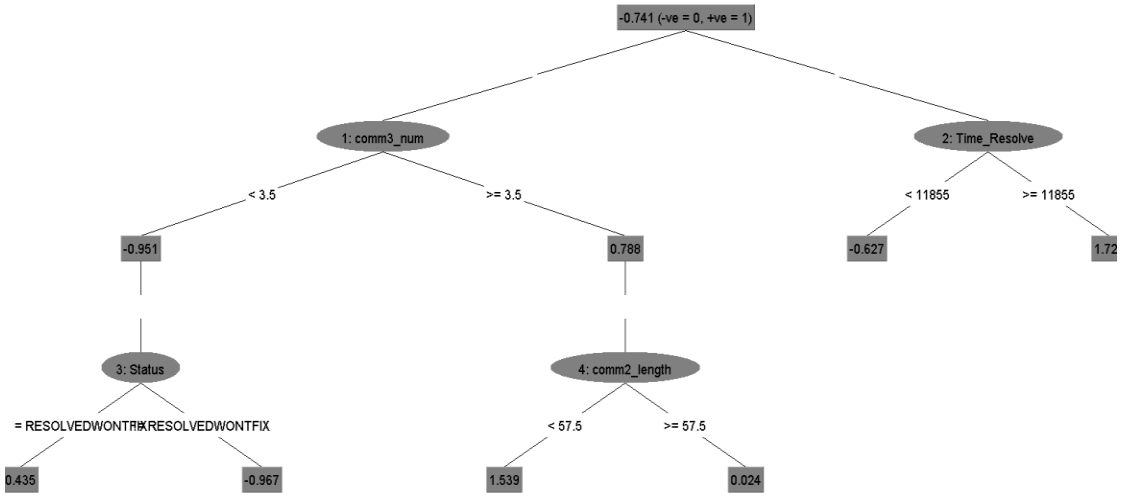

Figure 9 : Alternating decision tree generated by 4 number of boosting Iteration.

Above is example of ADtree generated by Weka. First value of root node is calculated that is half the log of weight of positive instances to weight of negative instances. Value of root node is -0.741. First Iteration decision stump classifier C1=comm3_num <3.5 is chosen as minimizes ZnodeC1 its classifying power -αC1 and +αC1 are calculated.Second Iteration decision stump classifier C2=Time_Resolve< 11855is chosen as minimizes ZnodeC2its classifying power -αC2 and +αC2 are calculated.Third Iteration decision stump classifier C3=Status = RESOLVEDWONTFIX is chosen with precondition C1=comm3_num < 3.5 as minimizes ZC1C3 its classifying power -αC3 and +αC3 are calculated. Fourth Iteration decision stump classifier C4=Status != RESOLVEDWONTFIX is chosen with precondition C1=comm3_num > 3.5 as minimizes ZC1C4 its classifying power -αC4 and +αC4 are calculated.

5.2 Results

5.2.1 Eclipse Project Results

Test Mode	10-fold Cross-validation
Instances	55336
Attributes	46

Table 9: Input description of Eclipse project data to Algorithms

Algorithm	Target Class	Instances	Correctly classified	Precision	Recall	F-Measure
BayesNet	0	48768	42429	0.963	0.870	0.914
BayesNet	1	6568	4960	0.439	0.755	0.555
NaiveBayes	0	48768	39641	0.955	0.813	0.878
NaiveBayes	1	6568	4703	0.340	0.716	0.461
ADtree	0	48768	47057	0.949	0.965	0.957
ADtree	1	6568	4061	0.708	0.618	0.658
C4.5	0	48768	47764	0.956	0.979	0.968
C4.5	1	6568	4255	0.809	0.661	0.728

Table 10: Efficiency of algorithms in predicting Reopened bugs for Eclipse

Above table shows efficiency of algorithms: BayesNet,NaiveBayes, ADtree, and C4.5in predicting reopen of eclipse bugs. Most efficient algorithm in F-measure of not reopened bugs was C4.5. It showed F-measure of 0.968 for not reopened bug while it showed F-measure of 0.728 for reopened bugs. While most efficient in recall of reopen was BayesNet. It was able to predict 75% of reopened bugs but showed less accuracy with F-measure 0.555 for reopened bugs which was low compared to C4.5 algorithm. Recall of reopened bugs can be increased in C4.5 algorithm by increasing cost sensitivity of reopened class.

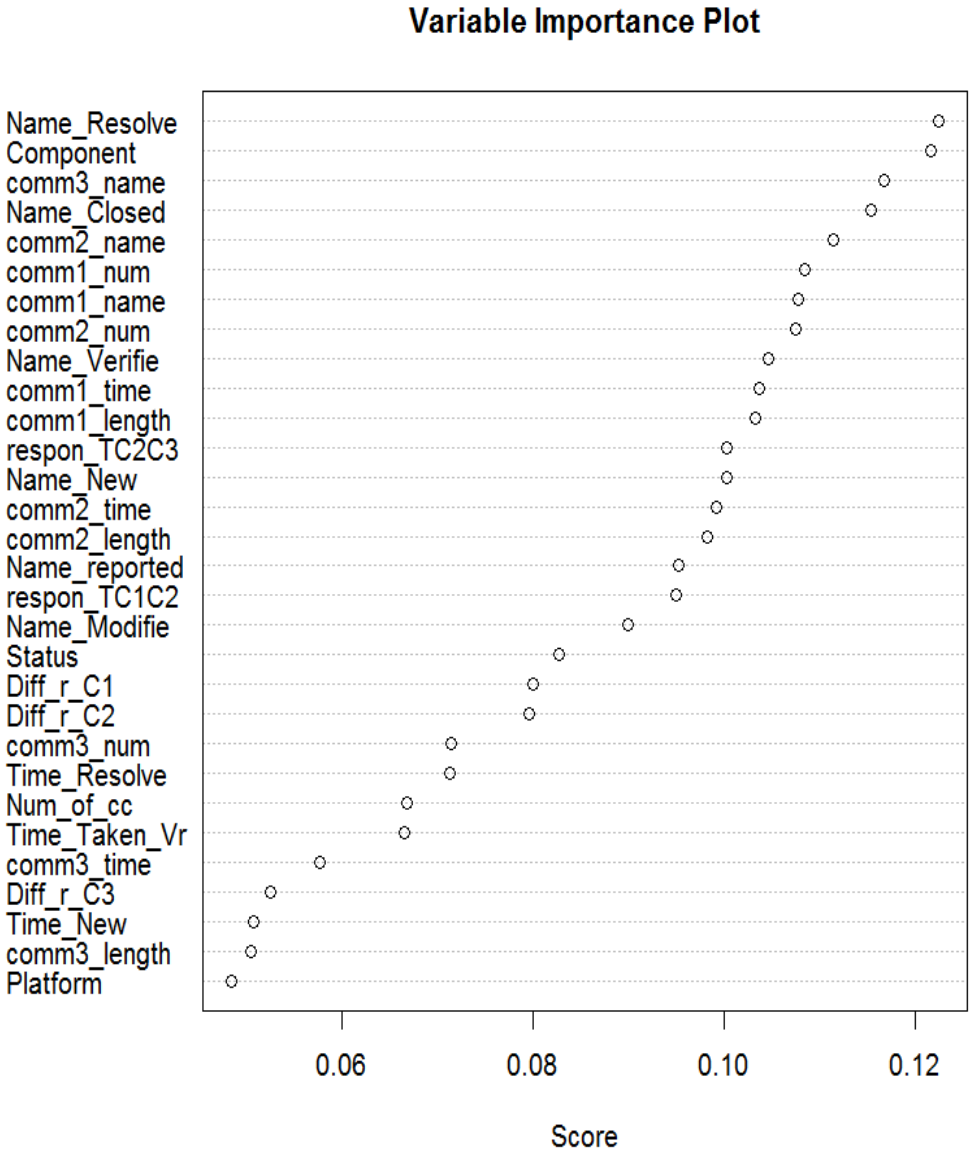

Figure 10: Important variable responsible for reopen in Eclipse Projects

Figure 10 is variable importance graph generated by AdaBoost algorithm using Rattle.The Y axis shows the name of variable and higher the value of y is for the variable more important it is.The X axis shows the reduction error rate when the variable is introduced in algorithm. AdaBoost reweights its instances for incorrectly classified instances so it biased towards in correctly classified examples. Name of the person resolved, closed, verified, Component, comment name and number were important factors responsible for reopen.

5.2.2 Open Office Project Results

Test Mode	10-fold Cross-validation
Instances	36880
Attributes	46

Table 11: Input description of Open Office project data to Algorithms

Algorithm	Target Class	Instances	Correctly classified	Precision	Recall	F-Measure
BayesNet	0	30798	27428	0.954	0.891	0.921
BayesNet	1	6082	4748	0.585	0.781	0.669
NaiveBayes	0	30798	25971	0.959	0.843	0.897
NaiveBayes	1	6082	4975	0.508	0.818	0.626
ADtree	0	30798	30573	0.949	0.965	0.957
ADtree	1	6082	3854	0.932	0.634	0.759
C4.5	0	30798	30235	0.946	0.982	0.964
C4.5	1	6082	4368	0.886	0.718	0.793

Table 12: Efficiency of algorithms in predicting Reopened bugs for Open Office

Above table shows efficiency of algorithms: BayesNet, NaiveBayes, Ad tree, and C4.5 in predicting reopen of eclipse bugs. Most efficient algorithm in F-measure of not reopened bugs was C4.5 with F-measure of 0.964 while it showed F-measure of 0.793 for

reopened bugs. While most efficient in recall of reopen was NaiveBayes it was able to predict 82% of reopened bugs but showed less accuracy with F-measure 0.626 for reopened bugs which was low compared to C4.5 algorithm. Recall of reopened bugs can be increased in C4.5 algorithm by increasing cost sensitivity of reopened class.

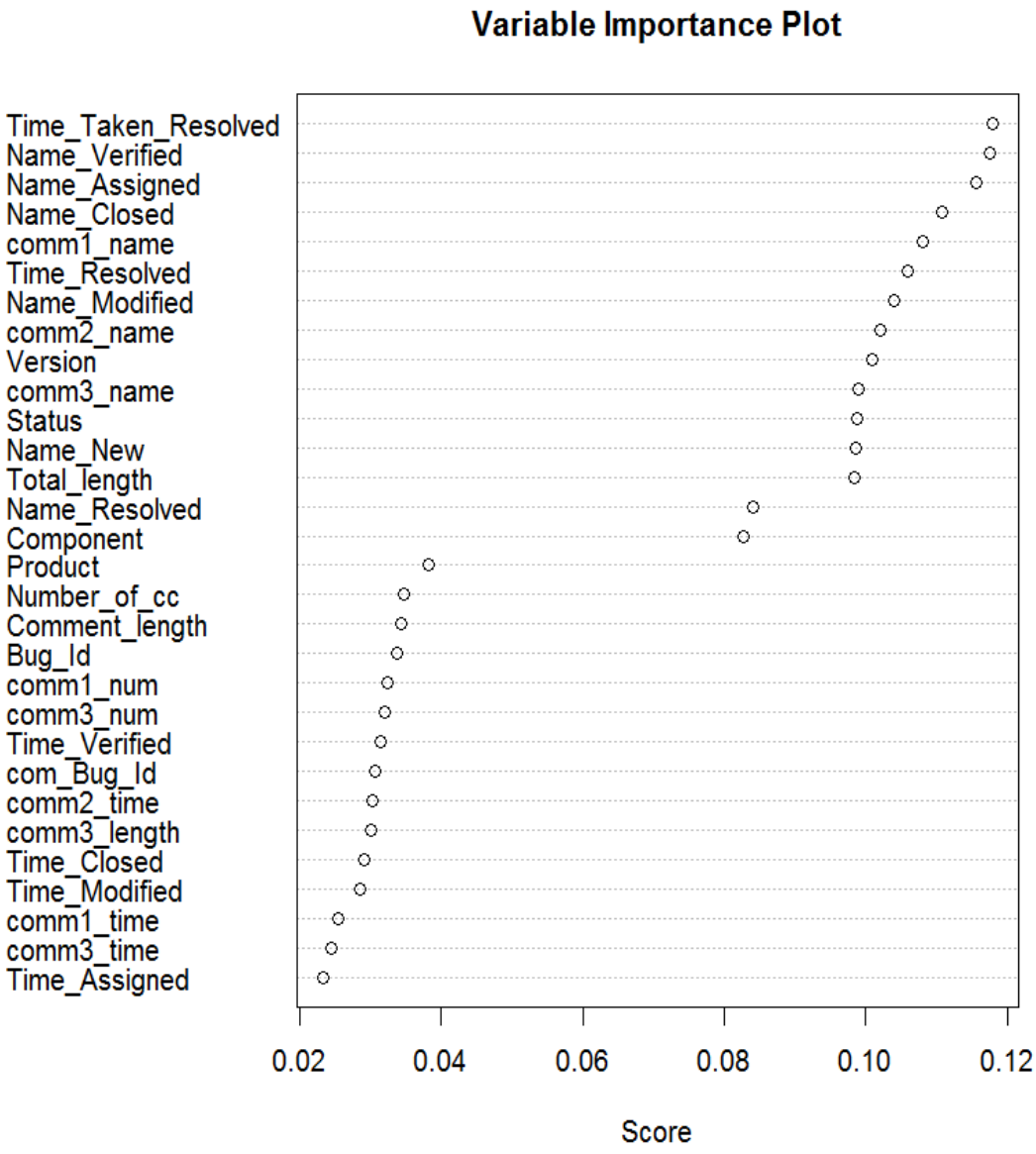

Figure 11: Important variable responsible for reopen in Open Office Projects

Time taken to resolve, name of the person closed, verified, Component name and number were important factors responsible for reopen.Above is variable importance graph generated by AdaBoost algorithm using Rattle .The Y axis shows the name of variable and higher the value of y is for the variable more important it is .The X axis shows the reduction error rate when the variable is introduced in algorithm. AdaBoost reweights its instances for incorrectly classified instances so it biased towards in correctly classified examples.

5.2.3 Apache Project Results

Test Mode	10-fold Cross-validation
Instances	18755
Attributes	46

Table 13: Input description of apache project data to Algorithms

Algorithm	Target Class	Instances	Correctly classified	Precision	Recall	F-Measure
BayesNet	0	16806	13416	0.961	0.798	0.872
BayesNet	1	1949	1405	0.293	0.721	0.471
NaiveBayes	0	16806	14379	0.955	0.813	0.461
NaiveBayes	1	1949	960	0.283	0.493	0.360
ADtree	0	16806	15791	0.960	0.940	0.950
ADtree	1	1949	1294	0.560	0.664	0.608
C4.5	0	16806	15689	0.953	0.934	0.943
C4.5	1	1949	1176	0.513	0.603	0.554

Table 14 : Efficiency of algorithms in predicting Reopened bugs for Apache

Above table shows efficiency of algorithms: BayesNet, NaiveBayes, Ad tree, and C4.5 in predicting reopen of eclipse bugs. Most efficient algorithm in F-measure of not reopen bugs was ADtreewith F-measure of 0.954 while it showed F-measure of 0.608 for reopened bugs. The most efficient in recall of reopen was BayesNet.It was able to predict 72% of reopened bugs but showed less accuracy with F-measure 0.471 for reopened bugs which was low compared to ADtree algorithm. Recall of reopened bugs can be increased in ADtree algorithm by increasing cost sensitivity of reopened class.

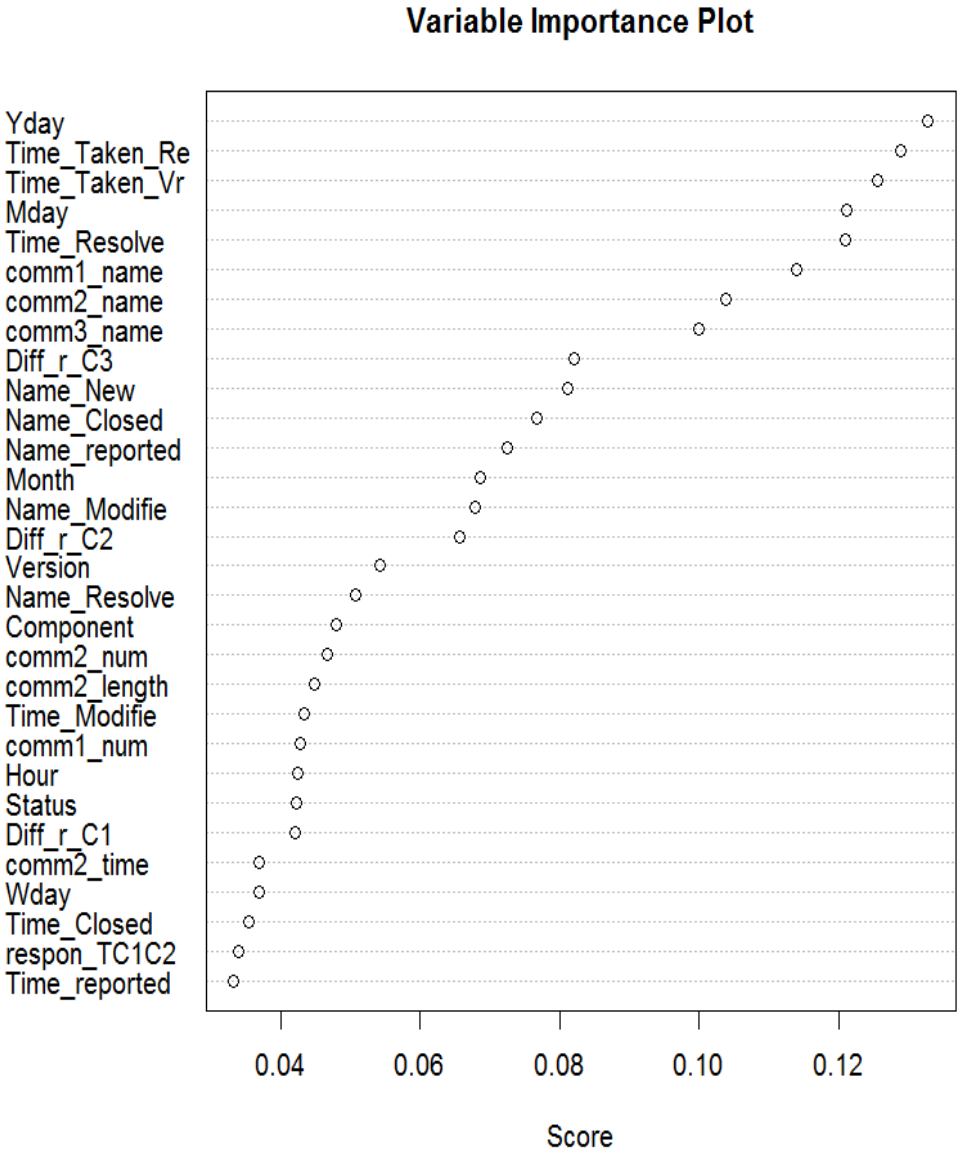

Figure 12: Important variable responsible for reopen in Apache Projects

Name of the person resolved, closed, verified, Component name and number were important factors responsible for reopen. Above is variable importance graph generated by AdaBoost algorithm using Rattle.The Y axis shows the name of variable and higher the value of y is for the variable more important it is.The X axis shows the reduction error rate when the variable is introduced in algorithm. AdaBoost reweights its instances for incorrectly classified instances so it biased towards in correctly classified examples.

5.2.4 Net beans Project Results

Test Mode	10-fold Cross-validation
Instances	37541
Attributes	46

Table 15: Input description of Net beans project data to Algorithms

Algorithm	Target Class	Instances	Correctly classified	Precision	Recall	F-Measure
BayesNet	0	33059	28991	0.957	0.877	0.915
BayesNet	1	4392	3098	0.432	0.705	0.536
NaiveBayes	0	33059	16686	0.978	0.505	0.666
NaiveBayes	1	4392	4091	0.197	0.915	0.324
ADtree	0	33059	32570	0.943	0.985	0.964
ADtree	1	4392	2431	0.833	0.554	0.665
C4.5	0	33059	31219	0.957	0.944	0.951
C4.5	1	4392	2987	0.618	0.680	0.648

Table 16: Efficiency of algorithms in predicting Reopened bugs for Net beans.

Above table shows efficiency of algorithms: BayesNet, NaiveBayes, ADtree, and C4.5in predicting reopen of eclipse bugs. Most efficient algorithm in F-measure of reopen bugs was ADtree. It showed F-measure of 0.964 for not reopened bug while it showed F-measure of 0.665 for reopened bugs. Most efficient in recall of reopen was BayesNet.It was able to predict 91.5% of reopened bugs but showed less accuracy withF-measure 0.324 for reopened bugs, which was low compared to ADtree algorithm. Recall of reopened bugs can be increased inADtree algorithm by increasing cost sensitivity of reopened class.

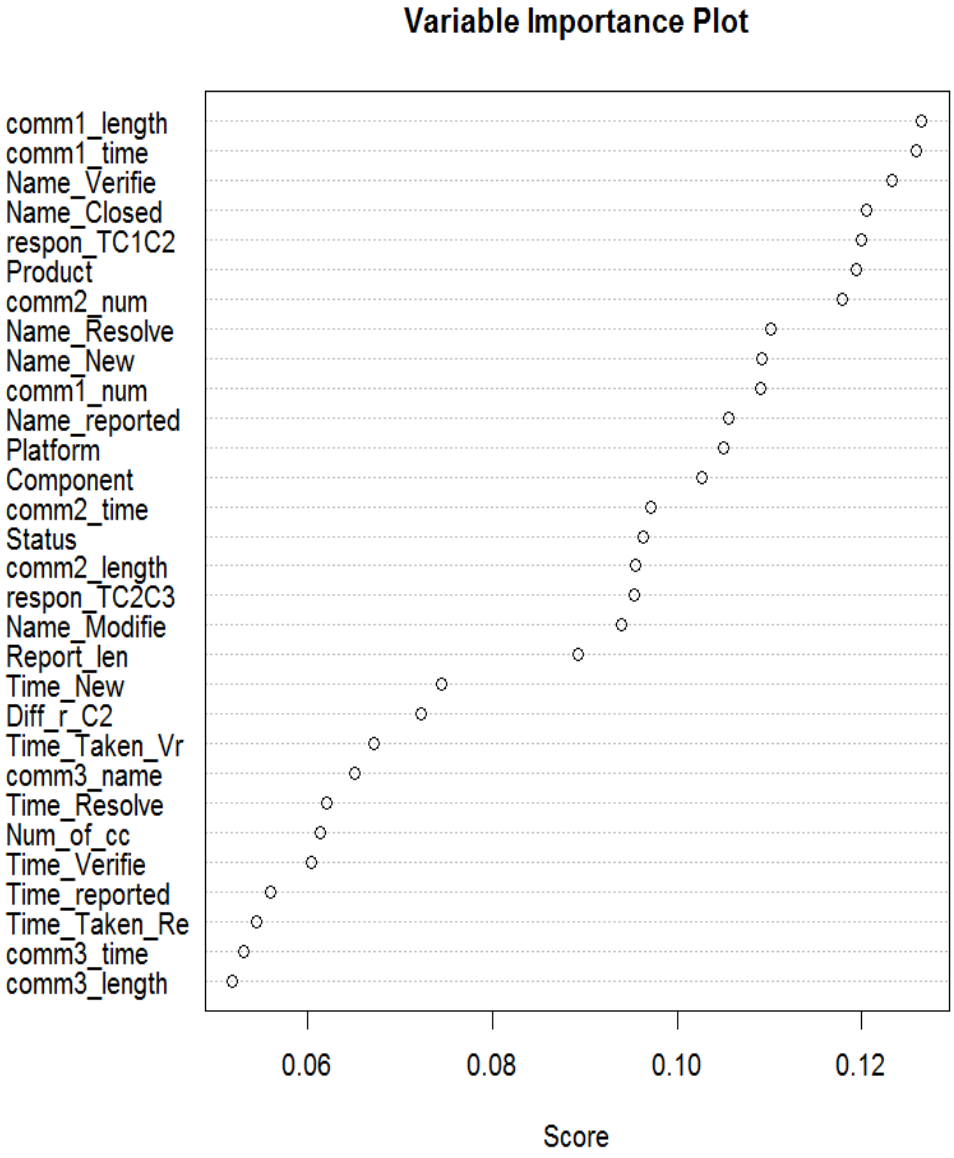

Figure 13: Important variable responsible for reopen in Net Beans Projects

Time taken to resolve, name of the person closed, verified, Component name ,Product name ,comment name and number were important factors responsible for reopen. Above is variable importance graph generated by AdaBoost algorithm using Rattle .The Y axis shows the name of variable and higher the value of y is for the variable more important it is .The X axis shows the reduction error rate when the variable is introduced in algorithm. AdaBoost reweights its instances for incorrectly classified instances so it biased towards in correctly classified examples.

5.2.5 Red hat Project Results

Test Mode	10-fold Cross-validation
Instances	25810
Attributes	46

Table 17: Input description of Red hat project data to Algorithms

Algorithm	Target Class	Instances	Correctly classified	Precision	Recall	F-Measure
BayesNet	0	23895	18042	0.969	0.755	0.849
BayesNet	1	1915	13332	0.185	0.696	0.329
NaiveBayes	0	23895	14417	0.975	0.603	0.761
NaiveBayes	1	1915	1544	0.140	0.806	0.239
ADtree	0	23895	22659	0.963	0.948	0.956
ADtree	1	1915	1044	0.458	0.545	0.498
C4.5	0	23895	23085	0.953	0.934	0.943
C4.5	1	1915	970	0.507	0.525	0.525

Table 18: Efficiency of algorithms in predicting Reopened bugs for Red hat.

Above table shows efficiency of algorithms: BayesNet, NaiveBayes, Ad tree, and C4.5 in predicting reopen of eclipse bugs. Most efficient algorithm in F-measure of

reopen bugs was C4.5.It showed F-measure of 0.943 for not reopened bug while it showed F-measure of 0.525 for reopened bugs. Most efficient in recall of reopen was NaiveBayes.It was able to predict 80.6% of reopened bugs but showed less accuracy with F-measure 0.239 for reopened bugs, which was low compared to C4.5 algorithm. Recall of reopened bugs can be increased in C4.5 algorithm by increasing cost sensitivity of reopened class.

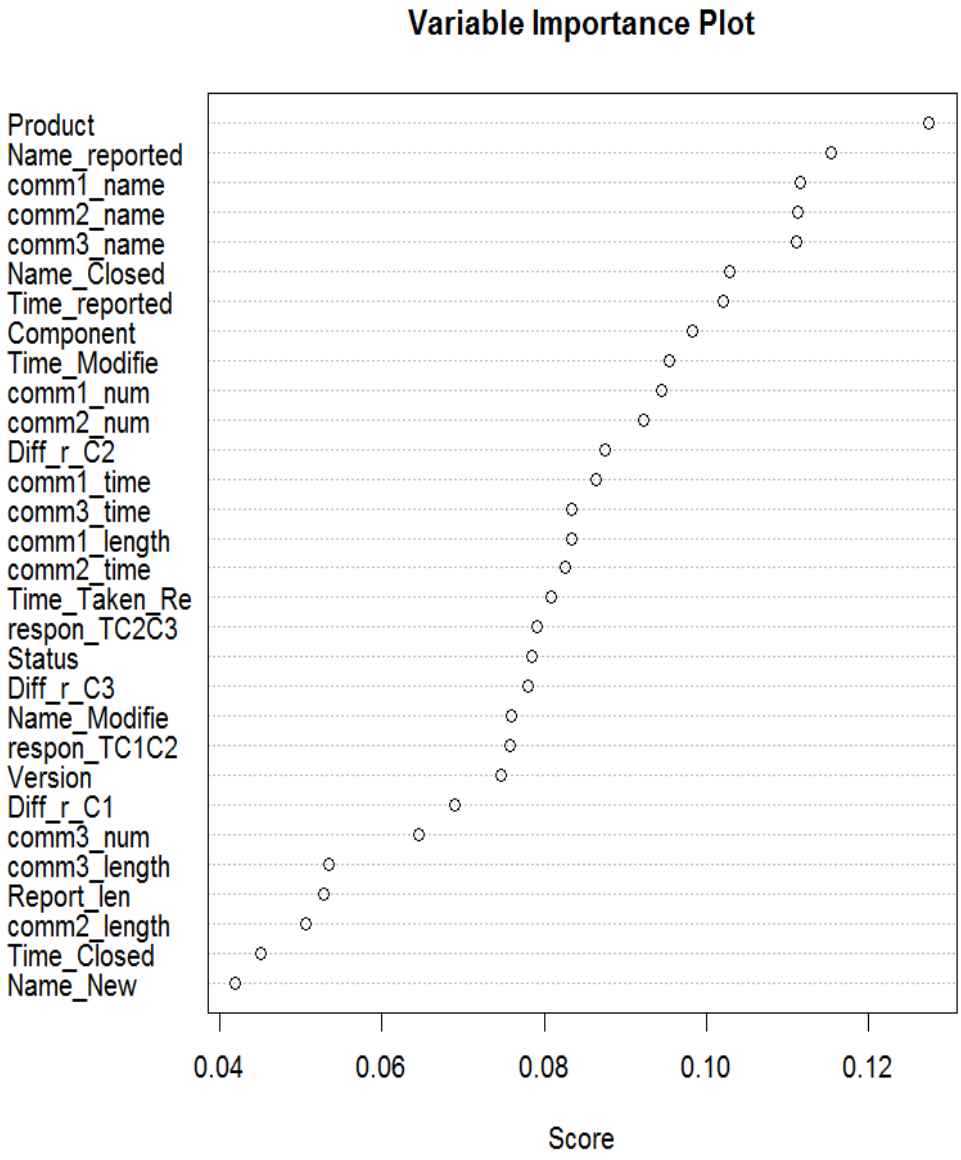

Figure 14: Important variable responsible for reopen in Red Hat Projects

Time taken to resolve, name of the person closed, verified, Component name and number were important factors responsible for reopen. Above is variable importance graph generated by AdaBoost algorithm using Rattle.The Y axis shows the name of variable and higher the value of y is for the variable more important it is .The X axis shows the reduction error rate when the variable is introduced in algorithm.AdaBoost reweights its instances for incorrectly classified instances so it biased towards in correctly classified examples.

5.2.6 Mozilla Project Results

Test Mode	10-fold Cross-validation
Instances	41736
Attributes	46

Table 19: Input description of Mozilla project data to Algorithms

Algorithm	Target Class	Instances	Correctly classified	Precision	Recall	F-Measure
BayesNet	0	36686	29286	0.952	0.798	0.868
BayesNet	1	5051	3560	0.325	0.705	0.445
NaiveBayes	0	36686	26929	0.943	0.734	0.825
NaiveBayes	1	5051	3349	0.256	0.663	0.369
ADtree	0	36686	32889	0.929	0.897	0.912
ADtree	1	5051	2535	0.400	0.502	0.445
C4.5	0	36686	33603	0.920	0.916	0.918
C4.5	1	5051	2146	0.410	0. 425	0.418

Table 20: Efficiency of algorithms in predicting Reopened bugs for Mozilla.

Above table shows efficiency of algorithms: BayesNet, NaiveBayes, Ad tree, and C4.5 in predicting reopen of eclipse bugs. Most efficient algorithm in F-measure of reopen bugs was ADtree.It showed F-measure of 0.912 for not reopened bug while it showed F-measure of 0.445 for reopened bugs. Most efficient in recall of reopen was BayesNet.It was able to predict 70.5% of reopened bugsbut showedsimilarly low accuracy with F-measure 0.445 for reopened bugs compared to ADtree algorithm.Recall of reopened bugs can be increased inADtree algorithm by increasing cost sensitivity of reopened class.

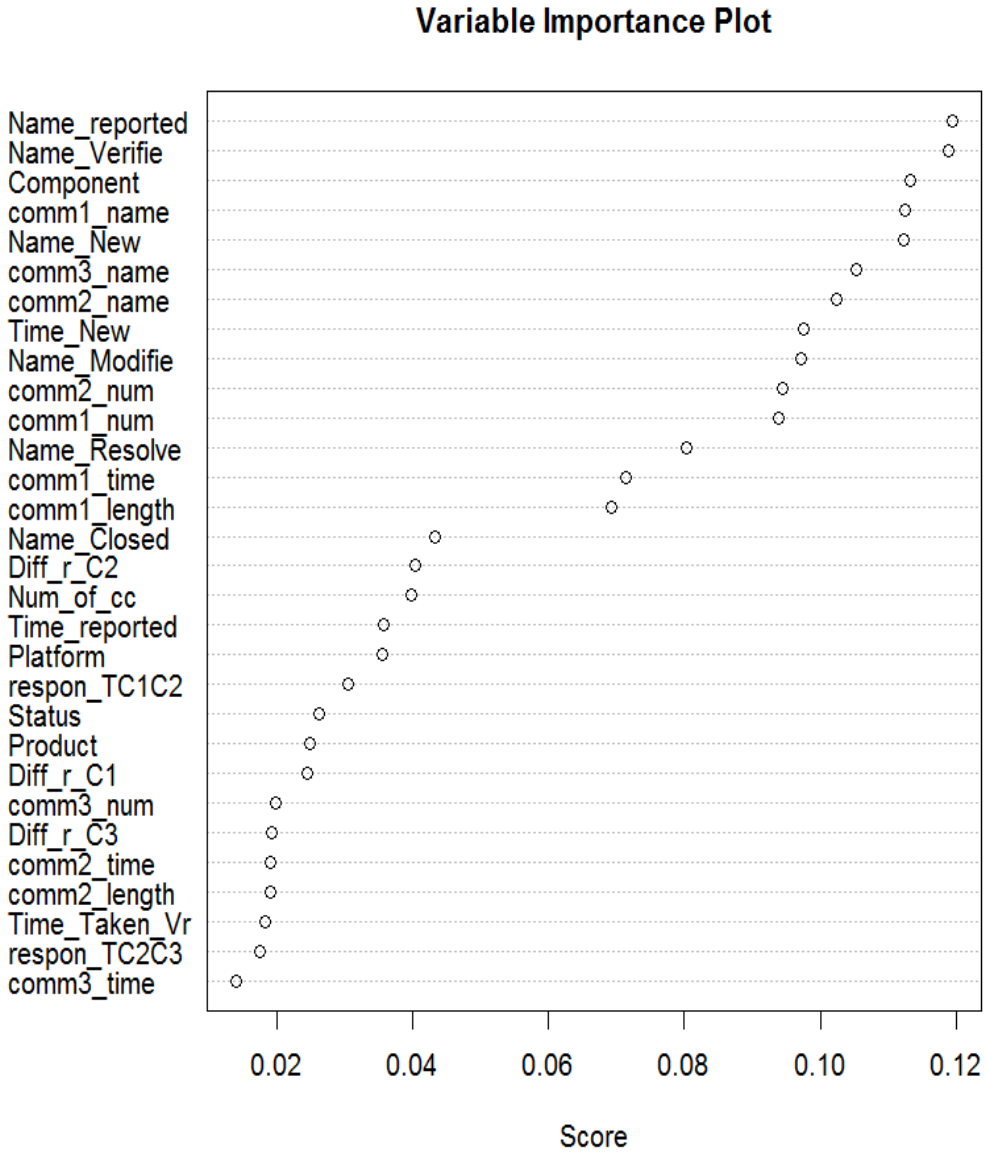

Figure 15:Important variable responsible for reopen in Mozilla Projects

Name reported, name of the person closed, name verified, Component name and number were important factors responsible for reopen.Above is variable importance graph generated by AdaBoost algorithm using Rattle.The Y axis shows the name of variable and higher the value of y is for the variable more important it is.The X axis shows the reduction error rate when the variable is introduced in algorithm. AdaBoost reweights its instances for incorrectly classified instances so it biased towards in correctly classified examples.

5.2.7 W3C Project Results

Test Mode	10-fold Cross-validation
Instances	7318
Attributes	46

Table 21: Input description of W3C project data to Algorithms

Algorithm	Target Class	Instances	Correctly classified	Precision	Recall	F-Measure
BayesNet	0	6745	5646	0.974	0.837	0.900
BayesNet	1	537	421	0.277	0.735	0.402
NaiveBayes	0	6745	5975	0.955	0.813	0.461
NaiveBayes	1	537	313	0.289	0.546	0.378
ADtree	0	6745	3569	0.993	0.995	0.994
Ad tree	1	537	406	0.450	0.709	0.550
C4.5	0	6745	6404	0.972	0.949	0.961
C4.5	1	537	390	.534	0.681	0.598

Table 22:Efficiency of algorithms in predicting Reopened bugs for W3C.

Above table shows efficiency of algorithms: BayesNet, NaiveBayes, ADtree, and C4.5in predicting reopen of eclipse bugs. Most efficient algorithm in F-measure of reopen bugs was C4.5.It showed F-measure of 0.961 for not reopened bug while it showed F-measure of 0.598 for reopened bugs. Most efficient in recall of reopen was BayesNet.It was able to predict 73.5 % of reopened bugsbut showed less accuracy with F-measure 0.402 for reopened bugs which was low compared to C4.5 algorithm.Recall of reopened bugs can be increased in C4.5 algorithm by increasing cost sensitivity of reopened class.

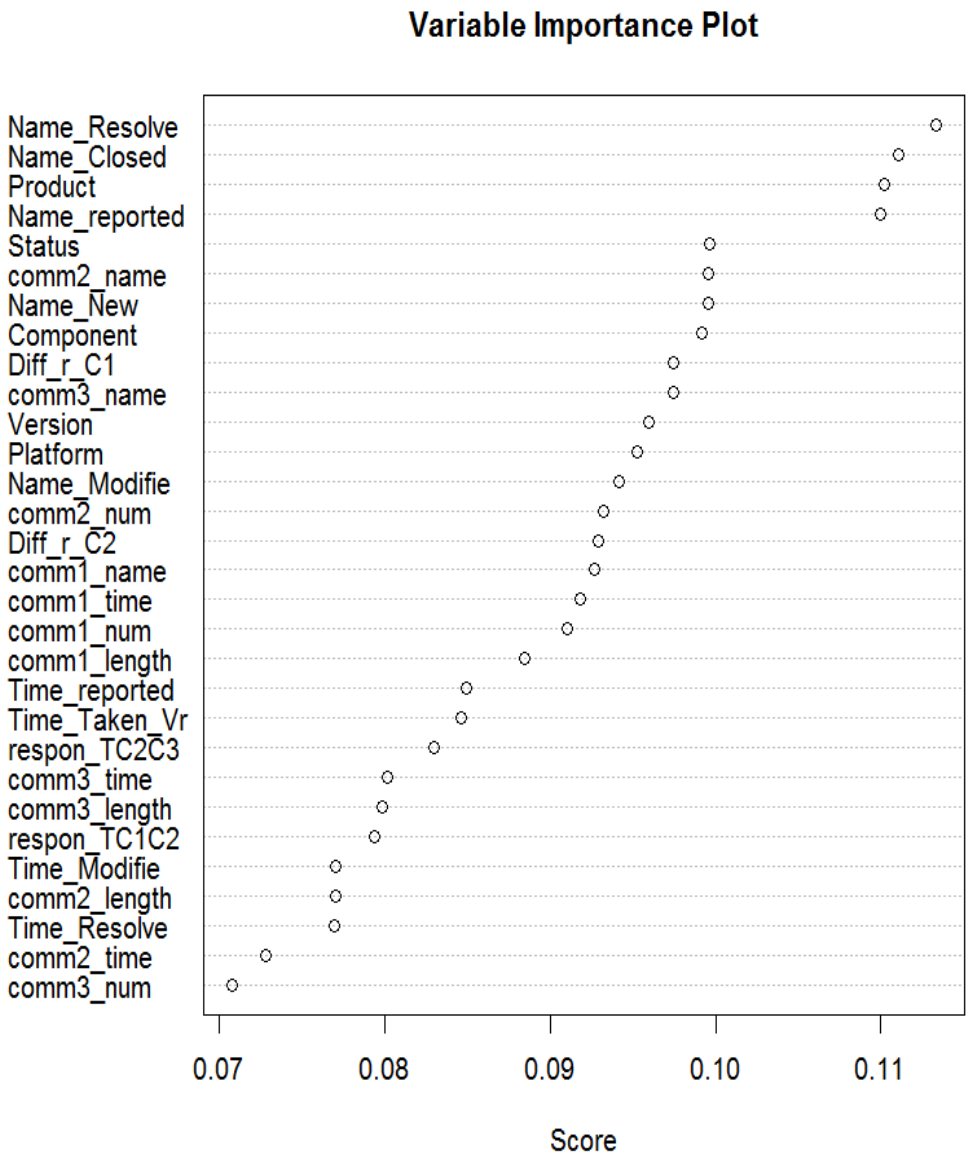

Figure 16: Important variable responsible for reopen in W3C Projects

Name of person who resolved, name of the person closed, verified, Component name and number were important factors responsible for reopen.Above is variable importance graph generated by AdaBoost algorithm using Rattle.The Y axis shows the name of variable and higher the value of y is for the variable more important it is.The X axis shows the reduction error rate when the variable is introduced in algorithm. AdaBoost reweights its instances for incorrectly classified instances so it biased towards in correctly classified examples.

5.2.8 GCC Project Results

Test Mode	10-fold Cross-validation
Instances	3663
Attributes	46

Table 23: Input description of GCC project data to Algorithms

Algorithm	Target Class	Instances	Correctly classified	Precision	Recall	F-Measure
BayesNet	0	3587	3375	1.000	0.941	0.970
BayesNet	1	76	76	0.264	1.000	0.418
NaiveBayes	0	3587	3379	0.955	0.813	0.461
NaiveBayes	1	76	74	0.283	0.493	0.360
ADtree	0	3587	3569	0.993	0.995	0.994
ADtree	1	76	50	0.735	0.658	0.694
C4.5	0	3587	3574	0.989	0.996	0.993
C4.5	1	76	38	.754	0.500	0.598

Table 24: Efficiency of algorithms in predicting Reopened bugs for GCC.

.

Above table shows efficiency of algorithms: BayesNet, NaiveBayes, Ad tree, and C4.5 in predicting reopen of eclipse bugs. Most efficient algorithm in F-measure of reopen bugs was ADtree.It showed F-measure of 0.994 for not reopened bug while it showed F-measure of 0.694 for reopened bugs. Most efficient in recall of reopen was BayesNet.It was able to predict 100% of reopened bugs but showed less accuracy with F-measure 0.418 for reopened bugs which was low compared to ADtreealgorithm. Recall of reopened bugs can be increased in ADtreealgorithm by increasing cost sensitivity of reopened class.

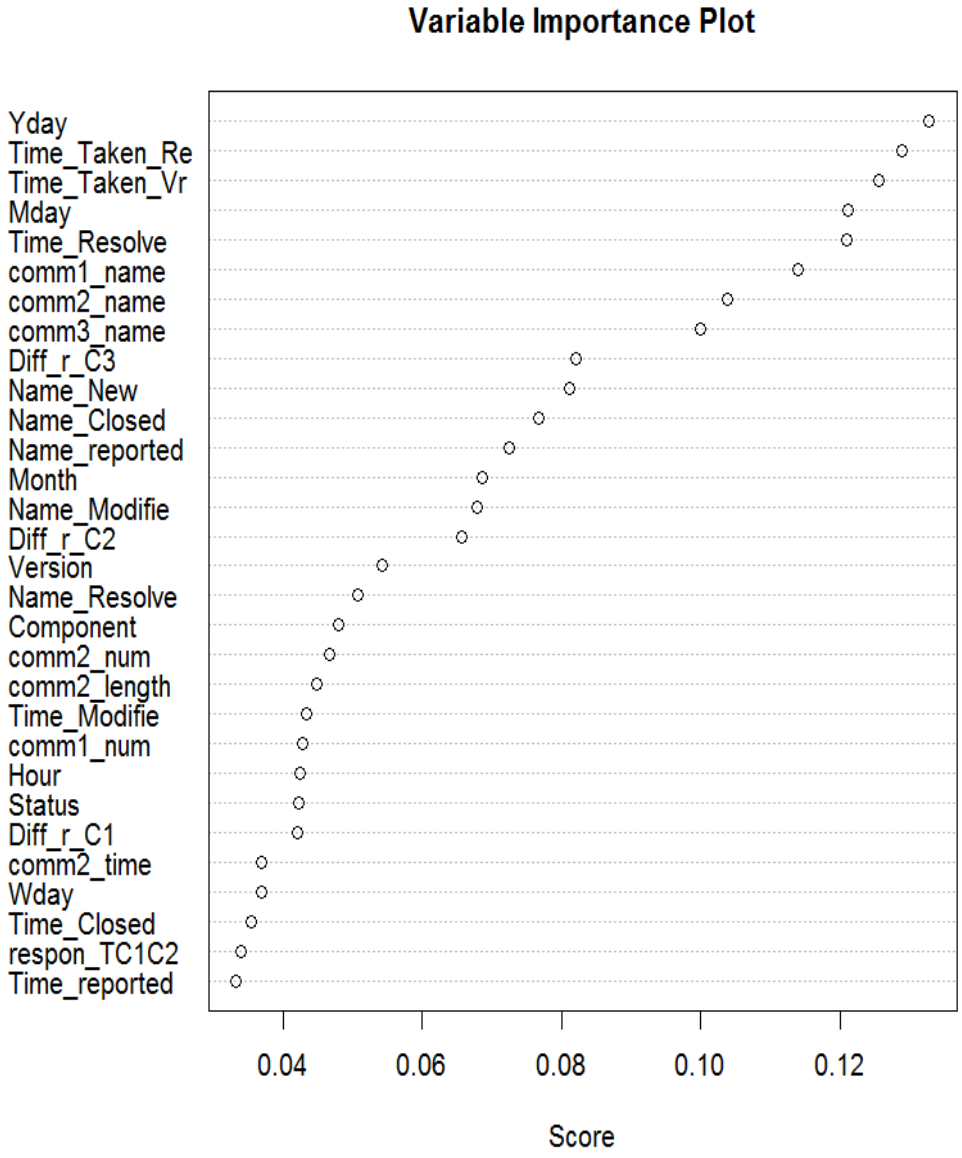

Figure 17: Important variable responsible for reopen in GCC Projects

Day of year, time taken to resolve, name of person who resolved, name of the person closed, verified, component name and number were important factors responsible for reopen.Above is variable importance graph generated by AdaBoost algorithm using Rattle.The Y axis shows the name of variable, the higher the value of y is for the variable more important it is. The X axis shows the reduction error rate when the variable is introduced in algorithm. AdaBoost reweights its instances for incorrectly classified instances so it is biased towards in correctly classified examples.

5.3 Analysis

Algorithm	Project	Instances	Correctly classified	Precision	Recall	F-Measure
ADtree	Eclipse	6568	4255	0.809	0.661	0.728
BayesNet	Eclipse	6568	4960	0.439	0.755	0.555
NaiveBayes	Netbeans	4392	4091	0.197	0.915	0.324
C4.5	Netbeans	4392	2987	0.618	0.680	0.648
NaiveBayes	Office	6082	4975	0.508	0.818	0.626
C4.5	Office	6082	4368	0.886	0.718	0.793
BayesNet	Apache	1949	1405	0.293	0.721	0.471
ADtree	Apache	1949	1294	0.560	0.664	0.608
NaiveBayes	Redhat	1915	1544	0.140	0.806	0.239
ADtree	Redhat	1915	970	0.507	0.525	0.525
BayesNet	Mozilla	5051	3560	0.325	0.705	0.445
BayesNet	W3C	537	421	0.277	0.735	0.402
C4.5	W3C	537	390	0.534	0.681	0.598
BayesNet	Gccgnu	76	76	0.264	1.000	0.418
ADtree	Gccgnu	76	50	0.735	0.658	0.694

Table 25:Summary of best algorithms in predicting reopen of bug by F-measure and recall

In our analysis C4.5 decision tree and alternating decision tree gave good results as prediction of reopen is not independent but depended on variables. NaiveBayes, which considers probabilities of independent event, gave lowest accuracy in prediction.Using top performing algorithm we achieved decent amount of Precision and Recall for reopened bugs. Precision ranged from 0.507 to 0.886 and Recall ranged from 0.525 to 0.718.ADtree and C4.5 showed high accuracy in predicting reopen of bug; both of them had highest F-measure.Reopened was most important class; its recall was most important.In our prediction BayesNetand NaiveBayes showed highest recall of reopened class. If we want to achieve high recall for reopened bug in C4.5 and ADtree, it can be done by increasing cost sensitivity of reopened class.

Category	Factors	Reason
Work Habits	Weekday, Month day, Year Day, Hour	Reopened percentage increased when bug was resolved, verified, or closed in last phase of week, month, year, and day.
Software Parts	Component, Products	Some components and product are tending to show larger rate of reopen. Variation in reopen rate in most projects was around 15%.
Difficulty in understanding Bug (Zimmermann T, 2010).	Comment name, Number of Comments	If the root cause is not properly understood, and the more comments that are made, while some developers making comment helps in understanding root cause thus reduces chances of reopen.
Amount of time taken. (Shihab E. , Ihara, Kamei, & Ibrahim, 2010)	Time taken resolve,Time taken verify	C4.5 calculates info gain of time taken to resolve at certain amount of time based on info gain it spits the decision into more than and less than of amount taken to resolve we have considered this decision as criteria for less and more time which is different for different projects. We have considered time less than If time taken to verify, fix, close is less the bug is easy to fix, and properly understood lesser chances of reopen.
Report description(Guo, Zimmermann, Nagappan, & Murphy, 2010).	Report Length	Less information in bug report was causes higher rate of reopen.
Reputation of committers (Jongyindee, Ohira, Ihara, & Matsumoto, 2011)	Name of person resolved, verified, closed.	Some of the committers are less proficient in performing task hence larger percentage reopen when they resolve, close, or verify.

Table 26: Category of causes responsible for reopen of bug

Based on previous research on bug reopen study and most important variable graphs we have categorizedbug reopen causes in 6 categories. Table 26 shows the 6 categories which are responsible for bug reopen.

Reason	Eclipse	Office	Apache	Net beans	Red hat	Mozilla	W3C	GCC	Total
Committer reputation	yes	yes	yes	yes	yes	yes	yes	yes	8
Amount of time taken	yes	yes	yes	yes	yes	no	no	yes	6
Software Parts	yes	yes	yes	yes	yes	yes	yes	yes	8
Bug under-standing	yes	yes	yes	yes	yes	yes	yes	yes	8
Bug des-cription	no	yes	no	yes	no	yes	yes	yes	5
Work Habits	no	no	yes	no	no	no	no	yes	2

Table 27: Frequency of Category for Projects

Using most important variable graph we determine whether the category was responsible for bug reopen for each project.In our observation, reputation of committers, software parts and not understanding of root cause categories had highest frequency across all projects. For Eclipse project, reputation of committers (Jongyindee, Ohira, Ihara, & Matsumoto, 2011)was important cause of bug reopen.Our observationswere consistent with this.Comment text and resolve time were variable responsible for bug reopen of Eclipse project (Shihab E. , Ihara, Kamei, & Ibrahim, 2010).Our observations were also consistent with this. If bug is not properly understood, chances of bug being reopened are

high (Guo, Zimmermann, Nagappan, & Murphy, 2010).Not properly understanding the bug was cause of reopen for all 8 projects,thus our observations were consistent with this result.

6. THREATS TO VALIDITY

6.1 Threats to Construct Validity

Construct validity to refers to degree at which operationalization of the measures in study actually refers the constructs in the real world (Shull, Singer, & Sjoberg, 2007). We have used the name of products and component as factor for reopen but we did not take in consideration the way the component are constructed, their problem domain, there code metrics.We have used reputation of fixer, verifier, and closer as variable but we have not measured their experience, background, expertise and tried to relate to reopen of bugs. Similarly, with people who make helpful comments in reducing reopen rate we have not measured there experience, background, expertise and tried it to relate to reopen of bugs.

6.2 Threats to Internal Validity

Internal validity threats affect the confidence that the identified factors actually caused the bug report to be reopened (Shull, Singer, & Sjoberg, 2007). Unknown factors can influence the results thus putting limitation on internal validity. We did not add data on version control repositories to find number of files changed. The quality of bug report was not analyzed. We do not know the code metrics of project and experience of reporter and fixer. We do not know size of and distribution of organization. Furthermore, there is risk of overfitting due to the large number of factors used, which affects the prediction capability of the models. Also, as the results were obtained at one point in time, they may change as new bugs are reported and additional bugs are reopened in the future. On the

other hand, the consistency of the findings to previously reported results provides some confidence in their validity.

6.3 Threats to External Validity

Threats to external validity concern the generality of the results (Shull, Singer, & Sjoberg, 2007). The data we acquired was just restricted to bug information collected on Bugzilla systems, thus may be affected by the way in which information is reported which could be different if data were acquired from other bug tracking systems. The data was limited to large, open source systems. Though we did not have data on commercial projects, the variety of systems studied gives some promise that similar results may be obtained in commercial systems.

7. CONCLUSIONS

In our research, we were able to automate data collecting techniques for mining bug repositories. We collected data from 8 projects from different software categories. Data was cleaned and designed in three different categories: report, activity, and comment. Classification algorithms were studied and then applied to predict the probability of reopened bugs. In all of the projects a decent amount of precision and recall was achieved. The precision for reopen bugs was from 40% to 90% while range of recall was from 40% to 100%. ADtree, C 4.5 achieved the best F-measure for prediction of reopens while NaiveBayes and BayesNet achieved the best recall of reopened bugs. We found the most important factors responsible for a reopen were component, name of person who fixed name of the person who verified the name of the person who closed the bug, the number, resolving time, verifying time, size, and name of person who made the comment. We developed a data mining methods that was different from other software repository miners, for we created a web crawler to get bug information from the web instead of a more traditional way of mining software repository through files. We were able extract information from around 1 million web pages. The advantage of this method was that we got the latest updated information of projects and that we had access to all the projects open to public. We introduced the name and time the person verified and closed and dimension of the last 3 comments. We had a higher precision and recall then the previous research, which was verified by application to different category of projects.

Using reopen analysis of bugs, developers can share data with bug reporters which shows likelihood of reopening a bug report if bug is from a certain component.

Components with high reopen rate can be studied to find their coding metrics. Committers which show high rate of reopen can be retrained to reduce their reopen rate. Assigning of the higher priority bugs can be restricted to committers with higher reputation. Guidance of developers whose comments help in reducing reopen rate can be used for higher priority bugs. By predicting whether bug will be reopened beforehand, more resources can be allocated before documenting it as fixed, thus percent of reopen will go down, increasing reliability of software.

8. FUTURE WORK

The amount of resources we had access to had been limited to bug repository. But the code metrics: lines of codes, global variables, cyclomatic complexity, read coupling, write coupling, address coupling, fan-in, fan-out, weighted methods per class, depth of inheritance, class coupling, and number of subclasses, all of which are important factors in finding bugs, can be incorporatedto enhance research regarding reopen bugs. We plan to understand the contents of reports and comments, weight them according to keywords present that can predict reopen. We plan to create a developer profile with their fixing experience with a type of modules and work habits. Adding the mentioned factors will enhance our knowledge of factors responsible for reopen and make out precision recall more accurate.

REFERENCES

Bayes. (1763). An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society*, 370-418.

Boehm, B., & Basili, V. (2001). Software Defect Reduction Top 10 List. *Computer Volume 34 Issue 1*, 135-137.

Drauschke, M. (2008). *Feature Subset Selection with Adaboost and ADTboost.* Bonn: Department of Photogrammetry,University of Bonn.

Fayyad, U. (1996). From Data Mining to Knowledge Discovery in Databases. *Scientific and Statistical Database Management*, 2 - 11.

Freund, Y., & Mason, L. (1999). The alternating decision tree learning algorithm. *International Conference on Machine Learning*, (pp. 124-133).

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. *In Machine Learning*, 148-156.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning. *Journal of Computer and System Sciences.*, 119–139.

Freund, Y., & Schapire, R. E. (1999). A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence.*, 771-780.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 131-163.

Gu, Z., Barr, E., Hamilton, D., & Su, Z. (2010). Has the bug really been fixed? *International Conference on Software Engineering*, 55 - 64.

Guo, P., Zimmermann, T., Nagappan, N., & Murphy, B. (2010). Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. *International conference on software engineering*, 495–504.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. (2009). The WEKA Data Mining Software.

Hassan, A. (2008). The road ahead for Mining Software Repositories. *Frontiers of Software Maintenance, 2008. FoSM 2008.*, 48 - 57.

Jongyindee, A., Ohira, M., Ihara, A., & Matsumoto, K.-i. (2011). Good or Bad Committers? A Case Study of Committers' Cautiousness and the Consequences on the Bug Fixing Process in the Eclipse Project. *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*, 116 - 125.

Kohavi, R., & Provost, F. (1998). Classifier performance evaluation. *Machine Learning*, 271-274.

Mitchel, T. (1997). *Machine Learning.* McGraw.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine. Learning.* , 81-106.

Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross Validation. *Encyclopedia of Database Systems (EDBS)*, 532-538.

Shihab, E., Ihara, A., Kamei, Y. I., Ohira, M., Adams, B., Hassan, A., et al. (2012). Studying re-opened bugs in open source software. *Empirical Software Engineering October 2013, Volume 18, Issue 5*, 1005-1042.

Shihab, E., Ihara, A., Kamei, Y., & Ibrahim, W. (2010). Predicting Re-opened Bugs: A Case Study on the Eclipse. *Reverse Engineering work shop*, 249-258.

Shull, F., Singer, J., & Sjoberg, D. (2007). Guide to Advanced Empirical Software Engineering. *Springer*.

Sliwerski, J., Zimmermann, T., & Zeller, A. (2005). *When do changes induce fixes?* 1–5: Proceedings of the 2005 international workshop on Mining software repositories, .

Vlasceanu, I. V., & Bac, C. (2008). *A study concerning the bug tracking applications.* TELECOM & Management SudParis.

Williams, G. (2009). *Rattle: A Data Mining GUI for R.* The R Journal.

Zimmermann, T., Nagappan, N., & Zeller, A. (2008). Predicting Bugs from History – Software Evolution. *Springer*, 69-88.