An Optimized SVM Model for Accurate Duplicate Bug Report Retrieval

Nawagata Nilambari[1] Shivani Gautam[2] Priyanka Verma[3]

[1,2,3]Department of Computer Engineering

[1,2]Galgotias University, Greater Noida, Utter Pradesh, India [3]Noida Institute of Engineering and Technology Greater Noida, Utter Pradesh, India

Abstract— Bug reporting facility is an integral part of software support system and a critical aspect of beta testing. Managing the incoming bug reports for large software projects is a challenging task. Software testers or end users submit bug reports, as and when potential defects are identified in the system. Sometimes two or more bug reports corresponds to the same defect. The analysis of reports to check whether the incoming report is for the same bug for which a master report already exists in the database is one of the most important task in managing the bug reports. This analysis needs to be done very carefully so that a report corresponding to a bug which is already reported is discarded while at the same time, a textually similar report, but corresponding to a different bug is to be attended properly. To address this problem with duplicate bug reports, a person called a triager needs to manually label these bug reports as duplicates, and link them to their ”master” reports for subsequent maintenance work. However, for large software systems, this process could be highly time consuming. To address this issue, recently, several techniques have be proposed using various similarity based metrics to detect candidate duplicate bug reports for manual verification. Automating triaging has been proved challenging as two reports of the same bug could be written in various ways. There is still much room for improvement in terms of accuracy of duplicate detection process. In this dissertation, an optimized SVM model is proposed to detect duplicate bug reports more accurately. The proposed techniques are validated on three large software bug repositories from Firefox, Mysql, and Apache. Analytical methods are presented over samples and results are provided which are in excellent agreement with the simulation results.

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

I. INTRODUCTION

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

With the increase in size and functionality of the softwares, the possibility of finding out the potential bugs increases. This in-turn, increases the pressure on the triager who needs to manage the overall bug repository to check for duplicates and non duplicates. The research till date involves the feature extraction process of 27 features and 27 from bigrams, which results in a total of 54 features for comparison between two reports to check whether the incoming report belongs to positive or negative examples. The proposed technique takes this feature extraction process a step more advance and consider the similarity measurement based on 30 features and 30 from bigrams resulting in a total of 60 features for similarity measurement and duplicate reports checking.

Duplicate bug report retrieval can be viewed as an application of information retrieval (IR) technique to the domain of software engineering, with the objective of improving productivity of software maintenance. In classical retrieval problem, the user gives a query expressing the information he/she is looking for. The IR system would then return a list of documents relevant to the query. For

duplicate report retrieval problem, the triager receives a new report and inputs it to the duplicate report retrieval system as a query. The system then returns a list of potential duplicate reports. The list should be sorted in a descending order of relevance to the queried bug report. Our approach adopts recent development on discriminative models for information retrieval to retrieve duplicate bug reports. Adapted from [5], in this paper, the duplicate bug report retrieval problem is considered as a binary classification problem, that is, given a new report, the retrieval process is to classify all existing reports into two classes: duplicate and non-duplicate. A total of 60 types of textual similarities between reports and use them as features for training and classification purpose.

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 detailed overview of research methodology. It also makes a background for the concepts and techniques presented in subsequent sections. It presents the proposed F60 model for comparison of an incoming bug report with positive and negative examples. Section III gives the simulation results and the plots for various features for similarity measurement. and concludes the paper.

II. AN OPTIMIZED SVM MODEL APPROACH

A. Automated Checking of Duplicate Reports

To automate the triaging process, two approaches can be used as mentioned below:

(1) To filter out duplicate report automatically from reaching the triager.
(2) To provide a list of similar reports, with each incoming report under investigation.

The latter approach is better in almost all contexts as the duplicate reports need not be useless and keeping these reports linked with the master report is not bad, rather than discarding and deleting. With this approach, the triager has to check from the top most k similar bug reports returned by the system, and if there is a semantically identical report in the set of k reports, then the incoming report is treated as duplicate. The triager then marks it as duplicate and add a link between the two duplicates for subsequent maintenance work.

The duplicate bug reports can complement each other at times if the need arises for a detailed description for bug investigation. This is because most of the times, one report usually does not carry enough information for developers to get the particulars of the defect under consideration.

To achieve automation in an accurate way and thus save triagers' time, it is important to improve the relevance of the ranked list of similar bug reports every time a new bug report is operated. Several studies on retrieving similar bug reports exists in literature. However, the performance of these systems is still relatively low, making it infeasible to apply them in practice. The low performance of existing techniques is partly due to the following factors.

(1) All the techniques currently existing in literature employs one or two features to describe the similarity between bug reports. There are other features which are also available for effective checking of similarity between bug reports and similarity between them can be taken into account for a much more accurate measurement. These features constitute the similarities between title, description and the summery of the report.
(2) Current techniques do not assign optimal weights in the similarity measure of the features. For example, the features capturing similarity between summaries of two reports are much more important in similarity or dissimilarity considerations as summaries of the reports consists of much more comprehensive representation than the descriptions. Therefore, comparatively greater weights should be assigned to similarity (dissimilarity) of the summaries as compared to descriptions. Also, as the project context evolves, a discriminative model should be employed to assign optimal weight assignments to the features. Thus automated techniques should be employed to ensure that the weight assignment to the features will remain optimal at all times as the bug repository expands in time.
(3) The cross similarity checking between any two features can also be considered, in different reports under consideration.

This paper proposes develop a discriminative model to retrieve similar bug reports from a bug repository, based on features capturing similarities between title, description and summaries as well as between any combination of these features.

The contribution of this dissertation is as follows:

(1) A total 60 features are employed to comprehensively evaluate the similarity between two reports.
(2) A discriminative model is proposed based solution to retrieve similar bug reports from a bug tracking system. With the adoption of the discriminative model approach, the relative importance of each feature will be automatically determined by the model through assignment of an optimum weight. Consequently, as bug repository evolves, our discriminative model also evolves to guarantee the all the weights remain optimum at all time. Thus, the proposed model automatically assign optimum weight to each feature and evolve along with the changes in size of bug repositories.

We improve the accuracy of existing automated duplicate bug detection systems by up to 45% on different open-source datasets.

B. Training a Discriminative Model

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 dissertation, a discriminative model is proposed based on 60 features to find the list of top k reports matching with the report under consideration to ease the burden of the triager. A total of 60 types of textual similarities between reports is considered and used as features for training and classification purpose.

Figure 1 shows the framework of the proposed technique.

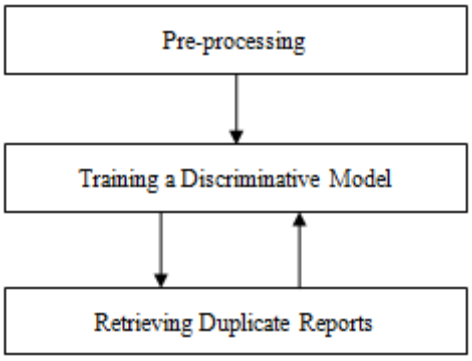

Fig. 1: Proposed Framework

Preprocessing involves a sequence of steps, which includes tokenization, stemming and stop word removal. A word token is a maximum sequence of consecutive characters without any delimiters. A delimiter in turn could be a space, punctuation mark, etc. Tokenization is the process of parsing a character stream into a sequence of word tokens by splitting the stream by the delimiters. Stemming is the process to reduce words to their *ground* forms. The motivation to do so is that different forms of words derived from the same root usually have similar meanings. By stemming, computers can capture this similarity via direct string equivalence. For example, a stemmer can reduce both "tested" and "testing" to "test". The last action is stop word removal. Stop words are those words carrying little helpful information for information retrieval task. These include pronouns such as "it", "he" and "she", link verbs such as "is", "am" and "are", etc.

After stemming, all the reports in the repository are organized into a bucket like structure as illustrated in figure 2

DATA REPOISITORY	
Master 1	Duplicate 1.1, Duplicate
Master 2	Duplicate 2.1, Duplicate
Master 3	Duplicate 3.1, Duplicate
.....	.....
.....	.....
Master n	Duplicate n.1, Duplicate

Fig. 2: Data Structure for Proposed System

Each row of this data structure can be considered as a bucket. Each bucket contains a master report as the key and all the duplicates of the master as its value. Different masters report corresponds to different defects while a master and its duplicates report the same defect. Therefore, each bucket stands for a distinct defect, while all the reports in a bucket correspond to the same defect. The structure of the bucket is shown diagrammatically in Figure 3.2. New reports will also be added to the structure after they are labeled as duplicate or non-duplicate by triagers. If a new report is a duplicate, it will go to the bucket indexed by its master; otherwise, a new bucket will be created to include the new report and it becomes a master.

Given a set of bug reports classified into masters and duplicates, the next step is to build a discriminative model or a classifier that gives a similarity measurement of two input bug reports. This question is essential in our retrieval system. This similarity measure is a probability describing the likelihood of these two reports being duplicate of each other. When a new report comes, each pair between the new report and all the existing reports in the repository is evaluated and then the duplicate reports based on the probability answers are retrieved. A multi-step approach involving example creation, feature extraction, and discriminative model creation via Support Vector Machines (SVMs) is followed.

Training a discriminative model can be described as shown in figure 3

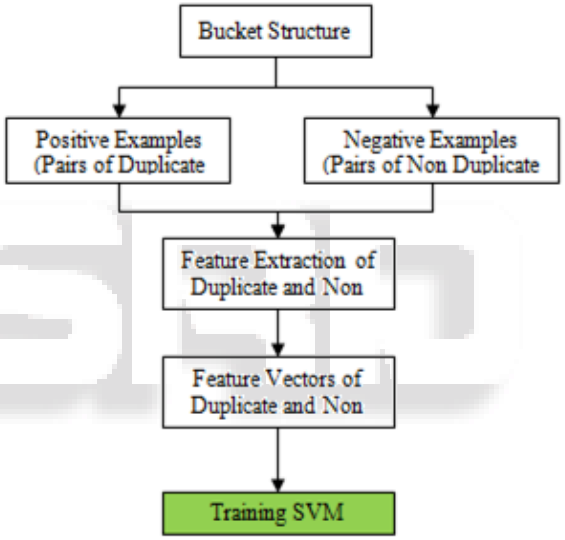

Fig. 3.3: Training a Discriminative Model

Positive Examples consists of the pairs of bug reports that are duplicates of each other while negative examples consists of pairs that are non duplicates. 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.

$$f(R_1 \text{ and } R_2) = sim(words\ from\ R_1, words\ from\ R_2) \quad eq.1$$

The following formula is used as a similarity measurement:

$$sim(B_1 \text{ and } B_2) = \sum_{w \epsilon B_1 \cap B_2} idf(w) \quad eq\ 2$$

sim(B1,B2) returns the similarity between two bags of words B_1 and B_2. The similarity 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.

C. Proposed f-60 model

The proposed f-60 model can be described as shown:

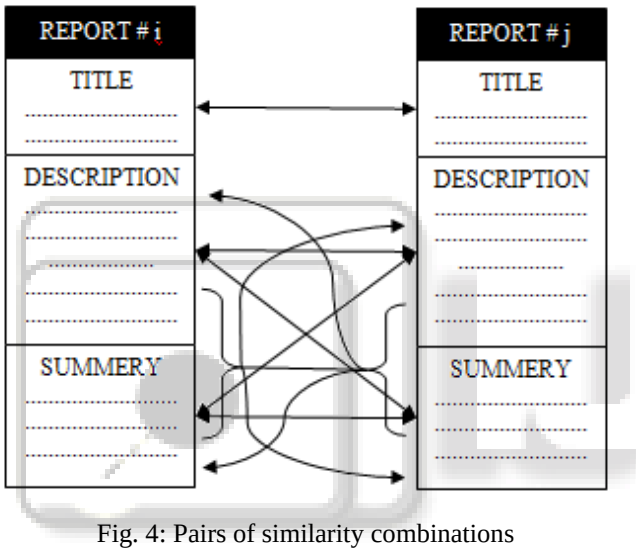

Fig. 4: Pairs of similarity combinations

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) Summery (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.

Libsvm is used to train a discriminative model from the training set. A typical classifier would only give binary answers; in our case, whether two bug reports are duplicate or not. The focus of this work is in knowing how likely two bug reports are duplicates. This is enabled by the probability estimation functionality of libsvm to train a discriminative model which is able to produce a probability of two bug reports being duplicates of each other..

III. Analysis Of Mozilla Firefox Bug Report

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

The preprocessing results for the same are tabulated below:

Title Corpus
Create download Downloaded files getting ??KB [RFE] Add Request: iconify The "What dialogs are Any download Continual Stalled When resuming *.dmg files Firefox refuses Cannot save Javascript timer Clean-up nsDownloadManager Negative (0-12:0-42) download manager Unable to my download Download Manager Download manager Cannot Open Firefox freeze Some larger firefox does Websites do download manager.....
Title Corpus with Stop Word Removal
Create download Downloaded files getting ??KB [RFE] Add Request: iconify "What dialogs Any download Continual Stalled resuming *.dmg files Firefox refuses Cannot save Javascript timer Clean-up nsDownloadManager Negative (0-12:0-42) download manager Unable download Download Manager Download manager Cannot Open Firefox freeze Some larger firefox Websites download manager Download completes Download.....
Description Corpus
Create download proxy for use in embedding. Downloaded files don't get correctly renamed if the /Users folder is in a non-standard location Downloaded files don't get correctly renamed if the /Users folder is in a non-standard location. getting ??KB in status when saving from windows share getting ??KB in status when saving from windows share. [RFE] Add capability to request disconnect upon completion of download.....
Description Corpus with Stop Word Removal
Create download proxy use embedding. Downloaded files don't correctly renamed /Users folder non-standard location

Downloaded files don't correctly renamed /Users folder non-standard location. getting ??KB status saving windows share getting ??KB status saving windows share. [RFE] Add capability request disconnect completion download [RFE] Add capability request disconnect completion download. Request: iconify Firefox tray browsing window closes, downloads pending Request: iconify Firefox tray browsing window closes.....
Summery Corpus
Create download proxy for use in embedding Downloaded files don't get correctly renamed if the /Users folder is in a non-standard location getting ??KB in status when saving from windows share [RFE] Add capability to request disconnect upon completion of download Request: iconify Firefox to system tray when last browsing window closes, but there are downloads pending The "What do you want me to do with this file" download dialog centers itself and is annoying. dialogs are truncated Any download freeze firefox until a large amount of file is downloaded.....
Summery Corpus with Stop Word Removal
Create download proxy use embedding Downloaded files don't correctly renamed /Users folder non-standard location getting ??KB status saving windows share [RFE] Add capability request disconnect completion download Request: iconify Firefox tray browsing window closes, downloads pending "What want file" download dialog centers annoying. dialogs truncated Any download freeze firefox large file downloaded Continual Stalled Loading Any Photo Program I try download Desktop. Irritating resuming download estimated time speed incorrect *.dmg files download Firefox.....
Stemming in Title Corpus
*Application of the Porter Stemming algorithm Used directly through the command stem = porterStemmer(inString); but results in bugs in context of natural language applications. However tf-idf score will not deviate if the same algorithm is applied on the copus.
'download->download' 'Continual->Continu' 'Stalled->Stall' 'resuming->resum' '*.dmg->*.dmg' 'files->file' 'Firefox->Firefox' 'refuses->refus' 'Cannot->Cannot' 'save->save' .....'
Stemming in Description Corpus
'replaced->replac' 'decimal->decim' 'separator->separ'
'Firefox->Firefox' 'download->download' 'manager->manag' 'popup->popup' 'Persian->Persian' 'locale.->locale.'
Stemming in Summery Corpus
'replaced->replac' 'decimal->decim' 'separator->separ' 'Firefox->Firefox' 'download->download' 'manager->manag' 'popup->popup' 'Persian->Persian' 'locale.->locale.'

A. *IDF values for Positive and Negative Examples*

word (w)	idf_d(w)	idf_s(w)	idf_sd(w)
firefox	0.736966	1.321928	0.152003
problem	0.321928	1.736966	0.514573
pdf	0.514573	1.736966	0.514573
filename	0.321928	1.736966	0.736966
extension	0.321928	1.736966	0.152003
save	0.736966	1.321928	0.152003
miss	0.514573	0.514573	0.514573
lost	0.321928	1.321928	0.514573
found	0.514573	0.514573	0.152003
renaming	0.736966	1	0.152003
icon	0.736966	0.736966	0.514573
remove	0.514573	1.321928	0.321928
unknown	0.736966	0.514573	0.321928

Table 1: IDF Values for Positive Examples

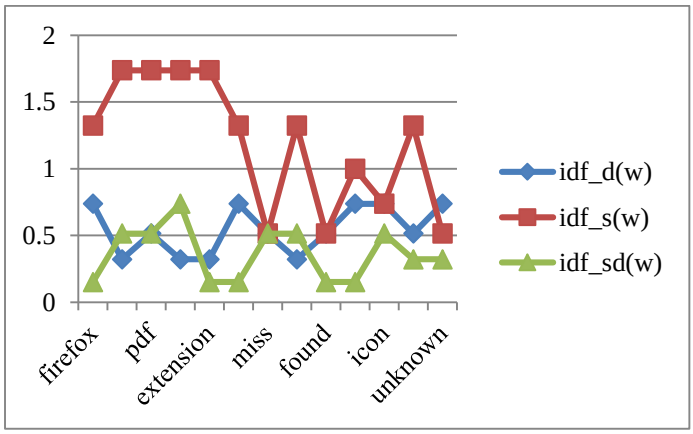

Fig. 5: IDF value Plot corresponding to Table 4.1. idf_d refers to idf values for description, _s for summery and _sd for both summery and description.

For positive reports from the bug repository (Filename : positive_ex.xls, included in CDROM), 10 positive examples are considered and it takes the following form:

word (w)	idf_d(w)	idf_s(w)	idf_sd(w)
unable	1	1	0.152003
delete	0.736966	1.321928	0.514573
download	1.321928	1.321928	0.514573
manager	1.321928	1.321928	0.321928
recognize	0.736966	1.736966	0.152003
special	1.321928	1.321928	0.152003
character	1.321928	1.736966	0.514573
progress	0.736966	1.736966	0.152003
progressbar	0.736966	1.321928	0.321928
appear	1.321928	1.736966	0.152003
open	1.321928	1.321928	0.321928
firefox	1.321928	1.321928	0.321928
freezes	0.736966	1.736966	0.152003
respond	0.736966	1.321928	0.321928
large	0.736966	1.736966	0.514573
filename	1.321928	1.321928	0.321928
extension	0.736966	1.736966	0.321928
save	0.736966	1.736966	0.514573
renaming	1.321928	1.736966	0.321928
download	1.321928	1.321928	0.152003
complete	0.736966	1.736966	0.514573
message	0.736966	1.321928	0.514573
pdf	0.736966	1.321928	0.321928

Table 2: Idf Values For Negative Examples

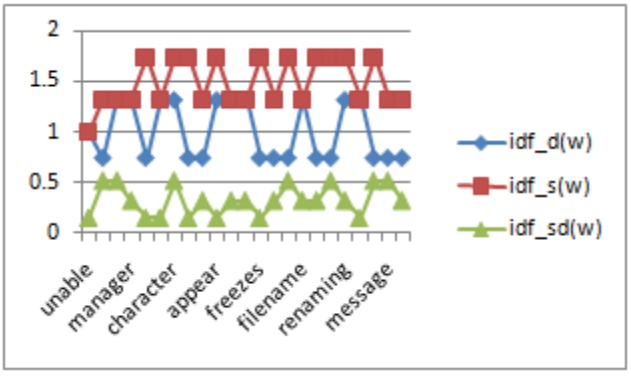

Fig 6 IDF value Plot corresponding to Table 4.2. idf_d refers to idf values for description, _s for summery and _sd for both summery and description.

B. Support Vector Machines

The positive and negative examples can be created using technique as described in chapter 3. The same data is needed to train the SVM to classify bug report as unique or belongs to some class. In this dissertation, the classification that is used is straight and direct. For a given bug report, the SVM can classify weather it belongs to the category of positive examples or negative examples.

Considering the idf for both summery and document, the feature matching process can match 30 different features as described in section II. The corresponding illustrations are depicted as shown.

A total of 100 positive and negative examples, are considered in this simulation. An incoming report is evaluated for similarity score with positive and negative examples. Red dots shows the evaluation values with positive examples and blue shows evaluated values with negative examples. The similarity score of an incoming report, feature-wise is illustrated in the following figures.

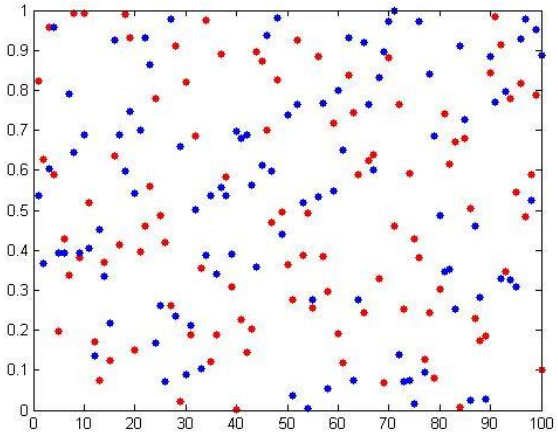

Fig. 7: Similarity score of an incoming report with feature matching of title with title (1 Feature). Vertical axes shows the similarity score. Horizontal axes is equally separated for 100 examples.

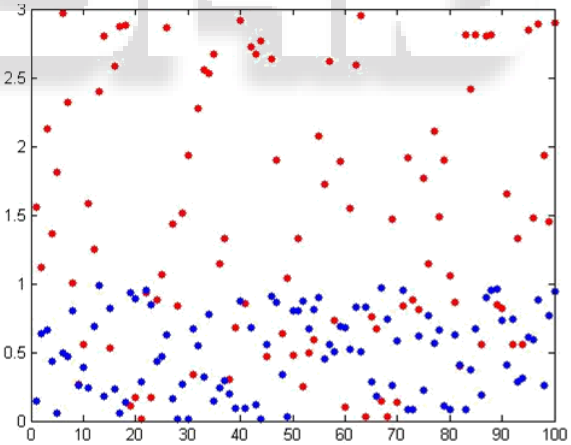

Fig. 8: Similarity score of an incoming report with feature matching of title with title, summery with summery and description with description (3 Feature). Vertical axes shows the similarity score. Horizontal axes is equally separated for 100 examples.

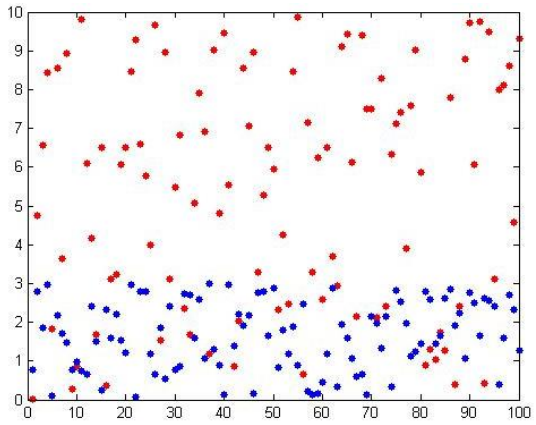

Fig. 8: Similarity score of an incoming report with feature matching of summery, description, and summery+description with all possible likewise pairing (9 Feature). Vertical axes shows the similarity score. Horizontal axes is equally separated for 100 examples.

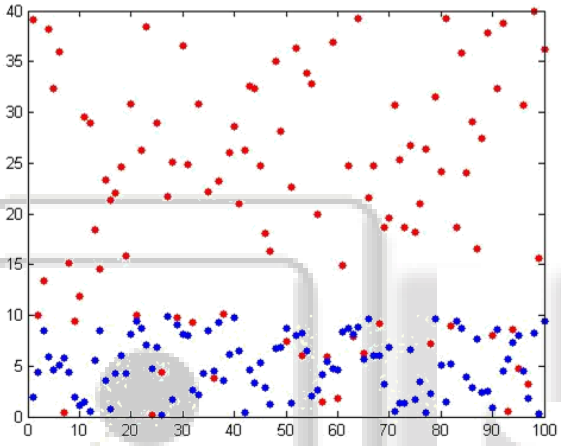

Fig. 9: Similarity score of an incoming report with feature matching of title-summery-description with title-summery-description, also summery+descrition with summery and vice verse (30 Feature). Vertical axes shows the similarity score. Horizontal axes is equally separated for 100 example

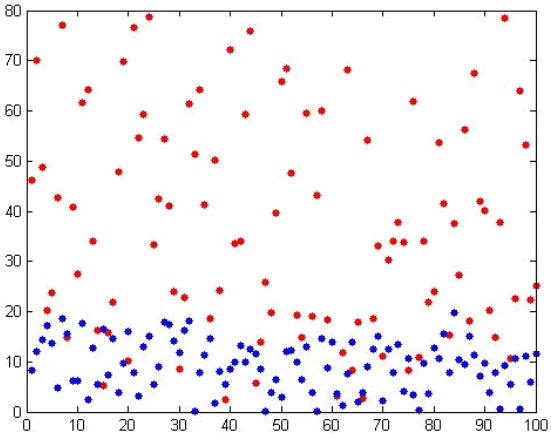

Fig. 10: Similarity score of an incoming report with proposed 60 Feature. Vertical axes shows the similarity score. Horizontal axes is equally separated for 100 examples

The above stats clearly indicates that the more is the number of features, the more better one can classify the given reports as being duplicate or unique.

C. SVM implementation in MATLAB to classify bug reports

The proposed discriminative model approach is applied to three large bug repositories of open source project, Firefox for three different versions. The bug repository is fed into the SVM trainer to train the classifier for the incoming bug report.

For comparison purpose, the stationary (non-discriminative model) is also implemented. To evaluate the performance of different approaches, the notion of *recall rate* is used, which is defined as follows.

$$Recall\ Rate = \frac{N_{detected}}{N_{total}}$$

$N_{detected}$ is the number of duplicate reports whose masters are successfully detected, while N_{total} is the total number of duplicate reports for testing the retrieval process. Given a candidate list size, the *recall rate* can be interpreted as the percentage of duplicates whose masters are successfully retrieved in the list. In terms of this measure, the result shows that proposed approach can bring remarkable improvement over the other approaches.

The SVM classifier implemented in MATLAB executed on Firefox data displays the following results:

Dataset	Version	Training Reports (Duplicate/All)	Testing Reports (Duplicate/All)
Firefox	3.0 beta	100/3112	1237/9125
Firefox	3.5 beta	100/4009	2459/30165
Firefox	3.6 beta	100/1345	1501/29305

Table 4.1: Summery Of Results

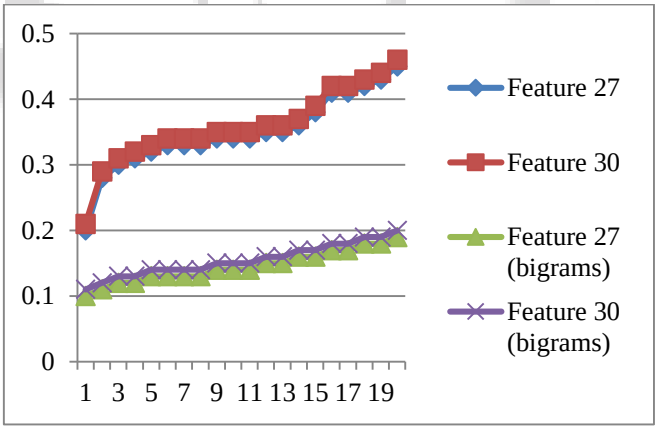

Fig. 11: Plot for Firefox 3.0 beta. Horizontal Axes shows the length of top K lists and vertical axis shows recall rate

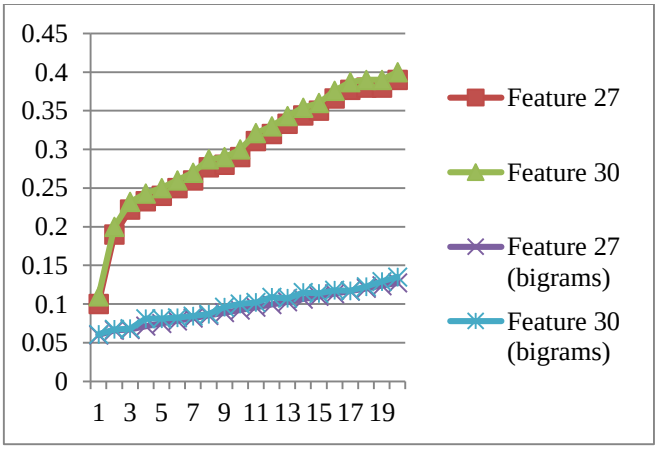

Fig. 12: Plot for Firefox 3.5 beta. Horizontal Axes shows the length of top K lists and vertical axis shows recall rate

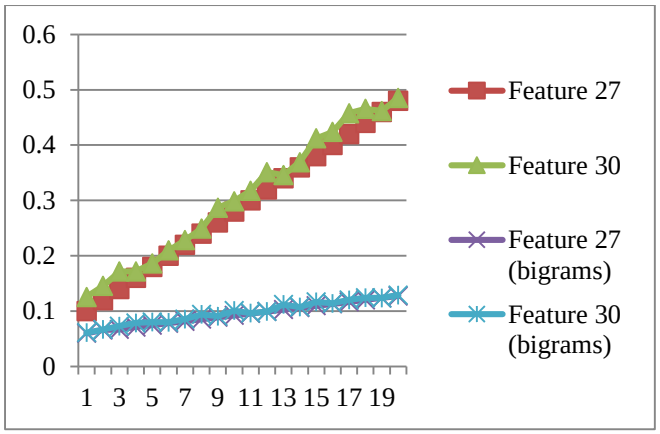

Fig. 13: Plot for Firefox 3.6 beta. Horizontal Axes shows the length of top K lists and vertical axis shows recall rate

The major overhead is due to the fact that 60 different similarity features between reports are considered. In contrast, previous works consider the similarity between summary+description and summary+description, and a few compares the similarity between summary and summary, and the similarity between description and description. The runtime overhead is higher at the later part of the experiment as the number of bug report pairs in the training set is larger. Consequently, SVM will need more time to build a discriminative model.

Neverthless, a higher runtime overhead does not mean that this approach is impractical. In fact, it is acceptable for real world bug triaging process as the datasets which are considered in the simulation model are real world datasets. The dataset from Firefox spans from April 04, 2002 to July 07, 2007 and contains more than 47,000 reports in total. For an active software project, considering reports in one-year frame is enough as bug reports are usually received for new software releases. In Firefox bug repository datasets, the average frequency of new report arrival is 1 report/hour. Therefore, the system still has enough time to retrain the model before processing a new bug report.

IV. Conclusion And Scope

In this work, we consider a new approach to detecting duplicate bug reports by building a discriminative model that answers the question “Are two bug reports duplicates of each other ?”. The model would report a score on the probability of A and B being duplicates. This score is then used to retrieve similar bug reports from a bug report repository for user inspection. The utility of our approach on mozilla bug repositories from 3 different versions is investigated. The experiment shows that our approach outperforms from the existing techniques by a relative improvement of 5–6%. As a future work, The utility of classifiers in machine learning in discriminative models for potential improvement in accuracy will be tackled. A completely different type of preprocessing needs to be done, in contrast to genral natural language, hich 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

[1] Alali, A. ; Kagdi, H. ; Maletic, J.I. " What's a Typical Commit? A Characterization of Open Source Software Repositories", Program Comprehension, 2012. The 16th IEEE International Conference on ICPC 2012.

[2] Yin, X. ; Jiawei Han ; Jiong Yang ; Yu, P.S., " Efficient classification across multiple database relations: a CrossMine approach ", IEEE Transactions on Knowledge and Data Engineering, Volume: 18 , Issue: 6 Year: 2006 , Page(s): 770 - 783.

[3] Yung-Yu Chang ; Zavarsky, P. ; Ruhl, R. ; Lindskog, D., " Trend Analysis of the CVE for Software Vulnerability Management ", IEEE third international conference on social computing (socialcom), 2011 , Page(s): 1290 - 1293.

[4] Bortis, G. ; Van der Hoek, A., " PorchLight: A tag-based approach to bug triaging ", 35th International Conference on Software Engineering (ICSE), 2013 , Page(s): 342 - 351.

[5] Chengnian Sun, David Lo2, Xiaoyin Wang, Jing Jiang2, Siau-Cheng Khoo "A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval", In ASE’13: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pages 34–43, 2013.

