Extended Feature Set Construction for Efficient Triaging of Bug Reports of Open-Source Software

Kulbhushan Bansal[1], Dr. Harish Rohil[2]

[1] *Research Scholar, Deptt. of Computer Science & Engg., Ch. Devi Lal University, Sirsa, Haryana, India*
[2] *Associate Professor, Deptt. of Computer Science & Engg., Ch. Devi Lal University, Sirsa, Haryana, India*
Email: [1]kul_bansal@yahoo.co.in, [2]harishrohil@gmail.com

Abstract: Bug report Triaging is an active area of research in the last few years. This is due to the rapid proliferation of the software and apps which are critically required in the market within a short period, and therefore, released without thorough testing. This is essential as the "time-to-market" has a profound effect on profit margins, whereas software testing is a time-consuming process. However, without thorough testing, software occasionally fails during its working. At this time, the user is prompted to write a bug report, and, in another case, a report is automatically generated and sent to a central database. This report has the details of what happened that cause the software to malfunction. This form of testing, which is done by software users is popularly known as Beta Testing, in contrast to alpha testing, which is done by software testing companies/ departments. A triager is a person or program who reads the bug reports and classifies them related to the bug identified. This classification is a central issue of beta testing and studied in detail by many researchers. Automatic classification involves machine learning over natural language processing. Also, as prior training is required to be delivered to the machine, a training data set is required to be constructed to calibrate the machine. The mathematical models of machine learning classifiers work on feature sets, extracted from raw data. In this research, techniques are presented for the extraction of textual and Contextual features of the reports which lead to the construction of a feature set with 40 features. Support Vector Machine (SVM) is implemented over R package as the machine learning classier and the results are compared with those of the benchmark techniques.

***Keywords*: Triaging, Open-Source Software, Bug, Feature Set**

1. Introduction

Handling a vast number of bug reports sent by a large number of software users need an organized approach.[1] Each bug report is required to be analyzed for checking whether it corresponds to a newly discovered bug or it is a duplicate of an existing report.

As most of the reports are duplicates of each other, possibly related to the same issue/bug, such reports are attached, with one most detailed report as the master report, so that the software developer, at the time of fixing the bug, gets a much clear understanding of what happened at the time of software malfunction. In this paper, the state of art models for extraction of textual and contextual features is explained in detail. These are vector space models which are further extended with extended matching. The BM24 (Binary Measure 25) model [2] used by Sun Microsystems is also investigated with proposed extension techniques.

2. Vector Space Models for Textual Similarity Scores

Tf-idf is an abbreviation commonly used for the term frequency-inverse document frequency. It is one of the most common methods to evaluate how important a word is, in a document in information retrieval or text mining, Tf-idf is a very unique method to change the textual illustration of information into a Vector Space Model, or converting it into sparse textual features. Tf-idf weight is a term mostly used in information retrieval and text mining. The term weight is an arithmetical measure used to compute the importance of a word in a document in a collection. The importance relates substantially to the frequency of a word that appears in the document and it is lowered by the number of times a word occurs in the corpus. Search engines often use a variation of the Tf-idf weighting scheme as an essential tool in scoring and ranking a document's relevance according to the user's query.

The VSM represents the features extracted from the document. The first step for converting a document into a vector space is to prepare a dictionary of terms present in the document. To do this, one can simply choose all terms from the document and change them to the dimension in the vector space. VSM, interpreted in a mathematical sense, is a space in which the text corresponds to a vector of numbers in place of its original string textual representation. [3]

It is to be noted that terms like 'is' and 'the' are ignored, as a consequence of stop word removal. The term frequency is used to represent each term in the vector space, term frequency is a measure of the number of times the term is there in vocabulary E(t). Term-frequency can be defined as a counting function:

$$tf(t,d) = \sum_{x \epsilon d} \quad fr(x,t)$$

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

where $fr(x,t)$ is a simple function defined as:

$$fr(x,t) = \{1\ ,if\ x = t\ \ 0\ ,otherwise$$

The $tf(t,d)$ simply returns how many times is the term t present in document d. This can go on into the formation of the document vector, which is represented by:

$$\overrightarrow{v_{dn}} = (tf(t_1,d_n), tf(t_2,d_n), tf(t_3,d_n) \dots\dots\dots\ tf(t_n,d_n)\ )$$

Since it has a collection of documents at this time represented by vectors, these can be represented by a matrix with |D| X F shape, in which |D| is the cardinality (number of elements) of the document space, and F is the number of features, in the view represented by the vocabulary size.

These matrices representing the term frequencies are likely to be very less in number with a majority of terms zeroed, and that is the reason why a common representation of these matrices uses sparse matrices. The major problem with the term-frequency approach is that it scales down rare terms and scales up frequent terms which are extra instructive than the high-frequency terms.

The tf-idf weight can be used to solve this difficulty. The tf-idf measure gives a score of the importance of a word to a document in the corpus, and this is the reason why tf-idf incorporates local and global parameters. This is because it takes into consideration not just the remote terms but also the terms within the document collection. Tf-idf , effectively, scale down the common terms and at the same time, scale up the rare terms. For example, a term that occurs more than 20 times not necessarily be 20 times more significant.

The inverse document frequency is defined as:

$$idf(t) = log\frac{|D|}{1 + |\{d: t\ \epsilon\ d\}|}$$

In which $|\{d: t\ \epsilon\ d\}|$ is the no. of documents in which term t appears, when the term-frequency function satisfy tf(t,d) ≠ 0, then 1 is supplemented in the formula to circumvent zero-division.

Tf-idf formula is :

$$tf - idf(t) = tf(t,d)\ X\ idf(t)$$

This formula is generally known as the weight of the term. When there is a high term-frequency (tf) present in the document which is the local parameter and a low document frequency of a term in the whole corpus that is a global parameter, a heavyweight of Tf-idf calculation is obtained.

Thus, the matrix for the document vector can be constructed as shown:

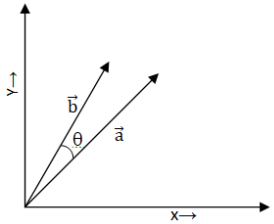
$$M_{train} = [\boldsymbol{tf(t_1,d_1)}\boldsymbol{tf(t_1,d_2)}\ \boldsymbol{tf(t_2,d_1)}\boldsymbol{tf(t_2,d_2)}\ \boldsymbol{tf(t_3,d_1)}\boldsymbol{tf(t_2,d_3)}\ \boldsymbol{tf(t_4,d_1)}\boldsymbol{tf(t_2,d_4)}\ ]\boldsymbol{X}[\boldsymbol{idf(t_1)}\ \mathbf{0\ 0\ 0}\ \ \mathbf{0}\ \boldsymbol{idf(t_2)}\ \mathbf{0\ 0}\ \ \mathbf{0\ 0}\ \boldsymbol{idf(t_3)}\ \mathbf{0}\ \ \mathbf{0\ 0\ 0}\ \boldsymbol{idf(t_4)}\ ]$$

The Cosine Similarity

The cosine similarity between the two vectors or two documents in the Vector Space is an evaluation that computes the cosine angle between them. The particular metric is a dimension of the orientation and not magnitude, and it can also be seen as an assessment between documents on a normalized space because it does not only take into account the scale of each word count of every document but also the angle between the documents. The cosine of the angle between two document vectors can be obtained using the cosine similarity equation of the dot product between two vectors:

$$cos\theta = \frac{\vec{a}.\vec{b}}{|\vec{a}|.|\vec{b}|}$$

Cosine Similarity creates metric shows the relation between the two documents by looking at the angle in place of magnitude, considering the following examples shown below:



Similar scores. Same Direction Vectors. Cosine of angle is 1 i.e.100%

Fig 2.1 Similarity Score: More for Vectors in the same direction

It is important to note that even if there is a vector that is pointing to a point distant from another vector, both can still have a small angle and it is one of the major points on the usage of Cosine Similarity. The dimensions tend to pay no role for similarity score, as given by higher term count on documents. Consider having a document with the word “sky” which is appearing 600 times and some other document having the word “sky” appearing 100, then the Euclidean distance between them will be superior but the angle will still be small since they are pointing to the same direction, and thus have a large similarity score.

3. Classifications of Duplicate and Non-Duplicate Bug Reports

In a duplicate bug report detection system, the incoming bug report is subjected to a detection system which then responds with a list of potential duplicate bug reports related to the input report. The list should be sorted in descending order of relevance to the queried bug report.

In our proposed approach, the generic textual similarity approach is augmented with increased feature set usage and the Latent Dirichlet Allocation approach.

The LDA approach is a topic modeling approach which maps the bug reports to the most relevant topic which may refer to the Non-Functional Attribute (NFA) of the software (like Efficiency, robustness) or might be related to the component of the software (like preferences, history, etc.) Applying information-retrieval (IR) tools, the proposed model takes the advantage of the knowledge of the software process and product. [4,5]

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

The contextual word lists, consisting of NFA and Component names, 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 features of the bug reports, such as description, priority, type, component, etc. proposed in Sun et al. work [2]. 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 report de-duplication performance. The proposed approach is evaluated on a large bug-report data-set from the Mozilla project About 115000 bug reports are analyzed in simulation. In this research, the contextual word lists are considered as components of the Mozilla Browsers (For e.g. Preferences, history, etc.). Our method results in a 16.07% relative improvement in accuracy. This work makes the following contributions.

1. It proposes the use of Topic Modeling 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.

3.1 Proposed Model

3.1.1 Bug Report Format and Corpus for textual features

The bug report analysis structure considered in this paper is specified below:

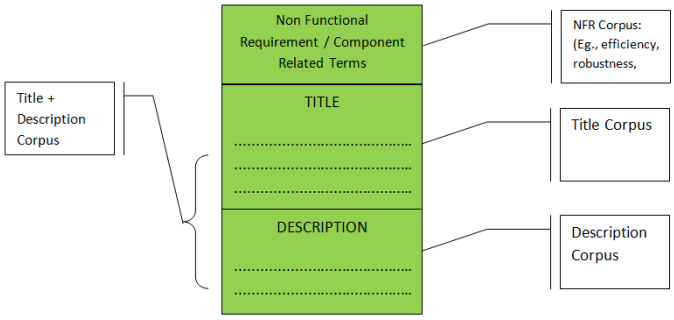


Fig 3.1 Corpus formation for Textual Features

The following formula is used for computing the textual similarity score of any two bug reports.

$$Similarity\ (B_1,\ B_2) = \sum_{w \epsilon B_1 \cap B_2} idf(w)$$

where

$$idf(w) = log_2 \frac{D_{all}}{D_{term}}$$

The idf values are computed based on the corpus formed by the document collection. As specified above, there are 4 different types of corpus considered based on the format of the bug report considered in this dissertation.

3.1.2 Contextual Features

The domain-specific contextual features are computed using Latent Dirchlet Allocation as described. Figure 3.2 shows the framework of the proposed technique.

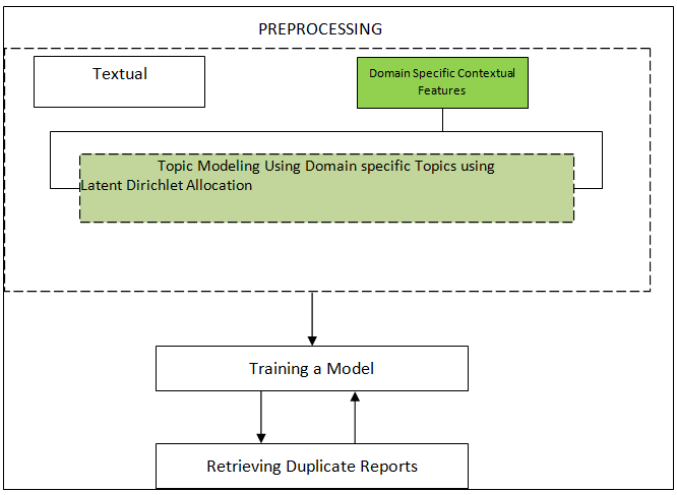


Fig. 3.2 Proposed Framework of LDA Topic Modeling

3.2 Similarity Score based on Textual Features

Considering a generic format of bug reports consisting of LDA Topics which are basically the non-functional requirements of the software, title, and description, the f-60 model can be described as shown:

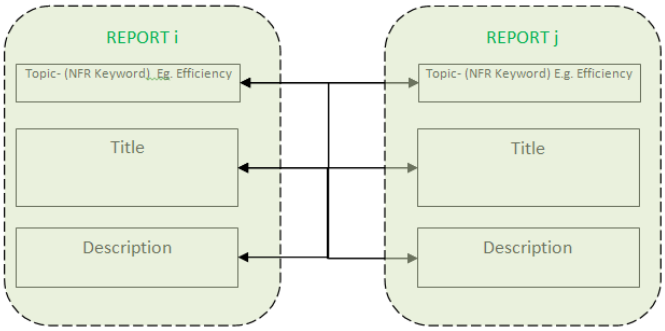


Figure 3.3 Pairs of Textual similarity combinations

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

1. Topic to Topic
2. Title to Title
3. Description to Description
4. Title (R_1) to Description (R_2)
5. Description (R_1) to Title (R_2)
6. Title and Description took together (R_1) - to- Title and Description taken together (R_2).
7. Title and Description took together (R_1) to Title (R_2).
8. Title and Description took together (R_1) to Description (R_2).
9. Title and Description took together (R_2) to Title (R_1).
10. Title and Description took together (R_2) to Description (R_1).

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

Considering each of the combinations as a separate feature, the total number of different features would be 10. Furthermore, one can compute four types of idf, as the bug repository can form three distinct corpora. One corpus is the collection of all the LDA Terms, another is the collection of all Titles, the other is the collection of all the Descriptions and the last one is the collection of all these three corpora taken together.

The four types of idf computed within the four different corpora are defined by idf^{NFR}, idf^{title}, idf^{desc}, and $idf^{combined}$ 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 a 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 10×4, which is equal to 40. 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 idf computed based on summaries, descriptions, or both, there are another 40 features which would then bring the number of features extracted to 80.

3.3 Bug Repository of Mozilla Firefox (Bugzilla Issue Tracker)

Few of the records corresponding to the issues reported to the tracking system of Mozilla Firefox are tabulated in table 4.1 shown below:

TABLE 3.1: SAMPLE RECORDS OF BUG REPOSITORY WITH BUG CATEGORY

Issue_id	Component	Duplicated_issue	Title	Description	Status	Resolution	Version	Created time	Resolved time
10954	Preferences		Dialup properties need to be exposed in prefs	The dialup properties of the profile should be exposed in the prefs panels so; the user has an easy way to modify them. The only other alternative would be to; make people go to the profile manager to edit them, but we don't have that in; place either. Let's try the prefs panel approach first.	RESOLVED	WONTFIX	Trunk	7/30/1999 15:55	5/14/2008 11:44
14871	General	269442	[Find] Find whole word only	Please add Match Whole Word Only option to browsers Find on this page; dialog.	RESOLVED	DUPLICATE	Trunk	9/24/1999 14:49	10/5/2011 16:35
19118	Preferences		Plug-In Manager (ui for choosing mime-type-plugin associations)	I would really like a plug-in manager for my browser that allows me to choose; which mime types are controlled by which plug-ins.; In the browser preferences window there should be a plug-in manager with all; available mime types listed. Under each mime type, there should be a list of; installed plug-ins that can handle that mime type. Radio buttons can be used to; select the plug-in the user prefers for that mime type.; ; For example; ; audio/midi; O Live Audio; O Live Update Crescendo version 4.02; O QuickTime Plug-in 4.0.3; audio/aiff; O Live Audio; O QuickTime Plug-in 4.0.3	RESOLVED	WONTFIX	Trunk	11/17/1999 14:58	1/29/2013 11:48

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

83003	Bookmarks & History		#NAME?	would like to see a command-line option that allows the bookmark window to be; opened on startup; something like mozilla.exe –bookmarks	RESOLVED	WONTFIX	Trunk	5/28/2001 3:16	7/11/2009 20:43
84106	File Handling		[FIX]Not correctly retrieving post data when saving a page or frame generated from a form POST	From Bugzilla Helper:; User-Agent: Mozilla/5.0 (X11; U; Linux 2.4.5-1mdk i586; en-US; rv:0.9+); Gecko/20010604; Build ID: 2001060421; ; Mozilla does not save the respond to a posted form correctly.; Instead of saving the posted reply; it saves the form.; ; Reproducible: Always; Steps to Reproduce:; 1.go to the referenced form; 2.type in a URL in the big text box (say http://www.mozilla.org); 3.push dump links; 4. The form correctly displays the page source; 5. Now try saving the output; 6. You get the form itself and not the response; Actual Results: You get the form output; ; Expected Results: You should get the dumped page. You do get that in Netscape	RESOLVED	FIXED	Trunk	6/4/2001 23:28	3/28/2009 9:39
88541	Bookmarks & History		Show URI in the status bar on mouse-over of Back/Forward menu items	From Bugzilla Helper:; User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:0.9.1+); Gecko/20010627; BuildID: 2001062704; ; When right-clicking on the back button; a list appears of previously visited; pages. However; when mousing over that list; the url for the respective page; should appear in the status bar.; ; Reproducible: Always; Steps to Reproduce:; 1. Load a page; for example http://www.cnn.com; 2. Click one of the article links on the front page; 3. Once the article loads; right-click on the back button; 4. Move the mouse down to the entry for CNN.com Homepage; ; Actual Results: The status bar doesn't change when mousing over the links in; the back-button history.; ; Expected Results: The status bar should show the URL for the link; in this; case: http://www.cnn.com; ; I searched for dups; but I didn't come up with any. And; I wasn't entirely sure; of the correct component.	VERIFIED	FIXED	Trunk	6/30/2001 8:20	4/18/2013 19:46

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

91774	Bookmarks & History		Localization problems with Bookmarks Sorted By menu & History Sorted By menu.	Localization problems with Bookmarks Sorted By menu & History Sorted By menu.; Sorted by menu gets strings from headers - wrong (can't localize it in proper; way). It should have its strings for example in mail client.; ; Strings are from:; http://lxr.mozilla.org/seamonkey/source/xpfe/components/bookmarks/resources/locale/en-US/bookmarks.dtd#68; to #74; ; Instead of gettings labels from headers add labels like this:; sorted by.name.label; sorted by.url.label; etc.	RESOLVED	FIXED	Trunk	7/21/2001 15:47	11/26/2009 6:04
92073	Preferences		UI to allow external handlers for internal types	For those who cannot update QuickTime to 5.0.2 to solve bug 69719; and similar cases we need options (in Preferences dialog) to set; individual mime-types (images...) to be processed by Mozilla; internally even if they are associated with another application	RESOLVED	WONTFIX	Trunk	7/24/2001 4:29	5/14/2008 11:46

3.4 Creating Positive and Negative Examples

To train the SVM-based classier, it is important to create a set of classified data so that the machine can be trained over the manually classified data. From the term-document-matrix obtained from the Mozilla bug repository, the sample of which is shown above, we can match each record with the other documents of the corpus to obtain a set of master and duplicate bug reports as shown in the following figure.

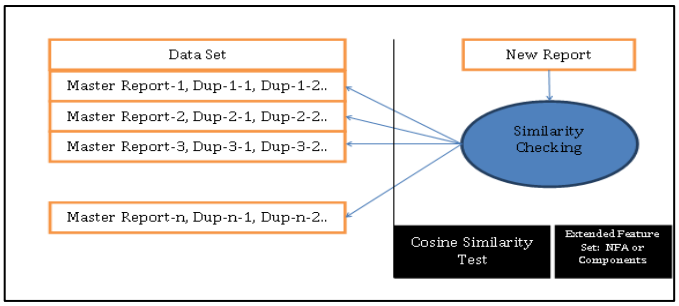


Figure 3.2 Preprocessing for creation of Positive and Negative Examples

To train the SVM, it is needed to create a set of positive and negative examples which are manually classified on all the parameters considered in this paper; namely, the textual, and contextual features. A stated previously, the positive examples consist of all the bug reports identical to each other; called duplicates and the negative examples consist of all the bug reports corresponding to different bugs; called non-duplicates.

4. Results and Analysis

The RPlot showing the most common terms in the corpus, on a scale of size, following their relative frequencies, is shown in figure 4.1

Figure 4.1. Word Cloud of most frequent terms in the corpus

The clustering of Bug Reports of Mozilla Firefox, as per the cosine similarity index is tabulated using R. It is shown in figure 4.1.

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

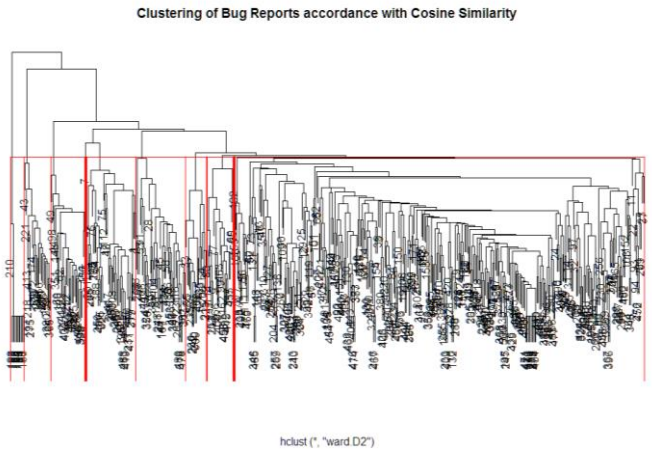


Fig 4.1 Clustering of Bug Reports following the cosine similarity index

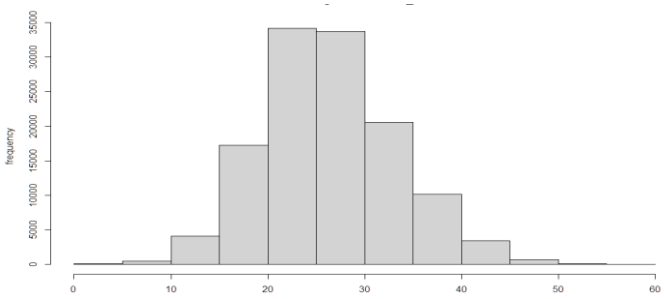


Fig 4.2 Histogram of Euclidian Separation between Bug Reports

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 80 textual features. The contribution of this work is to augment the extended feature set of textual and categorical feature sets with contextual features obtained from topic modeling using Latent Dirichlet Allocation. The feature set for classification of duplicates and non-duplicates, in terms of contextual measures, in the bug repository can be done by the creation of positive and negative examples. The positive and negative examples are created from the Bugzilla repository which consists of a total of 115814 records.

Support Vector Machine is trained by creating a 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.

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

4.2 Performance Measures

The Precision Rate of the Trained SVM of the entire set of bug reports is depicted in figure 4.10. The Recall rate of the system is defined as a 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 the effectiveness of the duplicity 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 the categories of bug reports. The recall curve of the proposed scheme for topic modeling is depicted in figure 4.5 below.

It turns out that the proposed scheme with 80 textual features and LDA topic modeling results in about 15 percent improvement in the number of similar bug reports detected from the repository, as compared to only the domain-specific topic modeling proposed by Anahita Alipour et.al.[7]

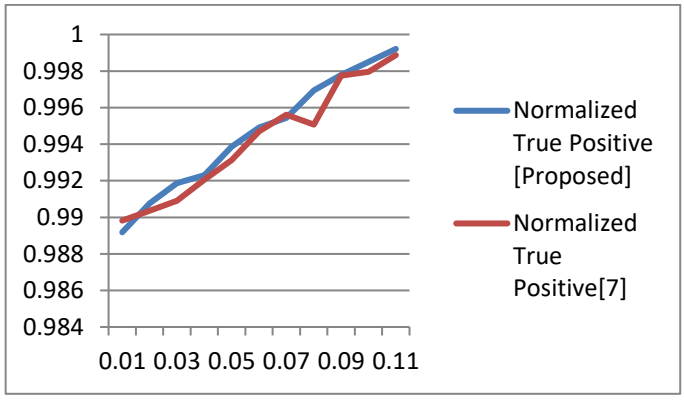


Fig 4.5 The Precision Rate of the Trained SVM. Horizontal axes show the normalized False Positive Rate and the vertical axes show the normalized True Positive Rate.

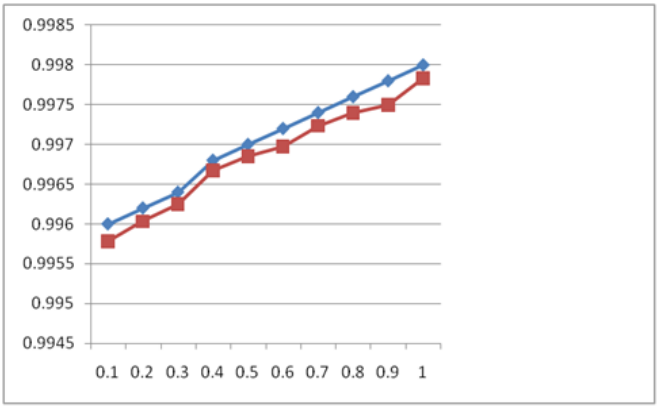


Fig. 4.6 Precision versus Recall measure of SVM

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

5. Conclusions

A system is proposed that automatically classifies duplicate bug reports as they arrive thereby saving the 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 the classification of duplicated bug reports. This is done by augmenting the feature set with the contextual features which relate 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 to map a document to certain topics through Dirichlet Distribution. Results are empirically evaluated using the R statistical package. A dataset of 10,000 bug reports is analyzed from the Bugzilla project.

The proposed system can reduce development costs by filtering out duplicate bug reports. It also gives much more precision in the case of comparatively small software packages having a 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.

As a future Scope of this work, a more accurate bug report triaging system can only be prepared with domain-specific knowledge of the software. Such a system cannot achieve a substantial recall rate and precision if it is modeled generically. As a future perspective 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 result 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 specific to the memory overrun error of the 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 comprise 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 improves 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]Hindle, A., Alipour, A. & Stroulia, E. A contextual approach towards more accurate duplicate bug report detection and ranking. Empir Software Eng 21, 368–410 (2016). https://doi.org/10.1007/s10664-015-9387-3

[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] Anahita Alipour, Abram Hindle, and Eleni Stroulia. 2013. A contextual approach towards more accurate duplicate bug report detection. In Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13). IEEE Press, Piscataway, NJ, USA, 183-192.

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731