Bug Report Triaging using Textual, Categorical and Contextual Features using Latent DIRICHLET Allocation

Anuradha Sharma
M.Tech. Scholar
Department of Computer Science and Engineering
Rajasthan Institute of Engineering and Technology, Jaipur

Sachin Sharma
Associate Professor
Department of Computer Science and Engineering
Rajasthan Institute of Engineering and Technology, Jaipur

Abstract

Software Bugs occur for a wide range of reasons. Bug reports can be generated automatically or drafted by user of software. Bug reports can also go with other malfunctions of the software, mostly for the beta or unsteady versions of the software. Most often, these bug reports are improved with user contributed experiences as to know what in fact faced by him/her. Addressing these bugs accounts for the majority of effort spent in the maintenance phase of a software project life cycle. Most often, several bug reports, sent by different users, match up to the same defect. Nevertheless, every bug report is to be analyzed separately and carefully for the possibility of a potential bug. The person responsible for processing the newly reported bugs, checking for duplicates and passing them to suitable developers to get fixed is called a Triager and this process is called Triaging. The utility of bug tracking systems is hindered by a large number of duplicate bug reports. In many open source software projects, as many as one third of all reports are duplicates. This identification of duplicacy in bug reports is time-taking and adds to the already high cost of software maintenance. In this dissertation, a model of automated triaging process is proposed based on textual, categorical and contextual similarity features. The contribution of this dissertation is twofold. In the proposed scheme a total of 80 textual features are extracted from the bug reports. Moreover, topics are modeled from the complete set of text corpus using Latent Dirichlet Allocation (LDA). These topics are specific to the category, class or functionality of the software. For e.g., possible list of topics for android bug repository might be Bluetooth, Download, Network etc. Bug reports are analyzed for context, to relate them to the domain specific topics of the software, thereby; enhancing the feature set which is used for tabulating similarity score. Finally, two sets are made for duplicates and non-duplicate bug reports for binary classification using Support Vector Machine. Simulation is performed over a dataset of Bugzilla. The proposed system improves the efficiency of duplicacy checking by 15 % as compared to the contextual model proposed by Anahita Alipour et.al. The system is able to reduce development cost by improvising the duplicity checking while allowing at least one bug report for each real defect to reach developers.

Keywords: Bug Reports, Textual Features, Contextual Features, Automated Triaging, Support Vector Machines, Classification

I. INTRODUCTION

A. Introduction:

A bug tracking system or defect tracking system is a software application that keeps record of reported software bugs in software development projects. It may be considered as a type of issue tracking system. Most of the bug tracking systems [1], such as those used by many open source software projects, allows end-users to enter bug reports directly. Other systems are used only inside a company or organization involved in doing software development. In general bug tracking systems are integrated with other Software Project Management applications.

The database of reports is critical for many software projects as it describes both bugs that need to be fixed and the new features to be added. Open source software projects typically have a bug repository that allows both developers and users to post problems encountered with the software, suggesting of possible enhancements, and commenting upon existing bug reports. Many popular open source projects get vast number of bug reports [2]. For most of the complex software, more bugs are reported than can be easily handled. Each report needs to be triaged, by a person, known as triager [3], to determine if reports are meaningful and if it does, it must be assigned to an suitable 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 an expensive task. The use of a bug repository can improve the development process in a number of ways:

1) It allows the evolution of the project to be tracked by knowing how many reports are outstanding.
2) It allows developers who are geographically distributed to communicate about project development.

3) It enables approaches to determine which developers have expertise in different areas of the product.
4) It can help improve the quality of the software produced, and
5) It can provide visibility to users about the status of problem reports.

The bug repository can thus provide a location for users, quality assurance teams, developers 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 lead to improvement in the code, this might be satisfactory cost to the project. However, for some projects, less than half of submitted reports lead to code improvements. For example, it was found that the Eclipse project had 5515 unproductive reports in 2004 [4].

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 determining how well the project refinement is taking place. For example, Crowston et al. [5] 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, 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 various sources, such as members of a technical support division, other developers, and the user community. Unfortunately, with so many diverse 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 somewhat which is beyond the control of the developers, such as the operating system. Sometimes, a triager also needs to filter the 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 criterion 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 [6]. One can call triage decisions that result in a report being designated as not meaningful as a repository-oriented decisions.

B. Problem Statement:

Bug Repositories 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 formulate 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:*
   a) Textual Features
   b) Categorical Features
   c) Domain Specific Contextual Features

A total of 60 textual features are extracted using TF-IDF similarity score. In addition, categorical features are extracted from the relational database table attributes. The contextual feature extraction is done using Latent Dirchlet Allocation (LDA) using topic modeling with the domain Specific Features 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.*

C. Motivation:

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 contextual 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, contextual information, obtained by human triager or automated topic modeling, may help reduce the uncertainty and imprecision in existing duplicate-detection approaches. Moreover, as shown by the examples,

either natural language information or contextual 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.

II. PROBLEM DEFINITION

Managing the incoming traffic 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, a machine learning approach is presented 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. The proposed approach is based on the classification technique, for which Support Vector Machine based classifier is used. Experimental evaluation of duplicacy checking using precision and recall measures reveals the feasibility of proposed approach for automatic bug triaging.

Many researchers have approached the bug report duplication problem using off-the-shelf information-retrieval tools, such as BM25F used by Sun Microsystems. In this work, the state of the art vector space model is extended by investigating how contextual information, relying on the prior knowledge of software quality, system-development (Latent Dirichlet Allocation [7]) topics and software architecture can be exploited to improve bug report duplicacy checking.

In this work, a new approach is proposed for improving the accuracy of detecting duplicate bug reports of a software system. This approach exploits domain knowledge to improve bug-report checking for duplicates. The knowledge of the software product is exploited to find the context of the reports. This is because the bug reports are likely to refer to software qualities, Software dictionaries and word lists, exploited by prior research, are used to extract the context implicit in each bug report. This context is software specific and the topics of two software may be entirely different. For example, topics of VLC player might list recording, zooming, crash etc., while topics of Mozilla might include download manager and history-tab. 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, priority, component, type etc. Thus, 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 triaging method is demonstrated on the bug repository of the Android ecosystem using R statistical package.

III. PROPOSED WORK

As the vast collective knowledge continues to be digitized and stored—in the form of news, blogs, video, Web pages, scientific articles, books, images, sound, and social networks—it becomes more difficult to find and discover what we are looking for. New computational tools are needed to help organize, understand and search these vast amounts of information. Right now, working with online information using two main tools—search and links is employed. Keywords are typed into a search engine and set of documents is found related to them. Documents in that set are looked upon, possibly navigating to other linked documents. This is a influential way of interacting with our online archive, but something is missing. Imagine searching and exploring documents based on the themes that run through them. It might need to be "zoom in" and "zoom out" to find specific or broader themes; or might look at how those themes changed through time or how they are connected to each other. Rather than finding documents through keyword search alone, there is a need to first find the theme that one is interested in, and then examine the documents related to that theme

While more and more texts are available online, Nobody has the power to read and study them to provide the kind of browsing experience described above. In the end, machine learning researchers have developed probabilistic topic modeling, a set of algorithms that aim to discover and explain large archives of documents with thematic information. Topic modeling algorithms are arithmetical methods that analyze the words of the original texts to discover the themes that run through them, how those themes are associated to each other, and how they modify over time.

A. Classification of Duplicate and Non-Duplicate Bug Reports:

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 systems 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, priority, type, component etc. proposed in Sun et al.[9] 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.

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:

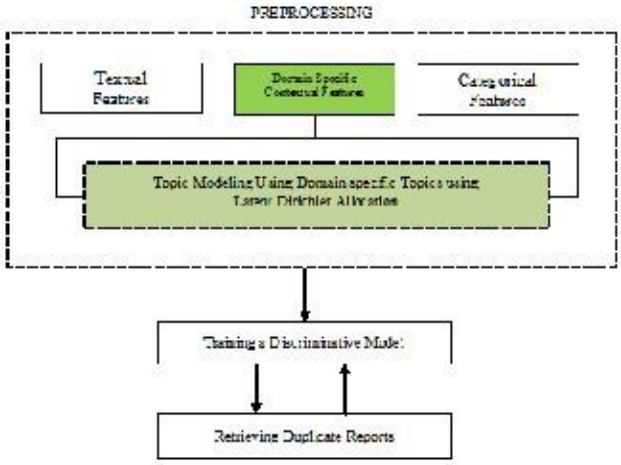

Fig. 3.1: shows the framework of the proposed technique.

Figure 3.1 Proposed Framework of LDA Topic Modeling

First phase of processing 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, symbols, phrases 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.

1) *Textual Features:*

A feature extraction process is used to extract the features between duplicate and non duplicate pairs and then these features are provided as input to the SVM learning algorithm to build a suitable discriminative model.

To create positive example, pairing can be done between master and duplicate from the same bucket or two duplicates from the same bucket. To create a negative example, pairing is done between master from one bucket and duplicates from another buckets or two duplicates from different buckets. It is immediately apparent that the number of negative examples is far more than the number of positive examples. However, suitable negative examples can be chosen to keep the number of two examples identical.

Feature engineering and extraction can be done to find the similarity (or dis-similarity) between two reports. Limited features make it hard to differentiate between two contrasting datasets: pairs that are duplicates and pairs that are non-duplicates. Hence a rich enough feature set is needed to make duplicate bug report retrieval more accurate.

Textual similarity can be given in terms of features. A feature is actually the similarity between two bags of words from two reports R_1 and R_2.

$$f(R_1 \text{ and } R_2) = sim(words\ from\ R_1, words\ from\ R_2) \qquad equation\ 3.1$$

The following formula is used as a similarity measurement:

$$sim(B_1 \text{ and } B_2) = \sum_{w \epsilon B_1 \cap B_2} idf(w) \qquad equation\ 3.2$$

sim(B1,B2) returns the similarity between two bags of words B_1 and B_2. The resemblance is the sum of idf values of all the shared words between B_1 and B_2. The idf value for each word is computed based on a corpus formed from all the reports in the repository.

Considering a generic format of bug reports consisting of title, description and summery, the f-60 model can be described as shown:

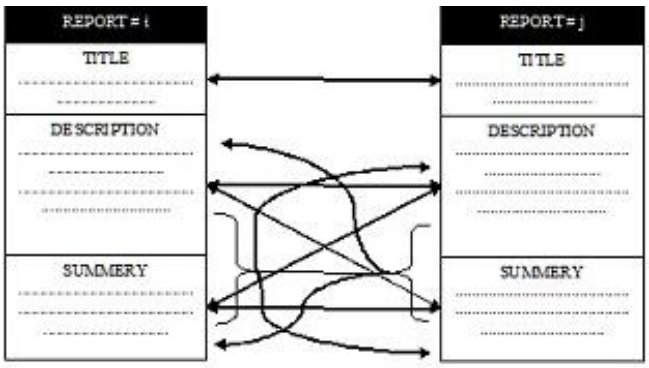

Fig. 3.2: Pairs of Textual similarity combinations

The Feature (similarity measurement) between two reports can be extracted in the following way:

1) Title to Title
2) Description to description
3) Summery to Summery
4) Description (R_1) to Summery (R_2)
5) Summary (R_1) to Description (R_2)
6) Description and summery taken together (R_1) - to- Description and summery taken together (R_2).
7) Description and summery taken together (R_1) to Description (R_2).
8) Description and summery taken together (R_1) to Summery (R_2).
9) Description and summery taken together (R_2) to Description (R_1).
10) Description and summery taken together (R_2) to Summery (R_1).

Considering each of the combinations as a separate feature, the total number of different features would be 10. Furthermore, one can compute three types of idf, as the bug repository can form three distinct corpora. One corpus is the collection of all the summaries, one corpus is the collection of all the descriptions, and the other is the collection of all the both (summary+description).

The three types of idf computed within the three corpora, are defined by idf^{sum}, idf^{desc}, and idf^{both} respectively. The output of function f defined in equation (3.1) depends on the choice of bag of words for R_1, the choice of bag of words for R_2 and the choice of idf. Considering each of the combinations as a separate feature, the total number of different features would be 10×3, which is equal to 30. Aside from considering words, bigrams can also be considered. A bigram refers to two consecutive words. With bigrams, considering different combinations of bag of words coming from and idf computed based on summaries, descriptions, or both, there are another 30 features which would then bring the number of features extracted to 60.

2) *Categorical Features:*

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.*

3) *Contextual Features:*

The 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.

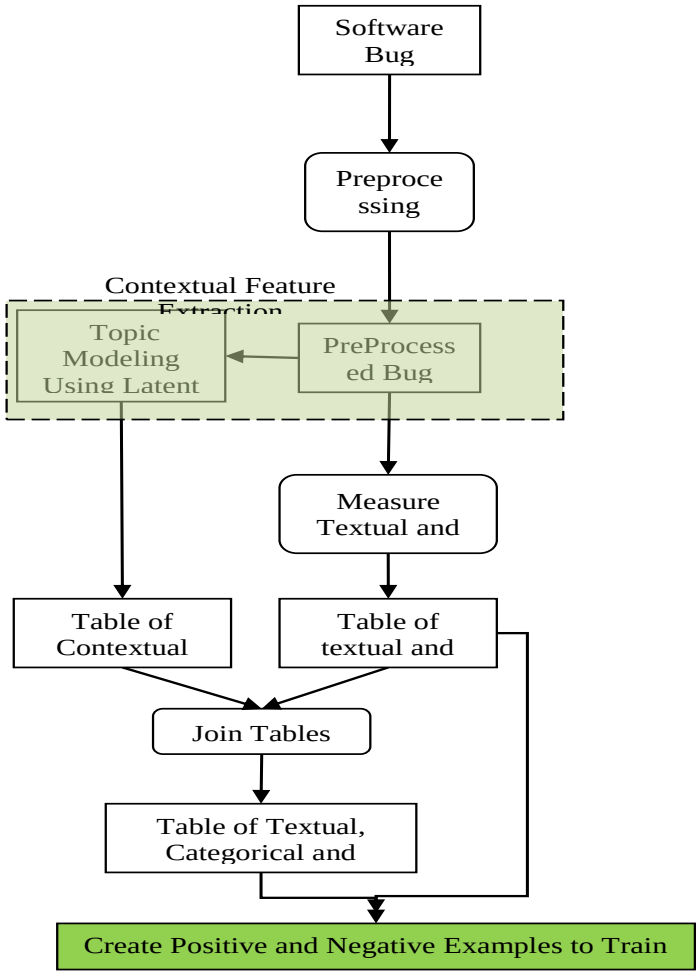

Fig 3.3: Proposed Architecture for Classification of Duplicates and Non-Duplicate Bug Reports

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

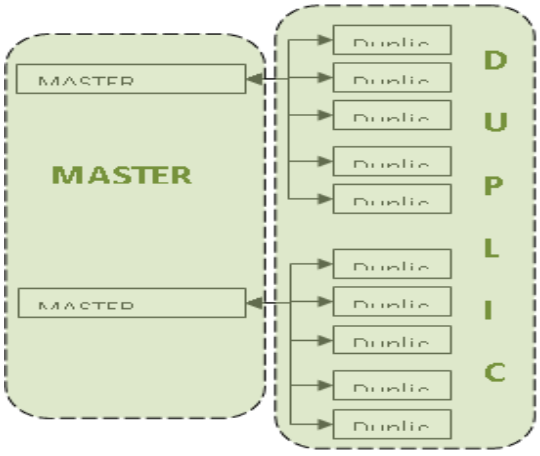

Fig. 3.4: Data Structure for Proposed System

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.

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. Bug Repository of Android Applications (Android Issue Tracker):

Few of the records corresponding to the issues reported to the tracking system of Android Play Store are tabulated in table 4.1 shown below:

TABLE - 4.1
SAMPLE RECORDS OF BUG REPOSITORY WITH BUG CATEGORY

Id	*title*	*Summary*	*Description*	*Category (DOMAIN SPECIFIC TOPICS)*
1	*tethering*	*FreeNAS is a program to turn a...*	*Updated to 4.1.2 yesterday on yakju galaxy nexus. Tethering does not work anymore. Devices connect but cannot contact the internet. Blue tooth tethering disconnects almost immediately. Wired and wireless stay connected but don't provide actory flash of 4.1.2 the problem persists.*	*Printing*
2	*galaxy nexus bluetooth*		*I can attest that Yakju 4.1.2 update on stock Galaxy Nexus broke radio, bluetooth and wifi. The phone enters airplane mode on its own. The radio loses all telco signal (only recover after a 10 second airplane mode). Bluetooth pretends to connect but the headset is unusable. Wiifi that the same issue as bluetooth. All in all, the phone connectivity is in a really poor state. Most people could not reach me today and I had to keep checking the phone the whole day to see if the radio/wifi was still working which it mostly wasn't.*	*Ui*

Corresponding to each of the attributes of the repository depicted above, the frequency of occurrences of important terms can be depicted using wordcloud as shown below.

B. Word Frequency Counts:

The wordcloud depicts a measure of the relative frequency of a word in the text corpus The size of a word is proportional to its relative (normalized) frequency in the text corpus in comparison to other words. The WordClouds corresponding to title, description, summary and category are depicted in figure 3.1 to fig 3.4 respectively.

Fig. 4.1: WordCloud for Title Corpus of Android Bug Reports

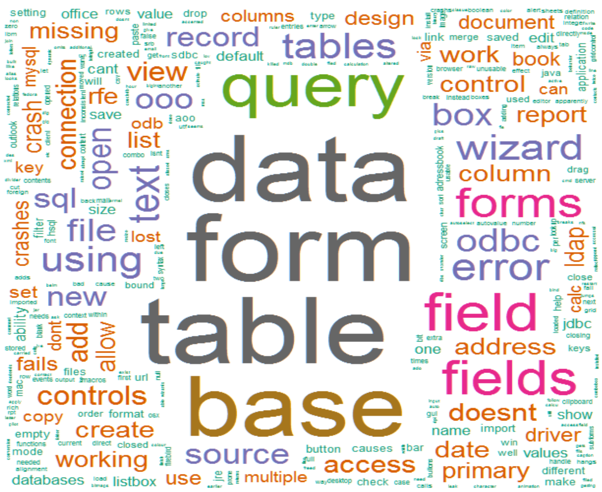

Fig. 4.2: WordCloud For Summary Corpus of Android Bug Reports

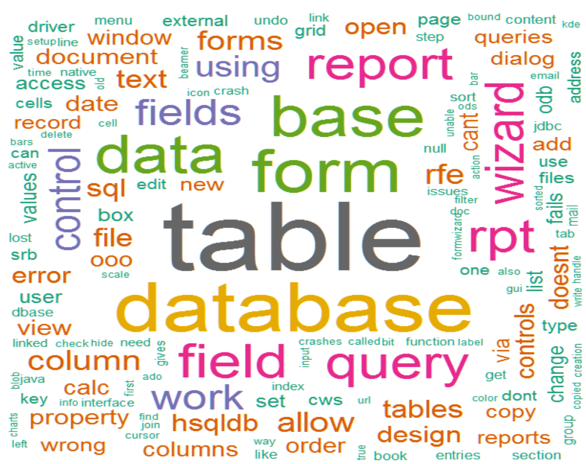

Fig. 4.3: WordCloud for Description Corpus of Android Bug Reports

Fig. 4.4: WordCloud For Category Corpus of Android Bug Reports

On the basis of relative frequency of occurrences of the words in the above specified corpus, the correlation values are depicted as shown in the figure 3.5.

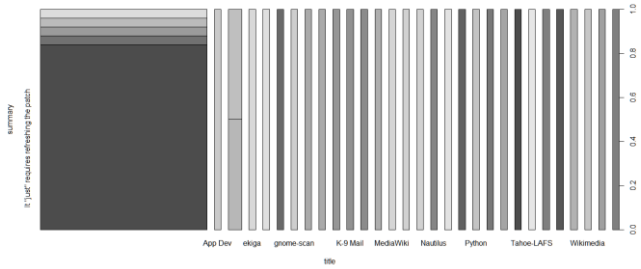

Fig. 4.5: Correlation of Word Frequencies between title and Summery

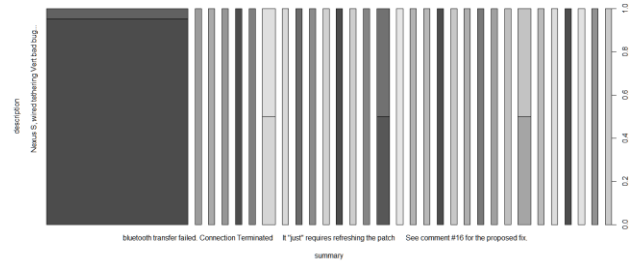

Fig. 4.6: Correlation of Word Frequencies between Summery and Description

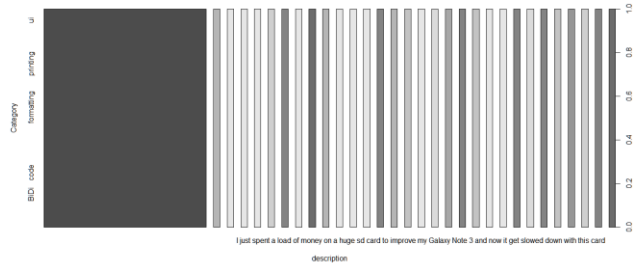

Fig. 4.7: Correlation of Word Frequencies between Category and Description

Support Vectr Machine is designed for classification of duplicates and non-duplicates as per the following specifications:

Table -4.2

Parameter Specifications Of SVM

Support Vector Machine object of class "ksvm"
SV type: C-svc (classification)
parameter : cost C = 100
Linear (vanilla) kernel function.
Number of Support Vectors : 344
Objective Function Value : -34210.61
Training error : 0.018

The textual feature set can be obtained from tf-idf measure which provides a total of 60 textual features. These features are then combined with 12 categorical features which are typical attributes of any bug repository. The contribution of this work is to augment this feature set of textual and categorical feature sets with contextual features obtained from topic modeling using Latent Dirchlet Allocation.

The feature set for classification of duplicates and non-duplicates, in terms of contextual measures, in the bug repository can be done by creation of positive and negative examples. The positive and negative examples is created from open office repository which consists a total of 10000 records. The illustration of positive and negative examples with a binary classifier is as shown in figure 3.8.

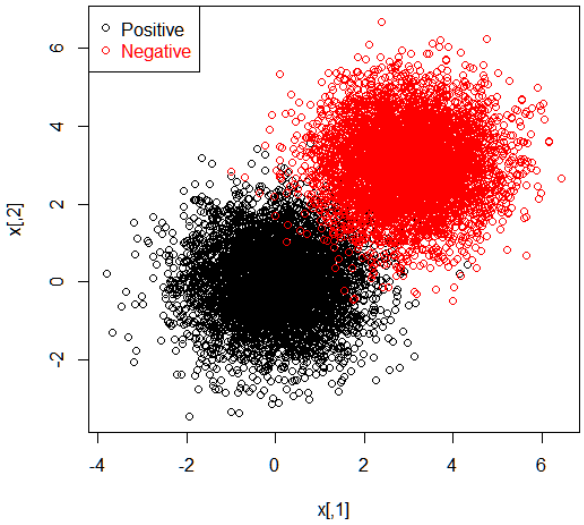

Fig. 4.8: Illustration of SVM with Linear Classification Function

Support Vector Machine can be trained by creating training set from a pre analyzed data set known as the training data set. This training data set provides a means for supervised learning. After training, the SVM can be tested over incoming bug reports to obtain the precision and recall rate measures.

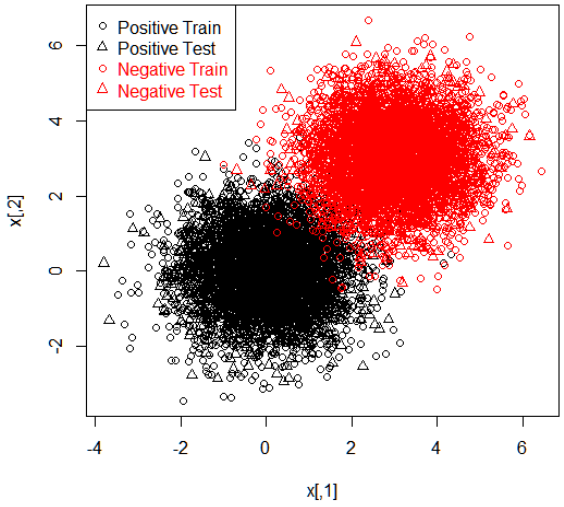

Fig. 4.9: SVM Classification for Testing and Training Data

This classification of duplicates in terms of contextual topics can be used to augment the feature set used for the checking of duplicacy in the bug repository. This gives a substantial improvement in the classification of the bug reports related with a particular version of the software.

C. Performance Measures:

The Precision Rate of the Trained SVM of another set of 500 new bug reports is depicted in figure 4.10. The Recall rate of the system is defined as percentage of the relevant results in the repository retrieved as search results in response to a query. Recall rate is used as a measure of effectiveness of the duplicacy checking method. The proposed scheme goes well beyond the recall rates as achieved by anahita alipour et.al as it matches the incoming bug reports to domain specific topics which are basically the categoeries of bug reports. The recall curve of the proposed scheme for topic modeling is depicted in figure 4.11 below.

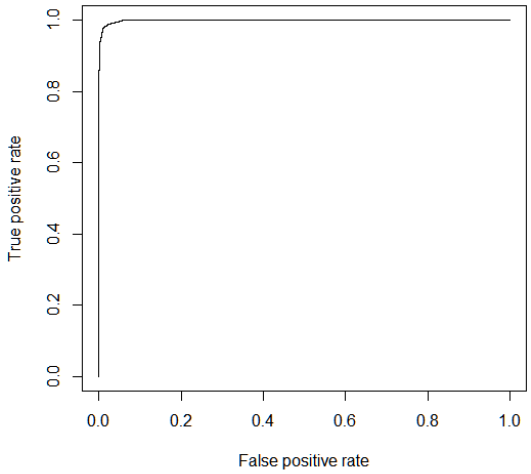

Fig 4.10: The Precision Rate of the Trained SVM

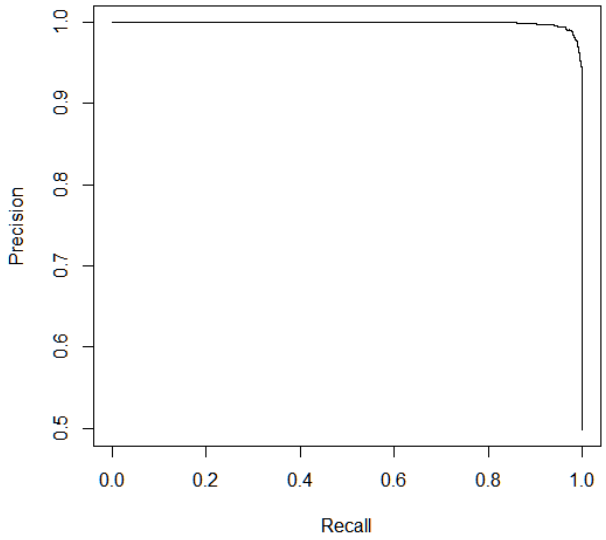

Fig. 4.11 Precision versus Recall measure of SVM

Section 5 compares the results with those of earlier authors and concludes the Paper. It also gives a brief outline of the scope of future work.

V. CONCLUSION AND FUTURE SCOPE

A system is proposed that automatically classifies duplicate bug reports as they arrive thereby saving developer's time. The proposed system uses textual, categorical and contextual features. The contribution of this paper is to extend the feature set used for classification of duplicated bug reports. This is done by augmenting the feature set with the contextual features which relates the bug reports to the domain specific topics. This topic modeling is done by identifying the topics and training a classifier for mapping bug reports to the appropriate topic. Topic modeling using latent Dirichlet allocation is used so a to map a document to certain topics through Dirichlet Distribution. Results are empirically evaluated using R statistical package. A dataset of 10,000 bug reports is analyzed from the Bugzilla project.

The proposed system is able to reduce development cost by filtering out duplicate bug reports. It also give much more precision in case of comparatively small software packages having relatively small number of topics to which a bug report is to be matched. Moreover, there is a high probability of a bug report found duplicate of some other bug report sent previously if they both belong to the same topic, say, for example, two bug reports of android-lollipop which are related with Bluetooth and downloads are more likely to be duplicates of each other.

VI. FUTURE SCOPE

A more accurate bug report triaging system can only be prepared with domain specific knowledge of the software. Such a system cannot achieve substantial recall rate and precision if it is modeled in a generic manner. As a future prospective of the work, more accurate bug report duplicacy checking will be made by inculcating the domain specific words so that the software code segments that results in crashes or failures might be figured out to precisely analyze the duplicacy of the bug reports. One example of this might be: "ArrayIndexOutofBounds". This combination of words is a system generated message which is actually specific to the memory overrun error of java programming environment. Another improvement that will be made is to consider the words which are written in abbreviated form or shorthand notation while writing a bug report. These comprises of words such as like→lik, this→dis, be→b etc, which are used very often in chat or while writing emails. A considerable extent of bug reports is filled with such words. Mapping such words systematically with their actual forms and reasonably improve the triaging process.

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. doi: 10.1109/ICSE-COMPANION.2009.5070993
[2] Behl, D.; Handa, S.; Arora, A., "A bug Mining tool to identify and analyze security bugs using Naive Bayes and TF-IDF," Optimization, Reliabilty, and Information Technology (ICROIT), 2014 International Conference on , vol., no., pp.294,299, 6-8 Feb. 2014. doi: 10.1109/ICROIT.2014.6798341
[3] 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. doi: 10.1109/MSR.2013.6624035
[4] 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 doi: 10.1147/sj.462.0289
[5] 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. doi: 10.1287/mnsc.1060.0550
[6] 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
[7] D. Blei, D., Lafferty, J. A correlated topic model of Science. Ann. Appl. Stat., 1, 1 (2007), 17–35.
[8] 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
[9] 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.
[10] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in DSN, 2008.
[11] 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.
[12] 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
[13] 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.
[14] 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.
[15] 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.
[16] 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
[17] J. Sutherland. Business objects in corporate information systems. In ACM Computing Surveys, 2006.
[18] G. Tassey. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology. Planning Report 02-3.2002, 2012.
[19] 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.
[20] 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.
[21] 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.
[22] 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.