Optimizing Automated Triaging Using Contextual Information and Feature Extraction
Shivani Gautam1 Nawagata Nilambari2
1,2Department of Computer Engineering,
1,2Galgotias University, Greater Noida, Utter Pradesh, India

Abstract— As software systems are getting larger and complex day by day, software bugs are inevitable. A software crash is one of the most severe manifestations of a defect (bug) in software, and is typically assigned a high priority to be fixed. To facilitate debugging process, many crash reporting systems such as Windows Error Reporting, Apple crash report, and Mozilla crash report have been deployed to automatically collect crash reports from users at the time of crash. Bug reports can also accompany other malfunctions of the software, mostly for the beta or unstable versions of the software. Most often, these bug reports are augmented with user contributed experiences as to what actually faced by him/her. Bugs occur for a variety of reasons, ranging from ill-defined specifications, to carelessness, to a programmers misunderstanding of the problem, technical issues, non-functional qualities, corner cases, etc. Addressing these bugs frequently accounts for the majority of effort spent in the maintenance phase of a software project’s life-cycle. This is why, researchers have been trying to enhance the bug-tracking systems to facilitate the bug-fixing process. The person who is in charge of processing the newly reported bugs, checking for duplicates and passing them to appropriate developers to get fixed is called a triager and this process is called triaging. In this paper, an model of automated triaging process is proposed based on contextual information and bug report textual similarity features. In proposed model, the preprocessed bug reports are analyzed for contextual information, to them to the non-functional requirements of the software. This extends the feature extraction a step further. The weighted sum of similarity score of all the features, including textual and contextual, is evaluated and used to classify bug reports. Positive and negative sets of existing classification are used to train SVM model. Simulation is done using R simulation package and the results shows a considerable improvement against classification without contextual information.

Key words: Information Retrieval, Feature Extraction, Support Vector Machines, Discriminative Model, Contextual Information, LDA

I. INTRODUCTION
A key collaborative hub for many software projects is a database of reports describing both bugs that need to be fixed and new features to be added. This database is often called a bug repository or issue tracking system. Open source software projects typically have a bug repository that allows both developers and users to post problems encountered with the software, suggest possible enhancements, and comment upon existing bug reports. Popular open source projects receive lots of bug reports. For most of the complex software, more bugs are reported than can be easily handled. Each report needs to be triaged, by a human, called the triager, to determine if reports are meaningful and if it does, it must be assigned to an appropriate developer for further handling. Moreover, one of the most important task of triager is to identify bug reports if these are duplicates of some previously submitted report related to some still-uncovered bug. In large projects where there are thousands of reports to search through, identifying duplicate bugs can be a daunting task. The use of a bug repository can improve the development process in a number of ways: it allows the evolution of the project to be tracked by knowing how many reports are outstanding, it allows developers who are geographically distributed to communicate about project development, it enables approaches to determine which developers have expertise in different areas of the product, it can help improve the quality of the software produced, and it can provide visibility to users about the status of problem reports. The bug repository can thus provide a location for users, developers, quality assurance teams and managers to engage in a user-integrated development process. However, the use of a bug repository also has a cost. Developers can become over-whelmed with the number of reports submitted to the bug repository.

Each report is triaged to determine if it describes a valid problem and if so, how the report should be categorized for handling through the development process. When developers are overwhelmed by reports, there are two effects. The first is that effort is redirected away from improving the product to managing the project. If a project gets thirty reports a day and it takes five minutes to triage a report, then over two-person hours per day are spent triaging reports. If all of these reports led to improvements in the code, this might be an acceptable cost to the project. However, for some projects, less than half of submitted reports lead to code improvements. For example, we found that the Eclipse project had 5515 unproductive reports in 2004. The second effect is that reports may not be addressed in a timely fashion. If the number of reports that enter the repository is more than can be reasonably triaged within a suitable amount of time for the project, then some reports may languish in the repository as other reports demanding more immediate attention take precedence. For an open-source project where the responsiveness of the development team to the community is often measured by how quickly reports are addressed and the number of outstanding reports, the rate at which reports are triaged can be an important factor in how the project is perceived. For example, Crowston et al. found that a measure of success for an open source project is the rate that users submitted bug reports and participated in project mailing lists. The person who triages the report, the triager, should have two goals. The first goal is to have the repository contain the smallest set of best reports for the project. The smallest set of best reports is desirable because reports typically enter the repository from a variety of sources, such as members of a technical support division, other developers, and the user community.
Unfortunately, with so many different sources of reports, some of the reports are not meaningful. For example, on a large project with many team members, several developers may submit a report describing the same bug. These duplicate reports need to be gathered together so that development effort is not wasted by having two developers solve the same problem. A triager also needs to filter reports that do not adequately enable a bug to be reproduced or that describe a problem whose cause is not the product, but rather is something beyond the control of the developers, such as the operating system. Sometimes, a triager also needs to filter out reports that are spam. Finally, a triager may indicate that the problem will not be fixed or that the feature will not be added to the product. Reports meeting any of these criteria must be identified so that development effort can focus on the reports that lead to product improvements. For example, nearly a third of the reports submitted to the Firefox project created between May 2003 and August 2005 were marked as duplicates. We call triage decisions that result in a report being designated as not meaningful as a repository-oriented decisions.

Bug tracking systems are the systems which manages the bug reports sent by a broad community of users. Usually, the users have different knowledge, skill and vocabulary level to issue report about a bug. Consequently, a bug tracking system is full of reports many of which are duplicates of each other, and bug triaging is time consuming and error prone. Triagers can become overwhelmed by the number of reports added to the repository. Time spent in triaging also typically diverts valuable resources away from the improvement of the product to the managing of the development process. This paper identifies the following key problem issues regarding the duplicate bug report retrieval process:

(1) Extracting the features from bug reports so that a similarity score between two reports can be established. To perform this task most accurately, three different types of features are extracted namely:

   (1) Textual Features
   (2) Categorical Features
   (3) Contextual Features

A total of 60 textual features are extracted using BM25F similarity score. In addition, categorical features are extracted from the relational database table attributes. The contextual feature extraction is done using Latent Dirichlet Allocation (LDA) using topic modeling with the non functional requirements of the software.

(2) Training a Support Vector Machine Classifier to classify incoming bug report as duplicate or non duplicate using Positive and Negative examples to be created from the existing bug repositories.

Neither natural language information, nor contextual information is always superior to the other in all cases. In particular, considering both kinds of information can have the following advantages. First, natural language information acquired from the bug description most likely represents the external buggy behavior observed by the bug reporter, while the corresponding execution information likely records the internal abnormal behavior. Thus, using both kinds of information can make it possible to consider both external and internal behaviors in duplicate-bug-report detection. Second, as descriptions in natural languages often contain uncertainty and imprecision, execution information, which is typically certain and precise, may help reduce the uncertainty and imprecision in existing duplicate-detection approaches. Moreover, as shown by the examples, either natural language information or execution information can be the dominant factor in detecting duplicate bug reports. Thus distinguishing which kind of information is the dominant factor may further facilitate duplicate-bug-report detection.

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 critical 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.

This paper is outlined as follows. Section I presents an overview of the subject matter and gives the problem statement and the approach for the research. Section II provides the overview of research methodology. Section III presents the proposed contextual and F60 textual and categorical model for comparison of an incoming bug report with positive and negative examples. Section IV gives the simulation results and the plots for various features for similarity measurement, and concludes the paper.

II. CONTEXTUAL INFORMATION RETRIEVAL

Many researchers have approached the bug-deduplication problem using off-the-shelf information-retrieval tools, such as BM25F used by Sun et al. In our work, we extend the state of the art by investigating how contextual information, relying on our prior knowledge of software quality, software architecture, and system-development (LDA) topics, can be exploited to improve bug-deduplication.

In this work, a new approach is proposed for improving the accuracy of detecting duplicate bug reports of a software system. Our approach exploits domain knowledge, about the software-engineering process in general, to improve bug-report checking for duplicates. The knowledge of the software process and product is exploited to find the context of the reports. This is because the bug reports are likely to refer to software qualities, i.e., non-functional requirements (possibly being desired but not met), or software functionalities (related to architectural components responsible for implementing them). Software dictionaries and word lists, exploited by prior research, are used to extract the context implicit in each bug report. Comparison is done between the contextual word lists to the bug reports and new features are recorded, in addition to the primitive textual and categorical features of the bug reports, such as description, component, type, priority, etc. Thus, our
The proposed work extends the number of features that can be extracted from bug reports, and thereby provides a similarity score, much more accurate as compared to one obtained through only textual or categorical analysis. These features are used to train the SVM creating positive and negative examples. The effectiveness of our contextual bug-deduplication method is demonstrated on the bug repository of the Android ecosystem.

A. Functional and Non Functional Requirements of Software

A functional requirement describes what a software system should do, while non-functional requirements place constraints on how the system will do so. Functional requirements are the main things that the user expects from the software. For example if the application is a banking application that application should be able to create a new account, update the account, delete an account, etc. Functional requirements are detailed and are specified in the system design. An example of a functional requirement would be that a system must send an email whenever a certain condition is met (e.g. an order is placed, a customer signs up, etc). A related non-functional requirement for the system may be that emails should be sent with a latency of no greater than 12 hours from such an activity. The functional requirement is describing the behavior of the system as it relates to the system’s functionality. The non-functional requirement elaborates a performance characteristic of the system.

Typically non-functional requirements fall into areas such as:

<table>
<thead>
<tr>
<th>Non Functional Requirements</th>
<th>Accessibility</th>
<th>Effectiveness</th>
<th>Interoperability</th>
<th>Portability</th>
<th>Resilience</th>
<th>Scalability</th>
<th>Supportability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity, current and forecast</td>
<td>Disaster Recovery</td>
<td>Extensibility</td>
<td>Maintainability</td>
<td>Quality</td>
<td>Response Time</td>
<td>Security</td>
<td>Testability</td>
</tr>
<tr>
<td>Compliance</td>
<td>Efficiency</td>
<td>Fault Tolerance</td>
<td>Privacy</td>
<td>Reliability</td>
<td>Robustness</td>
<td>Stability</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Non Functional Requirements of Software

Non-functional requirements are sometimes defined in terms of metrics (something that can be measured about the system) to make them more tangible. Non-functional requirements may also describe aspects of the system that does not relate to its execution, but rather to its evolution over time.

Bug reports are submitted by the stakeholder pertains to any of the non functional requirement expected by the user but not met. Thus, a few software dictionaries and word lists are utilized, exploited by prior research, to extract the context implicit in each bug report. This process is done using Latent Dirichlet Allocation (LDA).

The contextual word lists are compared to the bug reports and the comparison results as new features for the bug reports are recorded, 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 utilized to compare bug reports and detect duplicates.

III. OPTIMIZING AUTOMATED TRIAGING USING CONTEXTUAL INFORMATION AND FEATURE EXTRACTION

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, about the software-engineering process in general and the system specifically, 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. 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 Android project, which is a Linux-based operating system with several sub-projects. About 37000 Mozilla bug reports are analyzed in simulation. In this research, five different 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: Android architectural words, software Non-Functional Requirements words, Android topic words extracted applying Latent Dirichlet Allocation (LDA) method, Android topic words extracted applying Labeled-LDA method, and random English dictionary words (as a control). We indicate that our 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.
B. Proposed Model

Figure 3.1 shows the framework of the proposed technique.

![Proposed Framework](image)

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

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

Table 3.1: Typical Fields Of A Bug Repository (Relational Db Table)

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:

\[
\begin{align*}
F_{1}(B_1(title), B_2(title)) &= BM25F(B_1(title), B_2(title)) \\
F_{2}(B_1(title), B_2(Description)) &= BM25F(B_1(title), B_2(Description)) \\
F_{3}(B_1(title), B_2(Summery)) &= BM25F(B_1(title), B_2(Summery)) \\
F_{4}(B_1(Description), B_2(Title)) &= BM25F(B_1(Description), B_2(Title)) \\
F_{5}(B_1(Description), B_2(Description)) &= BM25F(B_1(Description), B_2(Description)) \\
F_{6}(B_1(Description), B_2(Summery)) &= BM25F(B_1(Description), B_2(Summery)) \\
F_{7}(B_1(Summery), B_2(Title)) &= BM25F(B_1(Summery), B_2(Title)) \\
F_{8}(B_1(Summery), B_2(Description)) &= BM25F(B_1(Summery), B_2(Description)) \\
F_{9}(B_1(Summery), B_2(Summery)) &= BM25F(B_1(Summery), B_2(Summery)) \\
F_{10}(B_1(Description) + Summery, B_2(Summery)) &= BM25F(B_1(Description) + Summery, B_2(Summery)) \\
F_{11}(B_1(Description) + Summery) &= BM25F(B_1(Description) + Summery)
\end{align*}
\]
Feature_{12}(B_1(\text{Summary}), B_2(\text{Description} 
+ \text{Summary})) 
= BM25F(B_1(\text{Summary}), B_2(\text{Description} 
+ \text{Summary}))

All above mentioned features can be considered for unigrams as well as bigrams, thus giving a total of 24 features.

This feature set is illustrated in the figure 3.2

Fig. 3.2: Illustration of feature extraction

The categorical features can be extracted in the following way:

Feature_{13}(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}

Feature_{14}(B_1(\text{Component}), B_2(\text{Product}))
= \begin{cases} 
1 \text{; if } B_1.\text{component} = B_2.\text{component} \\
0 \text{; otherwise}
\end{cases}

Feature_{15}(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}

Feature_{16}(B_1(\text{Priority}), B_2(\text{Priority}))
= \frac{1}{1 + |B_1(\text{Priority}) - B_2(\text{Priority})|}

Feature_{17}(B_1(\text{Version}), B_2(\text{Version}))
= \frac{1}{1 + |B_1(\text{version}) - B_2(\text{version})|}

Thus, features 1 to 12 accounts to textual similarity and features 13 to 17 accounts to categorical similarity.

This feature model, consisting of 17 features applies to most of bug repositories. However, this feature set can be extended using contextual similarity measurements using Latent Dirichlet Allocation by probabilistically finding out the non-functional requirements corresponding to the bug report. 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. 3.3: Architecture of Proposed Model, using Contextual Modeling

**DATA REPOSITORY**

<table>
<thead>
<tr>
<th>Master</th>
<th>Duplicate</th>
<th>n.1.</th>
<th>Duplicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master 1</td>
<td>Duplicate</td>
<td>1.1.</td>
<td>Duplicate</td>
</tr>
<tr>
<td>Master 2</td>
<td>Duplicate</td>
<td>2.1.</td>
<td>Duplicate</td>
</tr>
<tr>
<td>Master 3</td>
<td>Duplicate</td>
<td>3.1.</td>
<td>Duplicate</td>
</tr>
<tr>
<td>................</td>
<td>..................</td>
<td>..................</td>
<td>..................</td>
</tr>
<tr>
<td>Master n</td>
<td>Duplicate</td>
<td>n.1.</td>
<td>Duplicate</td>
</tr>
</tbody>
</table>

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.

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 positive 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 TECHNIQUES

A. 4.1 Analysis of Mozilla Firefox Bug Report

The Microsoft excel file for bug reports consists of several headers including the title, description and summary of the bug reports.

A portion of the file is shown here for ready reference:

<table>
<thead>
<tr>
<th>Bug_ID; Product; Component; Assignee; Status; Resolution; Title; Description; Summery; Changed</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>129576; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>Create download; Create download proxy for use in embedding.</td>
</tr>
<tr>
<td>192329; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>Downloaded files don't get correctly renamed if the/Users folder in a non-standard location.</td>
</tr>
<tr>
<td>206266; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>; getting ?KB; getting ?KB in status when saving from windows share.</td>
</tr>
<tr>
<td>233981; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>; [RFE] Add capability to request disconnect upon completion of download;</td>
</tr>
<tr>
<td>359375; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>; Request: iconify;</td>
</tr>
<tr>
<td>379784; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>The &quot;What&quot;; The &quot;What do you want me to do with this file&quot; dialog center is annoying, dialogs are truncated; The &quot;What do you want me to do with this file&quot; download dialog center is itself and is annoying.</td>
</tr>
<tr>
<td>404383; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>&quot;dialogs are truncated. Any download freeze freezes until a large amount of file is downloaded&quot;; &quot;dialogs are truncated&quot;;</td>
</tr>
<tr>
<td>404908; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>Any download; Any download freeze freezes until a large amount of file is downloaded. Continual Stalled Loading for Any Photo or Program that I try to download to my Desktop. Irritating.</td>
</tr>
<tr>
<td>410435; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>Continual Stalled Loading for Any Photo or Program that I try to download to my Desktop. Irritating. When resuming a download the estimated time and speed are incorrect&quot;; Continual Stalled Loading for Any Photo or Program that I try to download to my Desktop. Irritating.</td>
</tr>
<tr>
<td>411208; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>When resuming a download the estimated time and speed are incorrect&quot;; Continual Stalled Loading for Any Photo or Program that I try to download to my Desktop. I have a problem with Firefox.</td>
</tr>
<tr>
<td>411653; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>&quot;+&quot;; [RFE] Add; Javascript timer; Javascript timer blocks when downloading file.</td>
</tr>
<tr>
<td>411860; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>&quot;Firefox refuses&quot;; Firefox refuses to download anywhere but Local Settings /Temp&quot;; &quot;[RFE] Add; Javascript timer blocks when downloading file.</td>
</tr>
<tr>
<td>412419; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>&quot;Can not save&quot;; Javascript timer blocks when downloading file.</td>
</tr>
<tr>
<td>412810; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>Javascript timer; Javascript timer blocks when downloading file. Clean up nsDownloadManager : : GetRetentionBehavior();</td>
</tr>
<tr>
<td>414240; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>Clean-up nsDownloadManager : : GetRetentionBehavior(); Javascript timer blocks when downloading file.</td>
</tr>
<tr>
<td>415449; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>; Negative (0-12-0-42) time remaining when the file is bigger than the size announced by the HTTP server. download manager doesn't pop up while saving , doesn't show details of download and not able to save a web page.;</td>
</tr>
<tr>
<td>416006; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>; download manager; download manager doesn't pop up while saving, doesn't show details of download and not able to save a web page.;</td>
</tr>
<tr>
<td>416587; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>; Unable to; Unable to delete files partially downloaded. my download manager doesn't load anymore. can't save link as anymore. internet explorer can still download files though.;</td>
</tr>
<tr>
<td>416722; Toolkit; Download Manager; nobody; UNCONFIRMED;</td>
<td>; my download; my download manager doesn't load anymore. can't save link as anymore. internet explorer can still download files though.</td>
</tr>
</tbody>
</table>
manager doesn't load anymore. can’t save link as anymore. internet explorer can still download files though.;"40767.392650463"

417180.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Download Manager does not recognize space character. Download manager progress doesn't appear, but the download is going on.;"Download Manager does not recognize space character.;"40114.42575463"

418261.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Download manager progress doesn't appear, but the download is going on. Cannot Open Download manager.;"Download manager progress doesn't appear, but the download is going on.;"39660.1875"

418432.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Cannot Open.;"Cannot Open Download manager.;"39660.1875"

419160.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Firefox freeze.;"Firefox freeze system for 3-4 seconds when downloading any file with download manager.;"Firefox freeze system for 3-4 seconds when downloading any file with download manager.;"39660.1875"

419567.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Some larger downloads may tend to choke or freeze.;Firefox does not check return value of close() when downloading a file.;"Some larger downloads may tend to choke or freeze.;Firefox does not check return value of close() when downloading a file.;"40472.0822337963"

420358.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";firefox does.;"firefox does not check return value of close() when downloading a file. Websites do not work while downloading.;"firefox does not check return value of close() when downloading a file.;"40173.3876041667"

420895.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Website does.;"Website does not work while downloading.;"Website does not work while downloading.;"39660.1875"

420720.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Download manager;"download manager "hung" scanning download for a while.;"Download manager;"download manager "hung" scanning download for a while.;"39660.1875"

421100.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Download completes halfway through and file gets downloaded partially. Download manager/ Avira Antivirus Classic.;"Download completes halfway through and file gets downloaded partially. Download manager/ Avira Antivirus Classic.;"40267.5499652777"

421452.20;"Toolkit";"Download Manager";"nobody";"UNCONFIRMED";"";Download manager/ Avira Antivirus Classic.;"Download manager/ Avira Antivirus Classic.;"39660.1875"

<table>
<thead>
<tr>
<th>Topic 1</th>
<th>Topic 2</th>
<th>Topic 3</th>
<th>Topic 4</th>
<th>Topic 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 6</td>
<td>Topic 7</td>
<td>Topic 8</td>
<td>Topic 9</td>
<td>Topic 10</td>
</tr>
<tr>
<td>Topic 11</td>
<td>&quot;last&quot;</td>
<td>&quot;request&quot;</td>
<td>&quot;download&quot;</td>
<td>&quot;description&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;correctly&quot;</td>
<td>&quot;amongst&quot;</td>
<td>&quot;share&quot;</td>
<td>&quot;download&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;create&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

However, for mapping the bug reports to the list of non functional requirements one might need to train the topic modeling using a training set. It is desired to get the topic listing as depicted below as an example related to non functional requirements of the software.

<table>
<thead>
<tr>
<th>Topic 12</th>
<th>Topic 13</th>
<th>Topic 14</th>
<th>Topic 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;download&quot;</td>
<td>&quot;try&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Training to LDA architecture for topic modeling is a part of machine learning which can be implemented in R simulation using the following code.

```
library(RTextTools);
library(topicmodels);
library(tm);
matrix <- create_matrix(cbind(as.vector(data$description), as.vector(data$summary)), language="english",
removeNumbers=FALSE, weighting=weightTf)
rowTotals <- apply(matrix, 1, sum)
matrix.new <- matrix[rowTotals > 0,]
k <- length(unique(data$id));
Lda <- LDA(matrix.new, k);
terms(Lda); topics(Lda);
//code ends
```

Table 4.1: Bug Data Repository File

Following is the R simulation code to extract the topic labels out of summery and desctiption fields using LDA.

//code starts
install.packages(c("RTextTools", "topicmodels", "tm"));
library(RTextTools);
library(topicmodels);
library(tm);
data = read.table("sample.txt", sep=";", col.names=c("id", 
"product", "component", "assignee", "status", "resolution", 
"title", "description", "summary"), fill=FALSE, strip.white=TRUE)
matrix <- create_matrix(cbind(as.vector(data$description), as.vector(data$summary)), language="english",
removeNumbers=FALSE, weighting=weightTf)
rowTotals <- apply(matrix , 1, sum)
matrix.new <- matrix[rowTotals > 0,]
k <- length(unique(data$id));
Lda <- LDA(matrix.new, k);
terms(Lda);
topics(Lda);
//code ends

For the Mozilla bug repository considered, the topics listings extracted are:

- Topic 1
- Topic 2
- Topic 3
- Topic 4
- Topic 5
- Topic 6
- Topic 7
- Topic 8
- Topic 9
- Topic 10
- Topic 11
- "last"
- "request"
- "download"
- "description"
- "correctly"
- "amongst"
- "share"
- "download"
- "create"
- Topic 12
- Topic 13
- "download"
- "try"

```
library(topicmodels);
library(RTextTools);
install.packages(c("RTextTools", "topicmodels", "tm"));
library(tm);
data = read.table("training_set.txt", sep=";",
col.names=c("description", "topic"),
fill=FALSE, strip.white=TRUE)
train <- data
test <- data
train.lda <- LDA(train,5)
(train.topics <- topics(train.lda))
test.topics <- posterior(train.lda,test)
test.topics <- apply(test.topics,1,which.max)
```
The contents of the training_set.txt file are shown here for reference:

<table>
<thead>
<tr>
<th>description</th>
<th>“topic”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create download proxy for use in embedding;</td>
<td>efficiency</td>
</tr>
<tr>
<td>Downloaded files don’t get correctly renamed if the /Users folder is in a non-standard location;</td>
<td>accuracy</td>
</tr>
<tr>
<td>getting ??KB in status when saving from windows share;</td>
<td>defect</td>
</tr>
<tr>
<td>[RFE] Add capability to request disconnect upon completion of download;</td>
<td>defect</td>
</tr>
<tr>
<td>Request: iconify Firefox to system tray when last browsing window closes, but there are downloads pending;</td>
<td>defect</td>
</tr>
<tr>
<td>The “What do you want me to do with this file” download dialog centers itself and is annoying;</td>
<td>efficiency</td>
</tr>
<tr>
<td>dialogs are truncated;</td>
<td>robustness</td>
</tr>
<tr>
<td>Any download freeze firefox until a large amount of file is downloaded;</td>
<td>efficiency</td>
</tr>
<tr>
<td>Continual Stalled Loading for Any Photo or Program that I try to download to my Desktop. Irritating;</td>
<td>correctness</td>
</tr>
<tr>
<td>When resuming a download the estimated time and speed are incorrect;</td>
<td>efficiency</td>
</tr>
</tbody>
</table>

Table 4.2: Illustration Of Training Set For Bug Report Descriptions And The Corresponding Non Functional Requirements

Using these semantic features along with the textual and categorical features can be used to train the SVM model.

B. Support Vector Machines

Support Vector Machines (SVMs) are a new generation learning system based on recent advances in statistical learning theory. SVMs deliver state-of-the-art performance in real-world applications such as text categorization, handwritten character recognition, image classification, bio-sequences analysis, etc., and are now established as one of the standard tools for machine learning and data mining.

Support Vector Machines can be viewed as statistical learning systems that receives data and observations as input and outputs a function that can be used to predict some features of the future data. Statistical learning theory models this as a function estimation problem. Generalization Performance (accuracy in labeling test data) is measured as training parameter for SVM.

Basic idea of support vector machines is to find Optimal Hyper-plane for linearly separable patterns and to extend to patterns that are not linearly separable by transformations of original data to map into new space through the Kernel function.

The red and blue circles corresponds to two different classes of data points. It is evident from the above figure that multiple solutions can be provided to the set of data that is linearly classifiable. Each of the lines depicted above provides a different degree of classification. A plane is said to be the best amongst others if it passes farthest from all the points and this measurement is the metric for accuracy of the classification. The operation of the SVM is to find a hyper plane that passes farthest from all the training points close to the hyper plane of which decides the boundary of the hyper plane are called support vectors. Twice of this distance is called the margin. An optimal hyper plane is the hyper plane in which the margin between support vectors is maximum.

The above figure shows that there exists multiple hyper planes that can classify the given data. The optimal hyper plane is one which passes farthest away from all the points. The operation of the SVM algorithm is to find such an optimal hyper plane.

C. Analysis of Bug.xls using SVM in R package

To train the SVM, one needs to create training sets corresponding to the two classes of classification. In context of yes or no categorization of a set of data, these training sets are called positive and negative examples. For current context of checking of delicacy in bug reports, these positive and negative examples can be created to predict whether the incoming bug report corresponds to duplicate or non duplicate class.

1) Positive Examples

Positive examples can be constructed using pairs of reports in which one report is the master report and the other is one of its duplicates, or these can be two duplicate bug reports from the same bucket which corresponds to the same master report.

2) Negative examples

These are pairs of bug reports which are randomly selected from different buckets. That is, in the pairs corresponding to
negative bug reports, both the reports correspond to different master reports.

Positive and negative examples are constructed using a total of about 100 pairs for each. However, number of negative examples can exceed many fold as compared to number of positive examples. The pairs of bug reports corresponding to negative examples should be chosen from different containers so as to cover all possible combinations of negative examples.

The simulation over the dataset (bug_report.csv) produces the following output for prediction matrix:

```
Call:
svm(formula = nfr ~ ., data = data, method = "C-classification", kernel = "radial", cost = 100, gamma = 1)
Parameters:
  SVM-Type:  C-classification
  SVM-Kernel:  radial
  cost:  100
  gamma:  1
Number of Support Vectors:  10
  ( 4 1 3 1 1 )
Number of Classes:  5
Levels:
  accuracy correctness defect efficiency robustness
```

Fig. 4.3: Similarity score of an incoming report with textual feature (total 17) Vertical axes shows the similarity score. Horizontal axes is equally separated for 100 examples.

Fig. 4.4: Similarity score of an incoming report with textual and Categorical Features. Vertical axes shows the similarity score. Horizontal axes is equally separated for 100 examples.

Fig. 4.5: Similarity score of an incoming report with Textual, Categorical and Contextual Features. Vertical axes shows the similarity score. Horizontal axes is equally separated for 100 examples.

The above results clearly suggests that the use of contextual features extend the feature set thereby providing more accurate classification of duplicate bug reports. Nevertheless, this accuracy comes at the cost of processing the incoming bug reports for mapping with non functional requirements using machine learning algorithms. However, for most of the real world open source projects, this extra processing cost is not considerable in comparison to the accuracy in the triaging process.

D. Comparison of the results

Using the same dataset (bug.xls), the different simulations can be carried out using textual, categorical and contextual features. Using the feature set by Anahita Alipour et al [9] using 4 textual, contextual and ~10 categorical features, one can get a fairly classified dataset into positive and negative examples. However, the proposed work uses contextual information with 60 textual features and all categorical features of the relational database table, gives a significant improvement over [9].

Table 4.3 gives the feature set comparison with [9] on all possible domains.

<table>
<thead>
<tr>
<th>Feature set</th>
<th>title description</th>
<th>Summary</th>
<th>BiGrams</th>
<th>Cross Comparisons</th>
<th>Categorical</th>
<th>Contextual</th>
<th>Training Model using SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>all possible</td>
<td>yes</td>
<td>yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Anahita Alipour et.al.</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>Selective</td>
<td>selecive</td>
<td>yes</td>
<td>Other machine learning algorithm.</td>
</tr>
</tbody>
</table>

Table 4.3: Feature Set Comparison

Table 4.4 shows the corresponding confusion matrix regarding the prediction data for bug.xls.

<table>
<thead>
<tr>
<th>Non Functional Requirements</th>
<th>Records</th>
<th>accuracy</th>
<th>correctness</th>
<th>Defect</th>
<th>efficiency</th>
<th>robustness</th>
</tr>
</thead>
</table>

All rights reserved by www.ijsrd.com
The experiment shows that our approach outperforms from the existing techniques by a relative improvement of 8–13%. As a future work, The utility of classifiers in machine learning using a larger set of non-functional requirements related to contextual features of bug reports, 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 discrimination and detection of duplicate bug reports.

V. CONCLUSION AND FUTURE SCOPE

Retrieving contextual information from a textual document is a relatively new subject in the field of bug report duplicity checking. In this paper, the previously specified techniques for feature extraction using textual and categorical information is augmented with feature extraction corresponding to contextual information. Thus this paper provide a framework for a rich feature set including all possible features (till date) related to bug reports. These feature models are used to train the SVM over a set of positive and negative examples.

There have been several statistical studies and surveys of existing bug repositories. Anvik et al.[11] 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. [12] studied the relationships between bug reports and reported some

statistic results on duplicate bugs in open bug repositories. Additionally, Hoomeijer and Weiner [13] suggested a statistics based model to predict the quality of bug reports. After that, Bettenburg et al. [14] 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 [14], 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.

The experiment shows that our approach outperforms from the existing techniques by a relative improvement of 8–13%. As a future work, The utility of classifiers in machine learning using a larger set of non-functional requirements related to contextual features of bug reports, 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 discrimination and detection of duplicate bug reports.

REFERENCES


| Test Record 1 | 1 | 1 | 1 | 1 | 1 |
| Test Record 2 | 0 | 0 | 0 | 1 | 1 |
| Test Record 3 | 0 | 0 | 1 | 1 | 0 |
| Test Record 4 | 1 | 0 | 1 | 1 | 1 |
| Test Record 5 | 0 | 0 | 1 | 1 | 0 |
| Test Record 6 | 0 | 1 | 0 | 0 | 1 |
| Test Record 7 | 1 | 0 | 1 | 0 | 1 |
| Test Record 8 | 0 | 1 | 1 | 1 | 1 |
| Test Record 9 | 0 | 0 | 0 | 0 | 0 |
| Test Record 10 | 0 | 0 | 0 | 0 | 0 |

Table 4.4: Feature Set Comparison [Anahita Alipour El.AI]

<table>
<thead>
<tr>
<th>Non Functional Requirements</th>
<th>Records</th>
<th>accuracy</th>
<th>correctnes s</th>
<th>defec t</th>
<th>efficienc y</th>
<th>robustnes s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Record 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Test Record 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Test Record 3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Test Record 4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Test Record 5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Test Record 6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Test Record 7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Test Record 8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Test Record 9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Test Record 10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5: Feature Set Comparison [Proposed]

The cells in which the entry is 1 shows that predicted class by SVM is the actual class whereas a zero corresponds to a false detection.

It is clearly evident from the above table that the proposed work provides a more accurate classification of the duplicate detection problem as compared to previous approaches.

The improvement in the present context mentioned above is (26-24) / 24 = 8% over the baseline as specified by [9]. However, the proposed method can provide much more accurate results when bug repository database is large and LDA process can be rigorously applied for contextual feature extraction.
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4630070&isnumber=4630050

doi: 10.1109/ASE.2011.6100061
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6100061&isnumber=6100039

doi:10.1109 / CSMR.2012.48
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6178884&isnumber=6178853


doi: 10.1109/ASE.2011.6100061


http://doi.acm.org/10.1145/1321631.1321639