Online SVM based Optimized Bug Report Triaging using Feature Extraction

Ms. Neetika Sharma
M. Tech. Scholar
Department of Computer Science and Engineering
Kautilya Institute of Technology and Engineering, Jaipur (Rajasthan)

Dr. Vijay Kumar
Professor and Head
Department of Computer Science and Engineering
Kautilya Institute of Technology and Engineering, Jaipur (Rajasthan)

Abstract

Triage is medical term referring to the process of prioritizing patients based on the severity of their condition so as to maximize benefit (help as many as possible) when resources are limited. Bug Report triaging is a process where tracker issues are screened and prioritized. Triage should help ensure that all reported issues are properly managed - bugs as well as improvements and feature requests. The large number of new bug reports received in bug repositories of software systems makes their management a challenging task. Handling these reports manually is time consuming, and often results in delaying the resolution of important bugs. The most critical issue related with bug reports is that their number is vast and most of these are duplicates of some previously sent bug report. The solution to this problem requires that bug reports are to be categorized in groups where each group consist of all the bug reports which belongs to the same bug, and the number of groups is equal to the number of unique bugs addressed so far. Bug report corresponding to some new bug is to be placed in a separate group followed by its duplicates, if any. Classifying weather a bug report that arrived through a user, written in a natural language, is a duplicate or unique report is a time consuming task, especially when the number of bug reports that are received is large. Thus, this process needs to be automated. Bug reports have textual, contextual and categorical features and these features needs to be extracted for checking of duplicates and non duplicates. Moreover, in the group of reports, a particular report can be specified as master and all the reports that corresponds to the same bug are to be linked to it. Thus, duplicates need not be discarded so as to provide later, a complete description of the bug. In this paper, a much more extended set of textual features is considered for bug report duplicacy checking. Support Vector Machine classifier is used for classification of the incoming bug report as duplicate of non-duplicates. The simulation of the prescribed model is done using R Statistical Package. A sample of bug reports from Mozilla repository is considered, Results of the simulation model establishes the fact that Proposed classifier has higher efficiency as compared to existing technique BM25F which employs 25 feature sets.

Keywords: Bug Report Triaging, Feature Extraction, Machine Learning Algorithms, Bayesian Classifier

I. INTRODUCTION

An open source project typically maintains an open bug repository so that bug reports from all over the world can be gathered. When a new bug report is submitted to the repository, a person, called a triager, examines whether it is a duplicate of an existing bug report. If it is, the triager marks it as duplicate and the bug report is removed from consideration for further work. In the literature, there are approaches exploiting only natural language information to detect duplicate bug reports. These software repositories provide abundance of valuable information about open source projects [1]. With the increase in the size of the data maintained by the repositories, automated extraction of such data from individual repositories, as well as of linked information across repositories, has become a necessity. In this paper, a framework is described that uses web scraping to automatically mine repositories and link information across repositories. Mining software repositories is an important activity when analyzing large scale projects. Mining information across multiple data sources is one of the challenges [2]. It is observed that relevant information from one repository can complement the mining activity on another repository. For example, the Bugzilla bug tracker for Mozilla and Red Hat projects contain custom "keywords", textual tags, that help identify specific categories of bugs in the database. The keyword security relates to a security bug. This could have been used to identify the security bugs in projects like Fedora, Firefox etc. But the usage of the keyword was not consistent across bug reports. The Common Vulnerability Exposure (CVE) [3] site maintains information about publicly known vulnerabilities. The vulnerabilities tagged by CVE are contained in the National Vulnerability Database (NVD), a U.S. government repository devised to manage vulnerability data. The NVD database lists vulnerabilities specific to different types of products including Fedora (Red Hat), Firefox (Mozilla) etc. The external resource section of each vulnerability listed in the NVD has a mapping or link to the bugs in their respective bug-tracking system. This implies that information from the NVD can be utilised in mining security bugs in projects like Firefox, Fedora etc. For projects, like Ubuntu, that deploy the Launchpad bug tracker, the search engine allows to search for bugs with CVE tags [4]. These CVE tags in turn can be used to collect linked vulnerability characteristics in terms of the nature of exploits,

impacts, and their metric values present in the NVD. Extracting such information would give additional context to information available through individual repositories. Manually extracting such information is tedious.

II. PROBLEM STATEMENT, MOTIVATION AND RESEARCH APPROACH

A. Problem Statement

With the increase in size and functionality of the software, the possibility of finding out the potential bugs increases. This in-turn, increases the pressure on the triager who needs to manage the overall bug repository to check for duplicates and non duplicates. The research till date involves the feature extraction process of 27 features and 27 from bigrams, which results in a total of 54 features for comparison between two reports to check whether the incoming report belongs to positive or negative examples. This paper takes this feature extraction process a step more advance and consider the similarity measurement based on 147 features and 147 from bigrams resulting in a total of 294 features for textual similarity measurement. It further augments the textual feature set with 5 categorical features; namely product, component, type, version and priority and one most important feature called the context which corresponds to the non functional requirement of the software. Thus, this work extends the feature set to 300 features including textual, categorical and contextual features for duplicate reports checking.

B. Motivation

Managing the incoming deluge of new bug reports received in bug repository of a large open source project is a challenging task. Handling these reports manually by developers, consume time and resources which results in delaying the resolution of crucial (important) bugs which need to be identified and resolved earlier to prevent major losses in a software project. In this paper, we present a machine learning approach to develop a bug priority recommender which automatically assigns an appropriate priority level to newly arrived bugs, so that they are resolved in order of importance and an important bug is not left untreated for a long time. Our approach is based on the classification technique, for which we use Support Vector Machines. Experimental evaluation of our recommender using precision and recall measures reveals the feasibility of our approach for automatic bug priority assignment.

C. Research Approach

Duplicate bug report retrieval can be viewed as an application of information retrieval (IR) technique to the domain of software engineering, with the objective of improving productivity of software maintenance. In classical retrieval problem, the user gives a query expressing the information he/she is looking for. The IR system would then return a list of documents relevant to the query. For duplicate report retrieval problem, the triager receives a new report and inputs it to the duplicate report retrieval system as a query. The system then returns a list of potential duplicate reports. The list should be sorted in a descending order of relevance to the queried bug report. Our approach adopts recent development on discriminative models for information retrieval to retrieve duplicate bug reports. Adapted from [5], in this paper, the duplicate bug report retrieval problem is considered as a binary classification problem, that is, given a new report, the retrieval process is to classify all existing reports into two classes: duplicate and non-duplicate.

This paper is organized as follows. Section 1 presents an overview of the subject matter and gives the problem statement and the approach for the research. Section 2 provides the detailed overview of of problem Definition. Section 3 presents the proposed techniques and proposed model. Section 4 presents the results and interpretations for fault detection using Support Vector Machines. Section 5 concludes the Paper.

III. PROPOSED WORK

A. Automated Checking of Duplicate Reports

In this work, a new approach is introduced for improving the accuracy of detecting duplicate bug reports of a software system. Duplicate bug report retrieval can be viewed as an application of information retrieval (IR) technique to the domain of software maintenance, with the intent of improving productivity of software maintenance. In a typical IR system, the user inputs a query and the IR system respond with the list of documents relevant with the given query. In duplicate bug report detection system, the incoming bug report is subjected to detection system which then respond with a list of potential duplicate bug reports related to the input report. The list should be sorted in a descending order of relevance to the queried bug report.

In this approach, the domain knowledge is exploited which relates to the software system under consideration to improve bug-report de-duplication. Essentially, rather than naively and exclusively applying information-retrieval (IR) tools, the proposed model takes the advantage of knowledge of the software process and product through topic modeling. The approach is based on the primary hypothesis that bug reports are likely to refer to software qualities, i.e., non-functional requirements (possibly being desired but not met), or software functionalities (linked to architectural components responsible for implementing them). A few software dictionaries and word lists are used, exploited by prior research, to extract the context implicit in each bug report. The contextual word lists are compared to the bug reports and the comparison results are analyzed as new features for the bug reports, in addition to the primitive textual and categorical features of the bug reports, such as description, component, type, priority, etc.

proposed in Sun et al.'s work. Then, this extended set of bug-report features is used to compare bug reports and detect duplicates. Simulations are done using R simulation and the results demonstrate that the use of contextual features improves bug de-duplication performance. The proposed approach is evaluated on a large bug-report data-set from the Mozilla project, which is an open source web browser most commonly used all round the globe. About 37000 Mozilla bug reports are analyzed in simulation. In this research, contextual word lists are taken to study the effect of various software engineering contexts on the accuracy of duplicate bug-report detection. These word lists include: Browser related topics and software Non-Functional Requirements words. These words are extracted applying Latent Dirichlet Allocation (LDA) method, Mozilla topic words extracted applying Labeled-LDA method, and random English dictionary words (as a control). It turns out that the proposed method results in 16.07% relative improvement in accuracy and an 87.59% relative improvement in Kappa measure (over the baseline). This work makes the following contributions.

1) It proposes the use of domain knowledge about the software process and products to improve bug de-duplication performance.
2) The proposed method along with the textual feature model improves the accuracy of duplicate bug-report detection.
3) The effect of considering different contextual features on the accuracy of bug-report de-duplication is systematically investigated.

B. Proposed Model

Figure 3.1 shows the framework of the proposed technique.

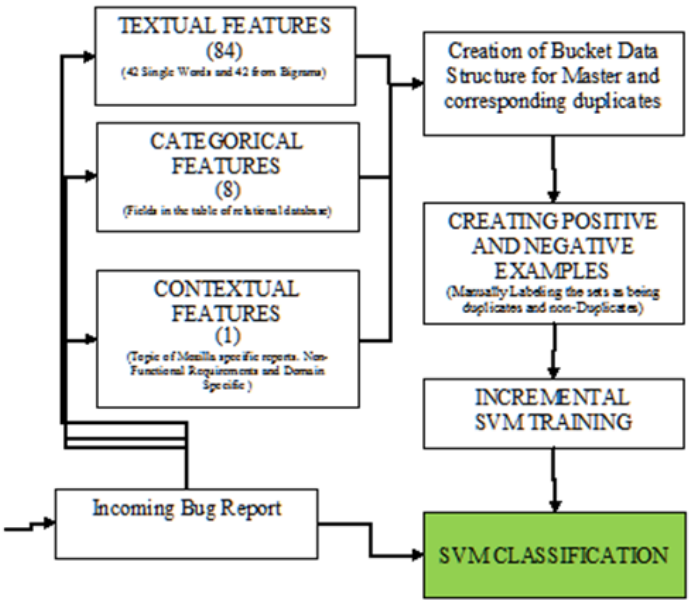

Fig. 3.1: Proposed Framework

A bug tracking system typically consists of the following fields as shown in table 3.1.

Table 3.1:
Typical Fields of A Bug Repository (Relational DB Table)

Field_name	*Description*
Bug Id	*Primary key for the table*
Title	*Title of the report*
Description	*Description of malfunction(s)*
Summery	*Summary of what had happened (requirements not met)*
Status	*Status of the Bug report (sorted or pending)*
Component	*Which component among several types of products is under consideration*
Priority	*Critical, High, Medium, Low*
Type	*Defect, or Undesired*
Version	*Version number of component*
Open Date	*Date at which report is submitted*
Close Date	*Date at which report is closed*
Merge Id	*Id of other reports found duplicate of this particular bug report.*

Preprocessing, in its first phase, consists of the following steps:

1) The first step of preprocessing is tokenization. It is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens.

2) Removing the stop words from the textual features (title, description and summery) of bug reports using a list of English stop words. These stop words includes I, am, is, are, the, etc.
3) Stemming is the process to reduce words to their ground forms. Thus, the following transformation takes place as a result of stemming operation:

saving→save, deleted→delete, documents→document etc.

C. Textual and Categorical Comparison

After preprocessing, the pair wise similarity between every two bug reports based on their primitive features (title, description, summery, component, type, priority, and version) is measured. For three of these, viz. title, description and summery, the textual feature extraction is performed and analyzed and for all other fields, categorical features are extracted to extend the feature set. The textual features for two bug reports B1 and B2 are computed using the following scheme:

- $\text{Feature}_1(B_1(\text{title}), B_2(\text{title})) = \text{BM25F}(B_1(\text{title}), B_2(\text{title}))$
- $\text{Feature}_2(B_1(\text{title}), B_2(\text{Description})) = \text{BM25F}(B_1(\text{title}), B_2(\text{Description}))$
- $\text{Feature}_3(B_1(\text{title}), B_2(\text{Summery})) = \text{BM25F}(B_1(\text{title}), B_2(\text{Summery}))$
- $\text{Feature}_4(B_1(\text{Description}), B_2(\text{Title})) = \text{BM25F}(B_1(\text{Description}), B_2(\text{Title}))$
- $\text{Feature}_5(B_1(\text{Description}), B_2(\text{Description})) = \text{BM25F}(B_1(\text{Description}), B_2(\text{Description}))$
- $\text{Feature}_6(B_1(\text{Description}), B_2(\text{Summery})) = \text{BM25F}(B_1(\text{Description}), B_2(\text{Summery}))$
- $\text{Feature}_7(B_1(\text{Summery}), B_2(\text{Title})) = \text{BM25F}(B_1(\text{Summery}), B_2(\text{Title}))$
- $\text{Feature}_7(B_1(\text{Summery}), B_2(\text{Description})) = \text{BM25F}(B_1(\text{Summery}), B_2(\text{Description}))$
- $\text{Feature}_8(B_1(\text{Summery}), B_2(\text{Summery})) = \text{BM25F}(B_1(\text{Summery}), B_2(\text{Summery}))$
- $\text{Feature}_9(B_1(\text{Description} + \text{Summery}), B_2(\text{Description})) = \text{BM25F}(B_1(\text{Description} + \text{Summery}), B_2(\text{Description}))$
- $\text{Feature}_{10}(B_1(\text{Description} + \text{Summery}), B_2(\text{Summery})) = \text{BM25F}(B_1(\text{Description} + \text{Summery}), B_2(\text{Summery}))$
- $\text{Feature}_{11}(B_1(\text{Description} + \text{Summery}), B_2(\text{Title})) = \text{BM25F}(B_1(\text{Description} + \text{Summery}), B_2(\text{title}))$
- $\text{Feature}_{12}(B_1(\text{Description}), B_2(\text{Description} + \text{Summery})) = \text{BM25F}(B_1(\text{Description}), B_2(\text{Description} + \text{Summery}))$
- $\text{Feature}_{13}(B_1(\text{Summery}), B_2(\text{Description} + \text{Summery})) = \text{BM25F}(B_1(\text{Summery}), B_2(\text{Description} + \text{Summery}))$

However, there are much more feature than specified above and can be counted out using all possible combinations on the format of the bug report. Total number of such features can be computed as follows:

Considering the bug reports as comprising of the format involving TITLE, DESCRIPTION and SUMMARY of the bug reports, the total number of ways of selecting one or more items from the set of three items is:

$$\binom{3}{1} + \binom{3}{2} + \binom{3}{3} = 3 + 3 + 1 = 7$$

Possible set of mapping from a set of seven elements to another set consisting of seven elements is: 7*7 = 49.

Also, there are three distinct type of corpuses, namely, summary, description and summary + description. Thus the total number of different types of mappings is 49*3 = 147. Considering bigrams, this gives a total of 147+147 = 294 features textual mapping.

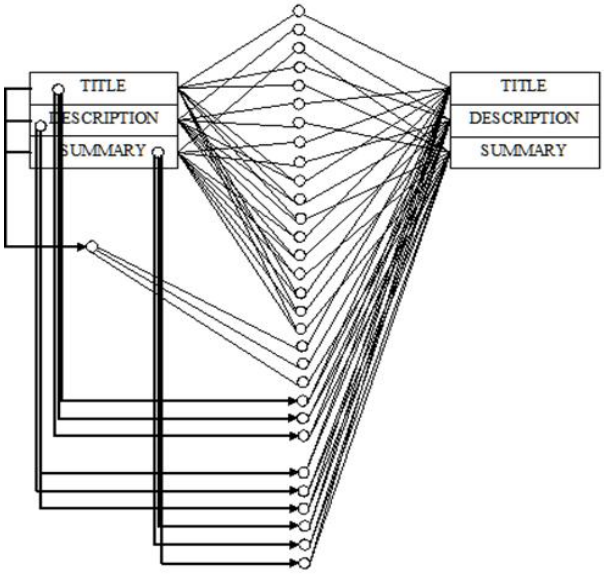

Fig. 3.2: Illustration of feature extraction

Hence, the proposed feature set consists of 294+5+1 = 300 textual features including 5 categorical and 1 contextual feature. It is worth to state here that this textual mapping formulates the largest number of Feature Set of all the textual reports that are analyzed in any of the approach.

A significant portion of this feature set is illustrated in the figure 3.2. Here, each of the circle at the middle of the figure represents one feature set.

The categorical features can be extracted in the following way:

- $\text{Feature}_1(B_1(\text{Product}), B_2(\text{Product})) = \begin{cases} 1; \text{if } B_1.\text{product} = B_2.\text{product} \\ 0; \text{otherwise} \end{cases}$
- $\text{Feature}_2(B_1(\text{Component}), B_2(\text{Component})) = \begin{cases} 1; \text{if } B_1.\text{component} = B_2.\text{component} \\ 0; \text{otherwise} \end{cases}$
- $\text{Feature}_3(B_1(\text{Type}), B_2(\text{Type})) = \begin{cases} 1; \text{if } B_1.\text{type} = B_2.\text{type} \\ 0; \text{otherwise} \end{cases}$
- $\text{Feature}_4(B_1(\text{Priority}), B_2(\text{Priority})) = \frac{1}{1+|B_1(\text{Priority}) - B_2(\text{Priority})|}$
- $\text{Feature}_5(B_1(\text{Version}), B_2(\text{Version})) = \frac{1}{1+|B_1(\text{version}) - B_2(\text{version})|}$

Thus, above specified features from 1 to 5 accounts to categorical similarity.

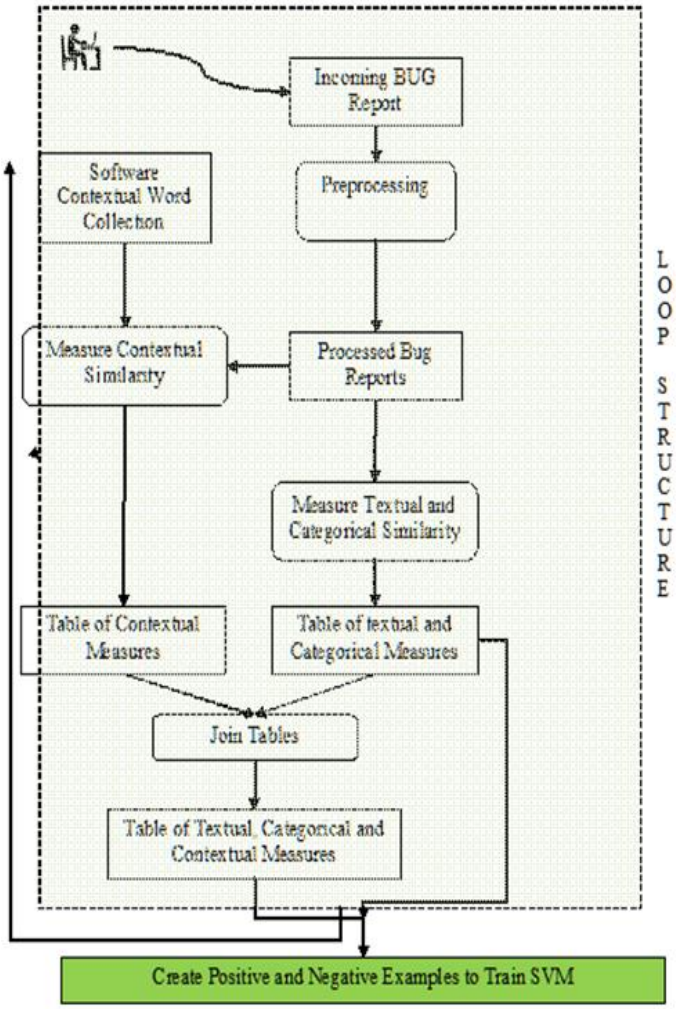

Fig. 3.3: Architecture of Proposed Model, using Contextual Modeling

This feature model, consisting of 294+5+1 features is applicable to most of bug repositories. The architecture of the proposed model can be depicted in figure 3.3.

After analysis and checking for duplicates, all the reports in the repository are organized into a bucket like structure as illustrated in figure 3.4

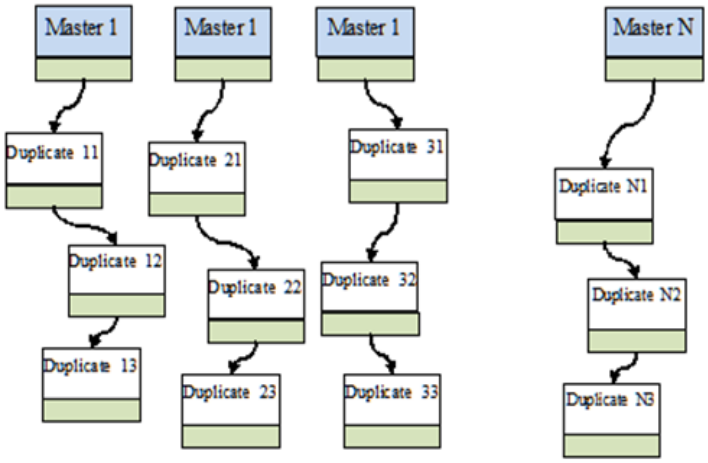

Fig. 3.4: Data Structure for Proposed System

The data structure as described in the figure 3.4 can be considered as a bucket structure with each row representing a bucket. Each bucket consists of a master report and one or more duplicate reports. These duplicate reports are not discarded but are kept attached with the master report so as to provide a complete description of the bug as described by the reports. New reports are classified by the triager as duplicate or non duplicates and are added to the repository. Duplicate reports are transferred to the buckets corresponding to the master reports whereas new buckets are created corresponding to non duplicate reports.

D. Algorithm for Populating Data Structure

The proposed algorithm works as follows:

1) Input bug report to the system.
2) If there is no bug report in the system, mark it as unique report, then go-to 6, otherwise; for all other reports labeled as master reports in the system, do;
3) Obtain the textual, contextual and categorical similarity measure with the master reports.
4) If the similarity score is above threshold, then; for any of the report for which the similarity score approaches the maximum score, label the report as the duplicate of the specified report
5) If the similarity score lies below the threshold, mark the report as a new master report.
6) End

E. Algorithm for SVM Training

The proposed algorithm works as follows:

1) Create a set of bug reports one from each repository in such a way that the similarity score between any pair or reports is smaller than the specified threshold. Label this repository as the set of negative examples.
2) Create a set of bug reports in such a way that the similarity score between each one of them is greater than the specified threshold. This can be formed using a single master reports and all its duplicates. However, bug reports from other repositories are to be prioritized having nearly equal score as that of the master and duplicates.
3) Name this set as training set and train an SVM based classifier on this set.
4) Test the accuracy of SVM on any incoming bug report.

Positive and negative examples are created using this bucket structure for training of the support vector machine, which is a linear classifier. Positive examples can be created using a master bug report and its duplicates, or two duplicates from the same bucket. Negative examples can be created using reports from distinct buckets. Thus the number of negative examples fairly exceeds the number of positive examples. Therefore, the negative examples should be chosen suitably to accommodate nearly all the distinct pairs.

In section 4, results corresponding to Mozilla Firefox bug repository are considered. Topic modeling is done using SVM to model the topics which are non-functional requirements of the software. Textual and categorical features are analyzed along with the semantic features to extend the feature set and to perform triaging more accurately.

IV. ANALYSIS OF PROPOSED WORK

A. Mozilla Firefox Bug Reports (bug.xls, included in CD-ROM)

The simulation of the proposed work is operated on the bug repository consisting of 14343 bug reports of Mozilla Firefox. A sample of these bug reports is shown in table 4.1 below. These reports corresponds to the set of non-duplicate bug reports.

Table 4.1:
Records From Mozilla Firefox Bug Repoisitory (Set Of The Records Of Non-Duplicates)

"... remixes" shows the username for each remix, which can show the same user 10 times
"/usr/bin/sh: compare-locales: command not found" in Windows B2G logs (highlighted as a failure, but doesn't cause orange)
"/websites/theme/" repository using different theme for its web interface than other repositories
"Active question" screen should have a sync button
"Advanced Search" options missing on Google search in Firefox for Android (vs Chrome)
"All Levels" setting for ToC does not work
"All Questions" view is very slow
"AttributeError: 'module' object has no attribute 'PROTOCOL_SSLv3" when setting up MozReview test environment
"Becoming A Mozilla Committer" should clarify the bug filing steps
"By N contributors" feature includes users who made only empty revisions

The word corpus from the text in all the bug reports can be generated from the simulation model and is depicted in the figure 4.1 below. These bug reports forms the set of all the non-duplicate bug reports as these are associated with almost all the bugs identified by different users across various versions and under various accessibility conditions.

The word cloud gives a pictorial representation of the keywords appearing in the word corpus after the stopword removal and stemming operation. The text size of a particular keyword corresponds to the frequency of the word in the corpus and thus is a measure of the term frequency in the whole corpus. In other words, the font size of a word is indicative of how important a word is in the entire corpus.

Fig. 4.1: Word Corpus from the set of Non-Duplicate bug report (Negative Examples)

The sample from the set of bug reports which corresponds to the duplicates is shown in table 4.2. These reports are all corresponds to the 404 error which is corresponding to "Page Not Found" http error.

Table 4.2:
Records From Mozilla Firefox Bug Repoisitory (Set Of The Records Of Duplicates)

"404 File not found" error on page http://education.mozilla-community.org/modules/marketpulse_firefox_os/introduction/
"Thank you for your try push" emails only link to ftp.m.o/..../firefox/... and so 404 for B2G-only pushes
/az/thunderbird/ gives 404 error
404 Errors in Volunteer introductory email for testing
404 Not Found
404 Not Found when visiting a non-existent locale's page
404 Page not found at http://www.mozilla.org/en-US/about/mission
404 error
404 error if you click log out after cookie has expired
404 error on page

The wordcloud corresponding to the set of duplicate bug reports is depicted in the figure 4.2 shown below.

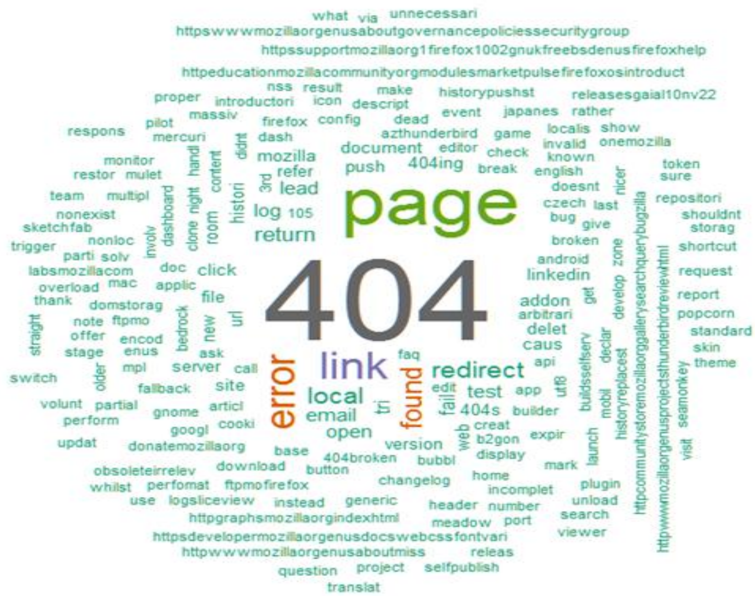

Fig. 4.2: Word Corpus from the set of Non-Duplicate bug report (Positive Examples)

It is clearly evident from figure 4.2 and the definition of wordcloud that for most of the duplicate bug reports, the keyword 404 appears in the text and consequently, most of the bug reports corresponds to the 404 error (http: Page Not Found Error).

The complete set of all the bug reports corresponding to duplicates and non-duplicates can be found in the files dup.csv and non_dup.csv enclosed in the CD-ROM enclosed. A training set can be prepared from the above two sets of reports as shown in table 4.3 below.

Table 4.3:
Training Data Set. Label 1 Corresponds To Duplicates And Label -1 Corresponds To Non-Duplicates

BUG REPORT	*LABEL*
"Thank you for your try push" emails only link to ftp.m.o/..../firefox/... and so 404 for B2G-only pushes	*1*
/az/thunderbird/ gives 404 error	*1*
404 Errors in Volunteer introductory email for testing	*1*
404 Not Found	*1*
404 Not Found when visiting a non-existent locale's page	*1*
404 Page not found at http://www.mozilla.org/en-US/about/mission	*1*
404 error	*1*
404 error if you click log out after cookie has expired	*1*
404 error on page	*1*
404 in mozilla NSS FAQ	*1*
404 on "Mark this as solved" in question response emails	*1*
404 page on donate.mozilla.org does not declare UTF-8 encoding properly	*1*
404 page shouldn't ask for filing bug reports if Referer is known	*1*
404 trying to clone releases/gaia-l10n/v2_2 repository from Mercurial	*1*
All URLs in description for Home Dash add-on are 404	*1*
Bubble Meadow launching to 404 error	*1*
Cannot create new team site, 404	*1*
Click on the last page number returns 404 Page not found	*1*
DOM/Storage (Local Storage) link is a 404/broken, on staging	*1*
Deleting a room whilst the room is open causes unnecessary 404s (105 invalid token) on the server	*1*
Download link for Japanese Seamonkey for Mac is 404	*1*
History returns 404 on localized pages using en_US fallback content	*1*
In the 404 page, perform a search based on what's in the URL	*1*
Link 404 from https://developer.mozilla.org/en-US/docs/Web/CSS/font-variant	*1*
LinkedIn pages are 404'ing in Firefox for Android after LinkedIn does a redirect	*1*
\... remixes\" shows the username for each remix which can show the same user 10 times"	*-1*
\/usr/bin/sh: compare-locales: command not found\" in Windows B2G logs (highlighted as a failure but doesn't cause orange)"	*-1*
\/websites/theme/\" repository using different theme for its web interface than other repositories"	*-1*
\Active question\" screen should have a sync button"	*-1*
\Advanced Search\" options missing on Google search in Firefox for Android (vs Chrome)"	*-1*
\All Levels\" setting for ToC does not work"	*-1*

\All Questions\" view is very slow"	*-1*
\AttributeError: 'module' object has no attribute 'PROTOCOL_SSLv3\" when setting up MozReview test environment"	*-1*
\Becoming A Mozilla Committer\" should clarify the bug filing steps"	*-1*
\By N contributors\" feature includes users who made only empty revisions"	*-1*
\Certified\" APIs don't offer any certification and therefor should be renamed"	*-1*
\Coming soon\" is misleading on Devices page"	*-1*
\Common Pitfalls to avoid in Web Design\" - this page is missing"	*-1*
\Contribute to this page\" link in footer is misleading"	*-1*
\Corrupt\" app installed on OSX via beta (34.0b1)"	*-1*
\Discard changes\" button should not appear on the page creation form"	*-1*
\Disconnect Search\" add-on breaks ability to turn off search suggestions"	*-1*
\Draft\" articles"	*-1*
\ERROR - device/qcom/b2g_common/treeid.sh: line 42: repo: command not found\" (and same for vendorsetup.sh) on all Hamachi device image builds"	*-1*
\Empty string passed to getElementById().\" error when selecting a job"	*-1*
\Error: t is null\" warnings in console when data ingestion tasks have not completed yet"	*-1*
\Exact search\" capability and small template for user profiles"	*-1*
\Flags Requested of You\" should provide a link to the anchor where the flag was requested"	*-1*
\Getting started\" notification shows in multiple places on demos/submit"	*-1*
\HTTP/1.1 501 Not Implemented\" error loading KB article"	*-1*

B. SVM Based Classifier for Classification and Prediction Results

The simulation of the SVM based classifier is done using R package. The SVM is trained on a set of 25 positive and 25 negative examples from the training set depicted in table 4.3. The parametric specification for the same is given as follows:

Call:
svm.default(x = container@training_matrix, y = container@training_codes,
kernel = kernel, cost = cost, cross = cross, probability = TRUE, method = method)
Parameters:
SVM-Type: C-classification
SVM-Kernel: linear
Cost: 1
gamma: 0.004132231
Number of Support Vectors: 46

The SVM is then subjected to a test set of 25 records from the set of duplicates. The prediction results and the probability are shown in table 4.4.

Table 4.4:
Test Results On The Set Of 25 Bug Reports From The Set Of Duplicates

Report Id	*Classified Class*	*Probability*
1	*1*	*0.9710717*
2	*1*	*0.9467327*
3	*1*	*0.9814862*
4	*-1*	*0.7107883*
5	*1*	*0.9623039*
6	*1*	*0.9639090*
7	*1*	*0.9730007*
8	*1*	*0.9797651*
9	*1*	*0.9776163*
10	*1*	*0.9321973*
11	*1*	*0.9378497*
12	*1*	*0.9709577*
13	*1*	*0.9170495*
14	*1*	*0.9589599*
15	*1*	*0.9467327*
16	*1*	*0.9372019*

17	*1*	*0.9656583*
18	*1*	*0.9710299*
19	*1*	*0.9708282*
20	*1*	*0.9298223*
21	*-1*	*0.7910240*
22	*1*	*0.9468047*
23	*1*	*0.9546395*
24	*1*	*0.9394033*
25	*1*	*0.9931366*

The graph corresponding to the table 4.4 is depicted in fig 4.3 given below. It is evident from this figure that the trained SVM predicted successfully 23 bug reports from among 25 bug reports on which it is tested. Moreover, the prediction probability is close to 1 which further revels that the SVM predicts the bug reports with a high accuracy over a relatively small training data set.

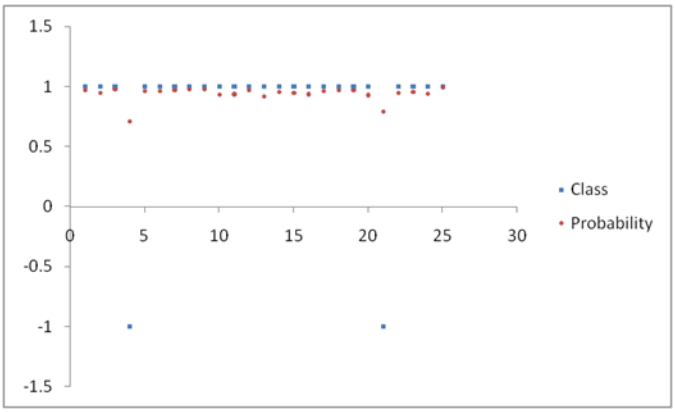

Fig. 4.3: Classification of 25 records from the set of Duplicates

The results of prediction of SVM classifier on the set of 25 records from the set of non duplicates is depicted in table 4.5.

Table 4.5:

Test Results On The Set Of 25 Bug Reports From The Set Of Duplicates

Report ID	*Class*	*Probability*
1	*-1*	*0.884907*
2	*-1*	*0.901708*
3	*-1*	*0.886295*
4	*-1*	*0.805409*
5	*-1*	*0.913438*
6	*-1*	*0.647916*
7	*-1*	*0.924346*
8	*-1*	*0.928007*
9	*-1*	*0.911923*
10	*-1*	*0.885048*
11	*-1*	*0.884907*
12	*-1*	*0.955447*
13	*-1*	*0.823318*
14	*-1*	*0.933366*
15	*-1*	*0.895953*
16	*-1*	*0.884907*
17	*-1*	*0.941175*
18	*-1*	*0.837616*

19	*-1*	*0.872933*
20	*-1*	*0.928318*
21	*-1*	*0.884907*
22	*-1*	*0.885048*
23	*-1*	*0.869309*
24	*-1*	*0.896995*
25	*-1*	*0.928909*

The graph corresponding to the table 4.5 is depicted in figure 4.4 shown below. It is evident from the figure that the proposed SVM classifies almost all the 25 records exactly and with high probability.

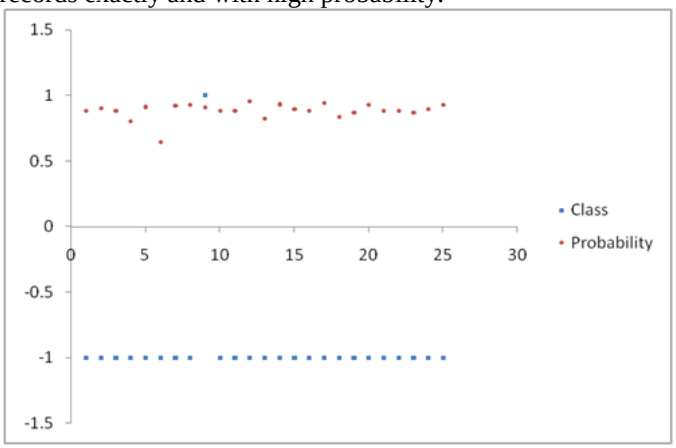

Fig. 4.4: Classification of 25 records from the set of Non-Duplicates

C. Prediction Accuracy of online SVM

Prediction Accuracy in case of online SVM can be analyzed by measuring the prediction accuracy as a function of number of records used in the training set. These are the records which are used to create the support vectors and hence contributes significantly towards prediction accuracy of SVM.

Prediction Accuracy as a function of number of records used in the training set is shown in table 4.6 and plotted in the figure 4.5 shown below.

Table 4.6:
Accuracy Measurement (Average) Considering A Test Set Of 25 Records

Number of Records in training Set	*Accuracy*	
	Duplicates	*Non Duplicates*
25	*92*	*96*
50	*95*	*96*
75	*95*	*97*
100	*95.5*	*97.5*

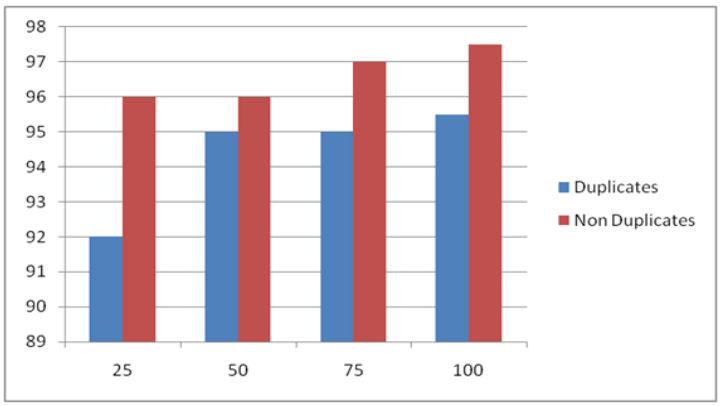

Fig. 4.5: Prediction Accuracy as a function of number of records in the training set (average values for a random set of 25 records in the test set). Horizontal axis shows the training set and the vertical axis shows the accuracy score.

Table 4.7:
Accuracy Measurement (Average) Considering A Test Set Of 25 Records

Number of Records in training Set	*Accuracy*	
	Duplicates	*Non Duplicates*
50	*90*	*85*
100	*91*	*88*
150	*92*	*89*
200	*92*	*89*

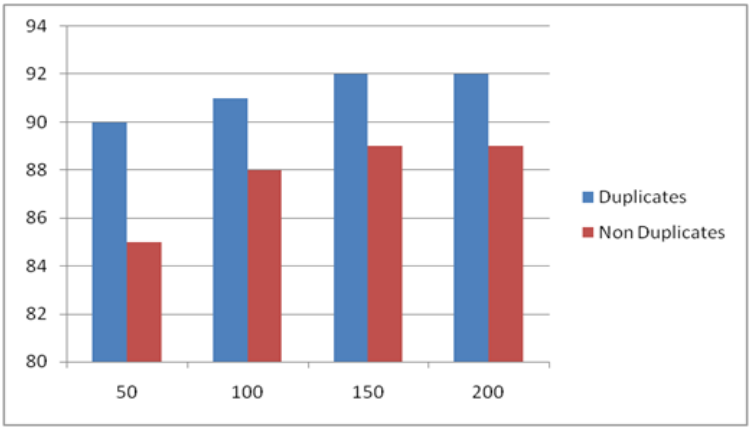

Fig. 4.6: Prediction Accuracy as a function of number of records in the training set (average values for a random set of 50 records in the test set). Horizontal axis shows the training set and the vertical axis shows the accuracy score.

It is evident from the results that more accuracy is achieved when the training set is large and the test set is small, and the accuracy decreases as the test set size increases. This is obvious results because the classification of a large number of data point of documents in vector space requires a large number of support vectors which can only be obtained if the training set is large.

Section 5 gives an insight into the results and conclude the paper and also gives an overview of the future scope of the proposed work.

V. CONCLUSION AND FUTURE SCOPE

There have been several statistical studies and surveys of existing bug repositories. Anvik et al.[] reported a statistical study on open bug repositories with some interesting results such as the proportion of different resolutions and the number of bug reports that a single reporter submitted. Sandusky et al. [] studied the relationships between bug reports and reported some statistic results on duplicate bugs in open bug repositories. Additionally, Hooimeijer and Weimer [] suggested a statistics based model to predict the quality of bug reports. After that, Bettenburg et al. [] made a survey on the developers of several well-known open source projects (Eclipse, Mozilla, and Apache) to study the factors that developers cared most on dealing with bug reports . Bettenburg et al. also suggested that duplicate bug reports were actually not harmful but useful for the developers. So the requirement for duplicate bug report detection became even stronger because it could not only reduce the waste of developer's time on duplicate bug reports but also helped developers to gather more related information to solve the bug more quickly. In general, none of these work proposed any approaches to duplicate-bug-report detection, but some of the work pointed out the motivation and effect of detecting duplicate bug reports.

In this work, we consider a new approach to detecting duplicate bug reports by building a incemental model that answers the question "Are two bug reports duplicates of each other ?". The model would report a score on the probability of A and B being duplicates. This score is then used to retrieve similar bug reports from a bug report repository for user inspection. The utility of the proposed approach on mozilla bug repositories from various versions is investigated. The experiment shows that our approach outperforms from the existing techniques by a relative improvement of 5–6%. As a future work, The utility of classifiers in machine learning in discriminative models for potential improvement in accuracy will be tackled. A completely different type of preprocessing needs to be done, in contrast to general natural language, which should be optimized for technical reports, and its utility will be investigated in improving detection of duplicate bug reports. The other interesting direction is adopting can include pattern-based classification to extract richer feature set that enables better classification of duplicate bug reports.

REFERENCES

[1] Zimmermann, T.; Premraj, R.; Sillito, J.; Breu, S., "Improving bug tracking systems," Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on , vol., no., pp.247,250, 16-24 May 2009.
[2] doi: 10.1109/ICSE-COMPANION.2009.5070993
[3] G. Tassey., The Economic Impact of Inadequate Infrastructure for software testing. National Institute of Standards and Technology. Planning Report: 2-03.2008-09, Chicageo, United States.
[4] Nistor, A.; Tian Jiang; Lin Tan, "Discovering, reporting, and fixing performance bugs," Mining Software Repositories (MSR), 2013 10th IEEE Working Conference, vol., no., pp.237,246, 18-19 May 2013.
[5] doi: 10.1109/MSR.2013.6624035
[6] Luo, L.; Hao, D.M.; Tian, Z.; Dang, Y.B.; Hou, B.; Malkin, P.; Yang, S.X., "Ariadne: An Eclipse-based system for tracking originality of source code," IBM Systems Journal , vol.46, no.2, pp.289,303, 2007
[7] doi: 10.1147/sj.462.0289
[8] Crowston, K., Annabi, H., & Howison, J. Defining Open Source Software project success. In Proceedings of the International Conference on Information Systems (ICIS 2003), Seattle, WA, USA, December.
[9] doi: 10.1287/mnsc.1060.0550
[10] Vijayakumar, K.; Bhuvaneswari, V., "How Much Effort Needed to Fix the Bug? A Data Mining Approach for Effort Estimation and Analysing of Bug Report Attributes in Firefox," Intelligent Computing Applications (ICICA), 2014 International Conference on , vol., no., pp.335,339, 6-7 March 2014 doi: 10.1109/ICICA.2014.75
[11] Anahita Alipour, Abram Hindle, and Eleni Stroulia. 2013. A contextual approach towards more accurate duplicate bug report detection. In Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13). IEEE Press, Piscataway, NJ, USA, 183-192.
[12] Yaxiong Li; Dan Hu, "Study on the Classification of Mixed Text Based on Conceptual Vector Space Model and Bayes," Asian Language Processing, 2009. IALP '09. International Conference on , vol., no., pp.269,272, 7-9 Dec. 2009 doi: 10.1109/IALP.2009.64
[13] C Sun, D Lo, X Wang, J Jiang, SC Khoo, " A discriminative model approach for accurate duplicate bug report retrieval ", Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, pp 45-54.
[14] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in DSN, 2008.
[15] A. Ko and B. Myers, "A linguistic analysis of how people describe software problems," in IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2006.
[16] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In Proceedings of the 30th international conference on Software engineering (ICSE '08). ACM, New York, NY, USA, 461-470. DOI=10.1145/1368088.1368151 http://doi.acm.org/10.1145/1368088.1368151
[17] N. Bettenburg, S. Just, A. Schr¨oter, C. Weiss,R. Premraj, and T. Zimmermann. What makes a good bug report? In SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pages 308–318, 2008.
[18] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information. In proceedings of the International Conference on Software Engineering, 2008.
[19] T. Menzies and A. Marcus. Automated severity assessment of software defect reports. In ICSM08:Proceedings of IEEE International Conference on Software Maintenance, pages 346–355, 2008.
[20] Chengnian Sun; Lo, D.; Xiaoyin Wang; Jing Jiang; Siau-Cheng Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," Software Engineering, 2010 ACM/IEEE 32nd International Conference on , vol.1, no., pp.45,54, 2-8 May 2010 doi: 10.1145/1806799.1806811
[21] J. Sutherland. Business objects in corporate information systems. In ACM Computing Surveys, 2006.
[22] G. Tassey. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology. Planning Report 02-3.2002, 2012.
[23] D. Lo, H. Cheng, J. Han, S.-C. Khoo, and C. Sun. Classification of Software Behaviors for Failure Detection: A Discriminative Pattern Mining Approach. In proceedings of the SIGKDD Conference on Knowledge Discovery and Data Mining, 2009.
[24] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful really? In ICSM08: Proceedings of IEEE International Conference on Software Maintenance, pages 337–345, 2008.
[25] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/cjlin/libsvm.
[26] Yu Chen; Fengguang Wu; Kuanlong Yu; Lei Zhang; Yuheng Chen; Yang Yang; Junjie Mao, "Instant Bug Testing Service for Linux Kernel," High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on , vol., no., pp.1860,1865, 13-15 Nov. 2013 doi: 10.1109/HPCC.and.EUC.2013.347.