IJRSET JANUARY 2017 Volume 4, Issue 1 Pages: 1-16

ISSN 2394-739X

International Journal for Research in Science Engineering and Technology

IMPROVED SUPPORT VECTOR BASED RANKING MODEL AND FEATURE SELECTION METHODS FOR BUG REPORTS ANALYSIS

[1] K. AARTHI PRIYA, [2] S. VETRIVEL, [3] N. NITHYA
[1] Assistant Professor, [2] Assistant Professor, [3] Assistant Professor
[1] Department of Computer Science, [2] Department of Information Technology, [2] Department of Business Administration & Computer Applications,
[1, 2, 3] AJK College of Arts & Science, CBE.

ABSTRACT- Bug report tracking systems have been used toward make possible the maintenance and development of software. On the other hand, duplicate entries presented in the bug reports in such software system are able to significantly force productivity inside software project. This reduction in productivity happens since duplicate entries demand more time designed for search and examination of bug reports. When a new bug report is received, developers frequently require to reproducing the bug and performing code reviews to discover the cause, a process with the purpose of can be difficult and time consuming. To solve this problem in this paper, Support Vector Machine (SVM) system is proposed that considers features with the purpose of association related to lexical gap via the use of project precise Application Programming Interface (API) report to attach Natural Language parameters in the bug report by means of programming language with the purpose of build in the source code. The major contribution of this paper is to propose Support Vector Machine (SVM) ranking method which solves bug report problem related to source files with the purpose of permits the faultless integration of a different type of features. Support Vector Machine (SVM) ranking approach counts the frequency value toward each bug reports make use of formerly fixed bug reports as training examples designed for the proposed ranking-model in combination by means of a learning-to-rank technique; with the file dependency graph toward describe features with the purpose of confine a determine of code complexity. The extensive experimentation and comparisons is done to traditional ranking methods, it concludes that the proposed FOFR system of the impact with the purpose of features producing higher ranking accuracy and lesser code complexity.

Keywords: [Support Vector Machine (SVM), Application Programming Interface (API), Bug report, tracking systems, feature evaluation.]

1. INTRODUCTION

In general software bug or defect is a coding error with the purpose of might basis unintentional or unpredicted activities of the software system [1]. Leading discovering abnormal activities of the software system, a developer or a software user determination report it in a document, named a bug reports. Bug report gives information with the purpose of might help in fitting a bug by

means of the overall objective of enhancing the quality of software system. A huge number of bug reports might be opened throughout the improvement life-cycle of a software system. Example considers there are 3,389 bug reports have been generated from Eclipse product in 2013 by you. In a software group, bug reports are expansively used by means of together managers and developers in their daily software development procedure [2]. A developer who is allocating a bug report frequently requires replicating the abnormal activities [3] and performing code evaluation [4] in order to discover the faults. On the other hand, the assortment and irregular quality of bug reports be able to formulate this procedure nontrivial. Fundamental information is frequently missing from a bug report [5]. Bacchelli and Bird [3] reviewed the software quality product with 165 managers, 873 programmers, and described with the purpose of detecting defects needs a higher level understanding of the program and knowledge by means of the appropriate source code files. From the survey it concludes that the 798 respondents answered with the purpose of it takes long time to analysis unknown files. If the number of source files in a software project or prototype is generally huge, the number of files with the purpose of include the bug is generally extremely small. As a result conclude that an automatic approach with the purpose of ranked the source files with related to their relevance designed for the bug report might speed up the bug detecting step via lessening the investigate to a smaller number of probably unfamiliar files. If the bug report is understudied via user given query and the source files in the software system are examined as a collection of documents, then the problem of discovering source files with the purpose is appropriate for a known bug report could be represented as a normal process in Information Retrieval (IR) [6]. For this purpose, proposed a new ranking model to solve ranking problem, here source files are ranked with respect to their relevance to a known bug report. In this framework, relevance is associated via the use of likelihood with the purpose of a specific source file consists of the source of the bug discussed in the bug report. This ranking model, ranking function is represented as the weighted combination of attributes, where the attributes described greatly on information precise to the software engineering area in order to calculate appropriate relationships among the bug report and the source code file. While a bug report might divide textual tokens by means of its appropriate source files, in common there is a considerable intrinsic mismatch among the natural language working in the bug report and the programming language used in the source code [7]. Some of the ranking methods in the literature is performed based on the lexical based matching scores value with suboptimal performance, in measurement appropriate to lexical mismatches among natural language statements in bug reports and technological terms in software model. The software systems consists of many attributes with the purpose of association the related lexical gap by means of using project particular API documentation in the direction of connect natural language terms in the bug report by means of programming language creates in the source code. Moreover, source code files should consist of a many number of methods of which simply a small number might be sourcing the bug. In the same way, the source code is syntactically parsed into methods and the attributes are designed to make use of method level measures of significance designed for a bug report. It has been formerly experimental with the purpose of software procedure metrics are more significant when compared to code metrics in software defect detection process [8]. Accordingly make use of the change history related to source code as a well-built report for linking fault-prone files by means of bug reports. One more helpful domain specific examination is with the purpose of a buggy source file might origin more than one abnormal activities, and consequently might be dependable designed for comparable bug

reports. If associate a bug report by means of a user and a source code file among an item with the purpose of the user might like or not, subsequently illustrate an analogy through recommender systems [9] and make use of the model of collaborative filtering. Consequently, if formerly fixed bug reports are textually related by means of the current bug report, then files with the purpose of have been related through the related reports might moreover be appropriate for the current report. In order to improve the accuracy of change management processes, some of the organization should use domain specific systems usually named as bug-trackers toward deal with accumulate and hold change needs moreover called as bug reports. A bug report is defined as a software object with the purpose of defines some defect, development, alteration request with the purpose of is submitted toward a bug tracker; usually, bug report submitters are developers, users, or testers. Those types of software systems are helpful since changes toward be completed in software be able to be rapidly acknowledged and presented toward the suitable people [10]. Examples of those type of software bug report systems are Bugzilla (http://www.bugzilla.org), Mantis (http: //www.mantisbt.org) and Trac (http://trac.edgewall.org). These software bug report systems , bug reports is stored by means of a diversity of fields of free text and custom fields described related to the requirements of each project. In Trac, for instance it is described as the methods and their fields designed for review and prescribed details of a bug report. In the same bug report it is able to moreover be recorded information related to the software version, dependencies among other bug reports that find the duplicate bug reports, for instance, and the person who determination is allocated to the bug report, between other information. However throughout the development of software life cycle model of a bug report and their related comments have been integrated into programming model to help to solving it. Additional challenges have presented by the use of bug trackers between them those issues are dynamic allocation of bug reports [11], change impact evaluation and effort evaluation [12], software quality of bug report definition [13], software analysis and traceability [14], and detection of duplicate bug reports. To solve all of these problems in this paper presents a new feature ranking model that finds bugs based on their count value of each feature in the bug report .The major contribution of the research work is described as follows, Support Vector Machine (SVM) system considers features with the purpose of association related to lexical gap via the use of project precise Application Programming Interface (API) report to attach Natural Language parameters in the bug report by means of programming language with the purpose of build in the source code. The major contribution of this paper is to propose Support Vector Machine (SVM) ranking method which solves bug report problem related to source files with the purpose of permits the faultless integration of a different type of features. Support Vector Machine (SVM) ranking approach counts the frequency value toward each bug reports make use of formerly fixed bug reports as training examples designed for the proposed ranking-model in combination by means of a learning-to-rank technique; with the file dependency graph toward describe features with the purpose of confine a determine of code complexity.

2. LITERATURE REVIEW

There is a multiplicity of work correlated toward mining bug report repositories. Conversely, the work establish scheduled the literature is relatively new. In common, these types of repositories encompass been mined used for dissimilar purposes, such as: bug reports parallel , dynamic assignment of bug reports, software evolution as well as traceability, modify impact analysis along with effort estimation, furthermore value of bug report descriptions. All of these proposed categories encompass regular objective toward develop software development, saving costs through software

maintenance. After that, discuss about every work associated toward the mentioned purposes. Furthermore, it will exist specified more attention on the way to process details involving work related toward duplicate bug reports detection, because this work as well addresses such purpose. Duplicate bug reports discovery consists taking place thorough used for past bug reports toward identify similar bug reports so as to illustrate the similar issue at the same time as the one being reported, during order toward avoid duplicate submission. In that way, the following work usually proposes methods on the way to aid such detection. First toward investigate bug reports similarity [15] explore during their occupation were software failures mechanically submitted as soon as the software do not work properly. Such reports were collected of information (profile) regarding state of the software by the side of the time failure occurred, as well as possibly through execution stack trace. This category of information is able to raise a general problem encounter by developers: they take delivery of additional reports than the times they encompass available toward investigate them. Thus the proposed an automated preserve for classify these information during organize toward prioritize as well as diagnostic their causes. The work perform within is closer toward ours than the first individual described [16]. It investigated the duplication problem cause through ordinary language bug reports submission. The proposed toward grouping related bug reports into centroids, consequently it would exist possible on the way to evaluate incoming bug reports toward centroid through high similarity. In this way, each bug report was processed toward compute value used for Term Frequency-Inverse Document Frequency (TF-IDF) [3] along with placed during centroid through higher similarity. The TF-IDF used for a single centroid was combination of the TF-IDF[3] of every one bug reports within same centroid. The effort reports results used for different thresholds, as well as thresholds for classification along with for the length of recommendation list. The move toward achieve 29% of precision as well as 50% of recall by its best. The most important difference from this work as well as our approach is so as to technique used during tool proposed doing not group the majority bug reports into centroids. In addition, our instrument is not a recommendation organization; users encompass toward perform investigate during order toward find similar bug reports. The recent work [17] addressed the problem of detect duplicate bug reports with Natural Language Processing (NLP) techniques. One advantage of such work was identification of two types bug reports: 1) individuals so as to explain same problem along with 2) individuals to facilitate explain two different problems through the same cause. The former describe equal failure, normally use related vocabulary, as well as the latter describe special failures along with may well use a different vocabulary. In the approach [18] toward moderate bug reports duplicate problem use NLP as well as execution information. Implementation information is disturbed toward information concerning the software execution while error occurred, such because technique calls otherwise variables state. This type of data was combined by means of natural language data toward get better recall. In order toward validate approach it was perform an experiment through Firefox as well as Eclipse data, resultant during a recall of 67%-93% next to its best. These outcomes are very suitable if compare through additional related work. Though, several points should exist outline toward experiment as well as the approach itself. Also known because Bug report Triage, this step of bug description tracking procedure consists of identify which be best developer toward explain a new bug report. The work [19-20] as well proposed a method toward bug report assignment, though during such work it was use information as of versioning systems mutual through bug reports information. In additional work [21], it was compare which category of repository is improved toward allocate best developer toward a bug report. The work concluded so

as to use bug description repositories is better if the objective is toward establish expertise group during less false positives, whilst versioning systems are better used for retrieving each and every one experts used for a specified problem. Mining bug repositories used for software evolution as well as traceability is concerned through under-standing what drives changes perform within software along the time. Software traceability regularly involve documents, supply code, bug reports, amongst extra assets. Usually, research related during this purpose combine data from source code repositories furthermore bug report repositories. Sandusky et al [22] conduct an empirical research regarding Bug Report Networks (BRNs) during open source projects. According toward them, a BRN is produced while member of a software growth project asserts duplication, dependency, otherwise reference relations amongst bug reports. They pointed so as to BRNs considerate preserve use decrease cognitive furthermore organizational effort, developed representation of software as well as work-organization issues, and rearrange relationships amongst project members. Antoniol et al [23] planned a framework toward merge information on or after bug description repositories, source code, as well as versioning systems. Such framework aids developer toward browser along with navigates during information provide through such artifacts inside interconnected way. For example, many developer probably make use of framework toward visualize what bug reports were fixed at the time of known software version. Moreover, he/she might well imagine what files of source code were altered or changed. Koponen and Lintula [24] introduce a new software bug report prediction methods toward incorporate versioning systems and bug report database from history files. It used information from Apache HTTP Server and Firefox. They also analysis the changes in software projects were driven by means of bug reports. From the results it concludes that simply a less percentage of changes were complete since of bug reports in Apache HTTP Server. On the other hand, in Firefox 60% of alters are guided by means of bug reports. Furthermore, they revealed with the purpose of developers who performed little changes are further appropriate toward be showed by means of bug reports. In recent times, researchers have introduces and develops a new bug report prediction methods with the purpose of give attention to ranking source files designed for known bug reports repeatedly [25]. Saha et al [25] syntactically parse the source code into four major categories are class, method, variable, and comment. The review and the explanation of a bug report are measured as two doubt fields. Textual similarities are calculated for each of the eight document type in the direction of query field pairs and then summed up addicted to a final ranking function. Zhou et al [26] also performs bug report prediction model by considering not only lexical similarity among a new bug report and every source file however also provide more weight in the direction of larger size files and files with the purpose of have been fixed earlier than for similar bug reports. This Bug Locator relying on single parameter and performed based on three different features. The parameter is tuned by using these three different features and it is used for experimentation, which means with the purpose of the results reported in their work related to performance of the software system. It is consequently uncertain how well their Bug Locator simplifies in the direction of unobserved bug reports.

3. PROPOSED METHODOLOGY

Support Vector Machine (SVM) system considers features with the purpose of association related to lexical gap via the use of project precise Application Programming Interface (API) report to attach Natural Language parameters in the bug report by means of programming language with the purpose of build in the source code. The major contribution of this paper is to propose SVM ranking method which solves bug report problem related to source files with the

purpose of permits the faultless integration of a different type of features. SVM ranking approach counts the frequency value toward each bug reports make use of formerly fixed bug reports as training examples designed for the proposed ranking-model in combination by means of a learning-to-rank technique; with the file dependency graph toward describe features with the purpose of confine a determine of code complexity. In the experimentation work wide-ranging evaluation and comparisons is done by comparing the results to state-of-the-art methods, it concludes that the proposed Support Vector Machine (SVM) system have achieves higher ranking accuracy. The proposed Support Vector Machine (SVM) system is trained to determine a matching score designed for any bug report 'r' and source code file 's' grouping. The ranking score function f(r,s) is described as the weighted value by means of sum of k features, where every feature $\phi_i(r,s)$) evaluates the a precise relationship among the source file s and the bug report r:

$$f(r,s) = w_T \Phi(r,s) = \sum_{i=1}^{K} w_i * \phi_i(r,s) \quad (1)$$

Known bug report r is given as input for evaluation during testing and training time .The FOFR system determines the score f(r,s) for each source code s in the software project and make use of this score value to rank all the codes in either ascending or descending order. The user is then obtainable by means of a ranked list of files, among the expectation with the purpose of files emerging advanced in the list are more possible toward be appropriate for the bug report, i.e., further expected to include the source of the bug. In this FOFR system tries to determine a set of parameters designed for which the rank scoring function ranks all source files with the purpose of known to be appropriate for a bug report on the top of the list designed for with the purpose of bug report.

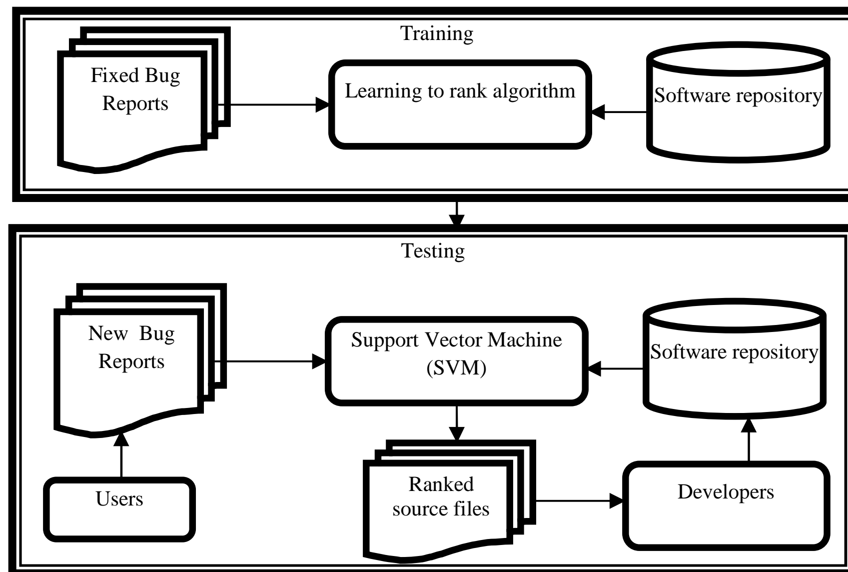

Figure 1- System architecture for training and testing

4. FEATURE REPRESENTATION

The proposed FOFR system needs a bug report - source file pair (r, s) have been denoted as a vector of k features $\Phi(r,s) = [\phi_i(r,s)]_{1 \le i \le k}$. The entire set of twenty features used in the FOFR system is discussed in the following section. These entire features in the bug report have been categorized into two major user defined query type those are described as follows:

Query dependent: This type of features $\phi_i(r, s)$with the purpose of rely on both the bug report r and the source code file s. A query dependent feature denotes a specific association among the bug report and the source file, and thus might be helpful in formative straightforwardly whether the source code file s consists a bug with the purpose of that is appropriate for the bug report r.

Query independent: This type of features relying on only the source code file, i.e., their computation does not need information of the bug report query. This type of feature might be used to determine the likelihood with the purpose of a source code file consists of a bug, irrespective of the bug report. Make use of Vector Space Model (VSM) for ranking purpose. This VSM, both the query and the document are denoted as vectors of term weights. For known bug report or a source code file report d calculate the term weights $w_{t,}$ for each term t in the vocabulary depending on the general tf.idf weighting scheme in which the term frequency factors are regularized is described as follows:

$$w_{t,d} = nf_{t,d} \times idf_t \quad (2)$$

$$nf_{t,d} = 0.5 + \frac{0.5 \times tf_{t,d}}{\max_{t \in d} tf_{t,d}} \quad idf_t = \log \frac{N}{df_t} \quad (3)$$

The term frequency factor $tf_{t,}$ is defined as the number of occurrences of each term t in bug report document d, whereas the document frequency factor df_t is defied the number of bug report documents in the bug report history repository with the purpose of includes the term t. N is to the total amount of documents in the bug report history repository, whereas idf_t is denoted as the inverse document frequency, which is determined via the use of logarithm in order to reduce the result of the document frequency factor in the entire term weight.

Surface Lexical Similarity

For a source code file, make use of its entire code and comments. In order to tokenize an input history bug report document initially divide the text document into a bag of words by using white spaces. From this results then remove punctuation, numbers, and stop a word that is conjunctions or determiners. The bag of words illustration of the bug report document is then improved by means of the resulting tokens “Work” and “Bench” in this example even as also maintenance the unique word as a token. Lastly, each and every one word is concentrated toward their stem by means of the Porter stemmer, as experimented in the NLTK package. This procedure will decrease derivationally similar words are “programming” and “programs” to the same stem “program”, which is well-known to contain a helpful impact on the performance of the final system. Let V be the vocabulary of each and every one text tokens show in bug reports and source code files.

Let $r = w_{t,r} \in V]$ and $s = w_{t,s} \in V]$ is denoted as the VSM vector from the bug report r and the source code file s, where the term weights $w_{t,r}$and $w_{t,s}$ are calculated by using the tf.idf formula given in Equation (2). Once these values are calculated then textual similarity among a source code file and a bug report have been calculated by using cosine similarity among their related vectors:

$$Sim(r, s) = \cos(r, s) = \frac{r^T s}{||r|| \, ||s||} \quad (4)$$

In VSM, cosine similarity have been applied straightly and used as feature for ranking score evaluation. But cosine similarity measure ignore the fact with the purpose of bugs are frequently restricted in a little part of the code. If the source code becomes very large, it related cosine similarity norm determination will also very large, which results lesser cosine similarity value to each bug report. To overcome these problems this work uses a new AST parser directly from the Eclipse JDT tool and split the source code

into many methods in order to compute cosine similarity to bug report. From this AST parser, each method m is considered as individual document and determines its lexical similarity to each bug report via the use of cosine similarity function. After-that finally determine surface lexical similarity by using the following formula is described as follows,

$$\phi_1(r, s) = max(\{sim(r, s)\} \cup \{sim(r, m) | m \in s\}) \quad (5)$$

From the greatest value of each and everyone per-method similarities, then complete file similarity. This is clearly under the category of query-dependent feature.

API-Enriched Lexical Similarity

In wide-ranging, many of the text in a bug report is described in normal English language, while most of the substance of a source code file is written in a Java programming language. Consequently used for every method in a source code file mine a set of class and boundary names from the explicit category declarations of each and every one local variables. By means of the project API condition, find the textual definitions of classes and interfaces, consists of the definitions of each and every one their direct or indirect super-classes. For each method m generate a document m.api by means of focus the related API definitions. At lastly, take the API definitions of each and every one method in the source file s and concatenate them into an entire document $s.api = \cup_{m \in s} [m.api]$ then calculate an API lexical similarity feature is described as follows,

$$\phi_2(r, s) = max(\{sim(r, s.api)\} \cup \{sim(r, m.api) | m \in s\}) \quad (6)$$

From the greatest value of each and everyone per-method similarities, then complete file API similarity. This is clearly under the category of query-dependent feature.

Collaborative Filtering (CF) Score

CF have been used earlier in other domains to enhance the accuracy of software recommender systems, therefore it is predictable to be helpful in information retrieval situation, too. Known a bug report r and a source code file s, consider $br(r, s)$ be the group of bug reports designed for which file s was predefined earlier r was bug reported. The CF feature is then computed as follows,

$$\phi_3(r, s) = sim(r, br(r, s)) \quad (7)$$

The feature calculates the textual similarity among the text of the present bug report r and the summaries of each and every one the bug reports in $br(r, s)$. This feature is also under the category of query-dependent.

Class Name Similarity

A bug report should straightforwardly state a class name in the review, which gives a useful information with the purpose of related source file implementing with the purpose of class might be appropriate for the bug report. For instance, the summary of the Eclipse bug report 409274 consists of the class names Workbench Window, Workbench, and Window after tokenization, however simply WorkbenchWindow.java is a relevant file. Let s:class is represented as the name of the major class experimented in source file s, and |s:class| the name length. From the observation compute a class name similarity feature is described as follows:

$$\phi_4(r, s) = \begin{cases} |s.class| & \text{if } s.class \in r \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

This feature is also under the category of query-dependent and computed automatically by using feature scaling step.

Bug-Fixing Recency

The source code change history gives the data with the purpose is able to help and find fault-prone files [27]. For instance, a source code file with the purpose was fixed extremely newly is more probable toward still consists of bugs than a file with the

purpose of was last fixed long time in the history, or never fixed. let $br(r,s)$ be the set of bug reports for which file s. Let $last(r,s)$ be the most new formerly fixed bug. Furthermore designed for some bug report r, permit r:month indicate the month when the bug report was generated. Then describe the bugfixing recency feature in the direction of the converse of the distance in months among r and $last(r,s)$

$$\phi_5(r,s) = (r.month - last(r,s).month + 1)^{-1} \quad (9)$$

Consequently, if s was proceeding fixed in the similar month with the purpose of r was created, $\phi_5(r,s)$ is 1. If s was last fixed one month previous to r was generated, $\phi_5(r,s)$ is 0.5.

Bug-Fixing Frequency

A source file with the purpose of has been regularly fixed might be a faultprone file. Accordingly describe a bug-fixing frequency feature from the present bug report:

$$\phi_6(r,s) = |br(r,s)| \quad (10)$$

This feature is also under the category of query-dependent and computed automatically by using feature scaling step. Neither of the two features $\phi_5(r,s)$ or $\phi_6(r,s)$ depending on the content of the bug report r. On the other hand, their computation still relies on the time duration of the bug report, so regard as the two modification history features as query dependent.

Structural Information Retrieval

By calculating similarities of each method and then maximizing across each and every one method in a source file follows the procedure of IR approach [28]. Here the author proposed a report-based bug localization approach which determines the final ranking function score by the summation of lexical similarities of each and every one possible eight document-query field pairs. The eight features ϕ_7 to ϕ_{14} determined to each input fields source file and the bug report in their entirety described as follows,

$$\phi_7(r,s) = sim(r.summary, s.class) \quad (11)$$

$$\phi_8(r,s) = sim(r.summary, s.method) \quad (12)$$

$$\phi_9(r,s) = sim(r.summary, s.variable) \quad (13)$$

$$\phi_{10}(r,s) = sim(r.summary, s.comment) \quad (14)$$

$$\phi_{11}(r,s) = sim(r.description, s.class) \quad (15)$$

$$\phi_{12}(r,s) = sim(r.description, s.method) \quad (16)$$

$$\phi_{13}(r,s) = sim(r.description, s.variable) \quad (17)$$

$$\phi_{14}(r,s) = sim(r.description, s.comment) \quad (18)$$

File dependency graph model

Wait for complex code toward be more prone to bugs than simple code. Consequently, the complexity of the source code file enclosed in a file could give a further functional signal with related to the likelihood with the purpose of the file contains bugs. A correct measure of code complexity might needs a correct representation of the semantics of the code. Since a wide-ranging semantic examination of code is presently not sufficient option to a classification of code complexity depending on syntactic features. For instance, a proxy evaluation designed for the complexity of a source code file might be described as follows: 1) The complexity increases by means of each new class with the purpose of is second-hand in the code. Because each class might be mapped to a specific source code file with the purpose of implements it, be able to reformulate this property and say with the purpose of the complexity of a source code s is absolutely correlated by means of the number of source code files on which s relies on the number of file addiction of source code file s. 2) The complexity of a source code s relies on the amount of dependencies between files, in addition it also rely on the complexity of each

dependencies. If s relies on two other source files s_1 and s_2, ,it concludes that the first source file is more complex than the second source file ,similarly source file one have produces more bugs than that of the source file two. With the purpose is to say, by means of using a complex construct is much complex than by means of using an easy construct, and consequently more expected toward direct to bugs.

3) The perceived difficulty of a code artifact that is class, source code file reduces by means of each extra use, as programmers develop into more familiar through it and consequently fewer possible to make use of it wrongly.

4) The complexity of source code file is relies on factors of each methods with their file dependencies. This is a catch-each and every one component of the complexity measure with the purpose of though complex to completely capture formally requires to be addressed in some helpful set description of code complexity. The above mentioned three properties are applied to solve code complexity with the purpose is similar to the description of web page quality used in the PageRank algorithm [29], where a hyperlink from page p_1 to page p_2 confers to p_2 a fraction of the quality of p_1, depending on the statement with the purpose by linking to p_2, the creator of p_1 thinks that p_2 is of high quality. Let G =(E, V) dependency graph to represent the software project, where V is the set of source code files and edge t s E represents with the purpose source file t is used in source file s that the t is a file dependency of s. Additionally, let s.inLinks and s.outLinks denote the number of edges related to source file s and leaving from s, correspondingly. Then the PageRank complexity of a source code file s have been described as follows,

$$K(s) = \sum_{t\ s} \alpha \times \frac{K(t)}{|t.outlinks|} + (1 - \quad) \times \frac{1}{|V|} \tag{19}$$

The initial term confine the first three properties specify above: 1) each file dependency t increases the complexity of s; 2) a source file t's involvement in the direction of the complexity of s relying on the complexity of t itself; 3) t's complexity contribution decreases by means of the number of files using it. The second term confines the fourth part of the complexity: 4) a fraction of the overall complexity of s is appropriate in the direction of factors not captured all the way through file dependencies. Designed for straightforwardness and tractability, let us presume that these other sources of complexity are circulated uniformly over each and every one the source code files in a project. The damping factors are denoted as the percentage of complexity with the intention of expected toward be captured all the way through file dependencies. Specified a source code file s calculate the number of file dependencies of s as well as the number of files with the purpose of depending on the number of outlinks of s:

$$[\text{illegible}]) = s.inlinks \tag{20}$$

$$[\text{illegible}]) = s.outlinks \tag{21}$$

PageRank score

Complexity of a source code file have been measured using PageRank approach is described as follows,

$$[\text{illegible}]) = PageRank(s) \tag{22}$$

Authority and Hubs score

The Hubs and Authorities [30]is also named as Hyperlink- Induced Topic Search (HITS) used for determining web pages by means of high quality data on a specific topic and web pages via high quality suggestion links designed for each data. A web page by means of high authority score is documented as an expert by means of provides relevant data on a topic, and is consequently linked by means of many hub pages. A web page through a high hub score is documented as a good list of links in the direction of numerous authority pages. In a Java project, an abstract

class with the purpose is implemented by means of several other files is predictable to have a high hub score. Correspondingly, a source code file with the purpose of consists the experimentation of a class with the purpose of extends another class and experiments several interfaces is predictable to have a high authority score,

$$(r, s) = Hub(s) \quad (23)$$
$$) = Authority(s) \quad (24)$$

Bug-Fixing Frequency occurrence

The proposed SVM model weight values of each bugs is computed from bug fixing frequency toward each bug reports r corresponding by means of source file s is calculated not only by considering bug reports or query itself, consideration of both. The FOFR weight values is calculated by taking both frequent occurrence of the bug reports r $FQ = \{fo_1, \dots fo_j\}$ and each bug reports $BR = \{br_1, \dots br_i\}$ .The SVM ranking is used for learning frequency occurrence of bug reports ranking functions with the purpose of aim thus reducing the required bug's reports error. Support Vector Machine (SVM) has been getting popularity in machine learning since proposed and has been extensively used to solve the problems of pattern recognition and regression estimation [31]. The original concept of SVM is proposed for binary classification. Given a training set $(x_i, y_i), i = 1, 2, \cdot \cdot \cdot, n, x_i \in R$, where xi is the ith input vector of d-dimension, $y_i \in \{-1, +1\}$ is the corresponding class label, and n is the number of training samples, SVM constructs a separating hyperplane that separates the training vectors perfectly (supposing that the training set is linearly separable) with the closest training vectors beside the hyperplane as far as possible away from those in the other class. This amounts to solving the following optimization problem

$$\min_{\omega, b} \frac{1}{2} ||\omega||^2 \ s.t. \ y_i(\langle \omega, x_i \rangle + b) \geq 1 \quad (25)$$

where and b are the undetermined parameters in the hyperplane $< \ , x + b > = 0$. In the real world applications, however, most problems are nonlinear [32]. In this case, the nonlinear data need to be mapped to a new space and allow for wrongly classified samples. Using the Lagrange method, the optimization problem (25) in the nonlinear case with a soft margin can be transformed to the following dual form

$$\min \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \Phi(x_i), \Phi(x_j) \rangle \ s.t. \ 0 \leq \alpha_i \leq C, \sum_i \alpha_i y_i = 0 \quad (26)$$

Where $_i$ is the Lagrange multiplier, (x) is the nonlinear mapping function and C is the penalty coefficient denoting the extent to which we restrict wrongly classification. The final decision function is

$$Class(x) = sign\left(\sum_i \alpha_i y_i < \Phi(x_i), \Phi(x) > +b\right) \quad (27)$$

The inner product in the new feature space $K(x, y) = < \ (x), \ (y) >$ is called kernel function. The introduction of kernels broadens greatly the application of SVM. Besides the linear kernel, of which the corresponding feature space is just the original space, Radial Basis Function (RBF) kernel is the most frequently used kernel [33]. Its expression is

$$K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}, \sigma > 0 \quad (28)$$

SVM-RFE is first proposed that selects features in the way of backward elimination. Specifically, in each iteration that removes one feature which influences the least the value of the objective function. The objective function is J = ½|| ||2 according to (27). Here, the same as in most related works, we

adopt the linear kernel. According to the Optimal Brain Damage (OBD) algorithm , the change of the objective function with respect to the removing of the ith feature satisfies

$$J(i) = \frac{J}{i} \Delta\alpha_i + \frac{{}^2J}{2\,i}(\Delta\ _i)^2 \quad (29)$$

$$J(i) = (\ _i)^2 \quad (30)$$

can ignore the first order term of (2) at the optimum of J, which leads to high accuracy. Therefore, we remove features iteratively in terms of the absolute or squared value of $_i$ as $_i$equals $_i$ in the case of removing the ith feature. The detailed procedure of SVM-RFE can be described as follow

1) Initialize the original feature set as F.

2) Train a linear SVM with feature set F.

3) Compute the weight vector $= \ _i y_i x_i \alpha_i$

4) Remove one feature with the smallest ^2_i value from F.

5) Repeat steps 2 to 4 until the size of F equals the predefined size of the final feature subset.

The final score value is normalized by means of the summation of the edit distance of each frequently happened bug reports toward the closest one in the dataset. Ranked sets of frequently occurred bug reports are used as training data designed for a ranking function. The computed ranking function has been used toward rank the frequently occurred bug reports as well as to score disregarded bug reports.

Feature scaling

Features with widely different ranges of values are detrimental in machine learning models. Feature scaling helps bring all features to the same scale so that they become comparable with each other. For an arbitrary feature , let $.min$ and $.max$ be the minimum and the maximum observed values in the training dataset. A feature f may have values in the testing dataset that are larger than $.max$, or smaller than $.min$. Therefore, examples in both the training and testing dataset will have their features scaled as follows:

$$= \begin{cases} 0 & \text{if} \ < \ .min \\ \frac{-\ .min}{.max - \ .min} & \text{if} \ .min < \ \ .max \\ 1 & \text{if} \ \ .max \end{cases} \quad (31)$$

This system is proposed that considers features with the purpose of association related to lexical gap via the use of project precise Application Programming Interface (API) report to attach Natural Language parameters in the bug report by means of programming language with the purpose of build in the source code. The major contribution of this paper is to propose FOFR ranking method which solves bug report problem related to source files with the purpose of permits the faultless integration of a different type of features with the purpose of confine a determine of code complexity.

4. RESULTS AND DISCUSSION

In the literature many bug localization methods have been used just one code revision to evaluate the software system performance depending on multiple bug reports. Therefore, via just one revision of the software source code package used for experimentation should increases performance with the purpose of overestimate the normal performance of the system when second-hand in practice. For instance, the dataset collected from [34] consists of 3,075 bug reports via a fixed version of the Eclipse 3.1 source code package. One of the files with the purpose of was fixed designed for this bug report is MethodBinding.java. At the time the bug report was submitted, this class mightn't consists of a isVarargs() method. The occurrence of potential bug-fixing information in the predetermined revision dataset is probable in the direction of direct toward an unrealistic approximation of the

system performance, as the bug report has a larger textual similarity by means of the future version of the MethodBinding.java file than by means of the current version.

4.1. Performance Evaluation Metrics

Given a test dataset of bug reports overall system performance is then calculated using the following evaluation metrics: Accuracy@k evaluates the percentage of bug reports used for which the software system creates at least one correct suggestion in the top k ranked files.

Prec@k is the retrieval precision over the top k documents in the ranked list:

Prec@k= # of relevant docs in top k/ k Recall is the fraction of the k documents with the purpose of is appropriate to the query with the purpose of is effectively retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \quad \{\text{Retrived Documents}\}|}{|\{\text{relevant documents}\}|}$$

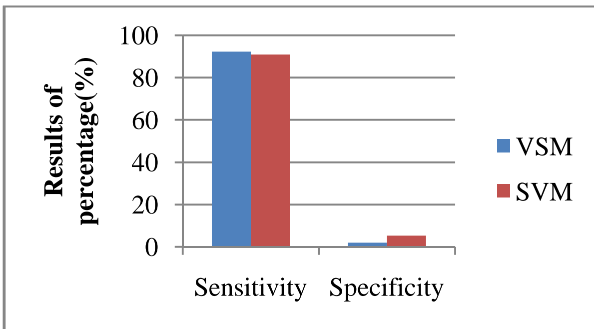

Figure 2- Sensitivity and Specificity comparison vs methods

From the experimental results it is explained that for the bug report dataset the proposed SVM algorithm produces 90.895 % and VSM produces 92.357% is illustrated in Figure 2. It concludes that the proposed SVM algorithm produces higher sensitivity value and less error rate when compared to all methods. From the experimental results it is explained that for the bug report dataset the proposed SVM algorithm produces 5.2568% and VSM produces 1.9632% is illustrated in Figure 2. It concludes that the proposed SVM algorithm produces higher specificity value when compared to all methods.

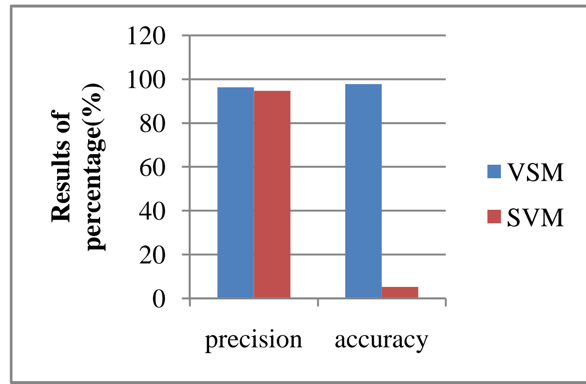

Figure 3 - Precision and Accuracy comparison vs methods

From the experimental results it is explained that proposed SVM algorithm produces 97.81 % and VSM produces 96.25% precision value is illustrated in Figure 3. It concludes that the proposed SVM algorithm produces higher precision value when compared to all methods. From the experimental results it is explained that proposed SVM algorithm produces 97.28% and VSM produces 94.72% is illustrated in Figure 3. It concludes that the proposed SVM algorithm produces higher accuracy value when compared to all methods.

Methods	Sensitivity	Specificity	Precision	Accuracy
VSM	92.357	1.9632	96.25	94.72
SVM	90.895	5.2568	97.81	97.28

Table 1- Metrics results comparison vs methods

CONCLUSION AND FUTURE WORK

To find a bug, developers make use of not only the information of the bug report however in addition gathering domain knowledge appropriate to the software project. In this paper work introduces a new learning based ranking approach with the purpose of imitate the bug discovering procedure employed by developers. Known a bug report, the ranking score value of each source file is calculated as a weighted grouping of an array of features, where the weights are trained repeatedly by using learning based ranking approach. Support

Vector Machine (SVM) system is proposed that considers features with the purpose of association related to lexical gap via the use of project precise Application Programming Interface (API) report to attach Natural Language parameters in the bug report by means of programming language with the purpose of build in the source code. The major contribution of this paper is to propose Support Vector Machine (SVM) ranking method which solves bug report problem related to source files with the purpose of permits the faultless integration of a different type of features. In the experimentation work wide-ranging evaluation and comparisons is done by comparing the results to state-of-the-art methods, it concludes that the proposed FOFR system have achieves higher ranking accuracy. Feature evaluation results are able to be making use to choose a subset of features in order to obtain an objective trade-off among software system accuracy and runtime complexity. In the scope of the future work determination influence further types of domain information are stack traces presented by means of bug reports and the file change history, as well as features formerly used in fault calculation systems. In addition plan to make use of the ranking SVM by means of nonlinear kernels and additionally assess the ranking approach on software projects in other programming languages. In scope the future work some other methods have been implemented to gather bug reports from the database with the purpose is the make use of web-services should be investigated and experimented in order to facilitate the integration of SVM model in the direction of existing bug repositories. Ranking models, initiate a new learning models in the direction of help searches be able to be integrated to the SVM toward enhance search features. For instance, it is able to be added search during tags and query reformulation. Moreover, other ranking techniques might be helpful to enhance search results.

REFERENCES

[1]. R. P. L. Buse and T. Zimmermann, “Information needs for software development analytics,” in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2012, pp. 987–996.

[2]. B. Bruegge and A. H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd ed. Upper Saddle River, NJ, USA, Prentice-Hall, 2009

[3]. T. D. LaToza and B. A. Myers, “Hard-to-answer questions about code,” in Proc. Eval. Usability Programm. Lang. Tools, New York, NY, USA, 2010, pp. 8:1–8:6.

[4]. A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern code review,” in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2013, pp. 712–721.

[5]. S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, “Information needs in bug reports: Improving cooperation between developers and users,” in Proc. ACM Conf. Comput. Supported Cooperative Work, New York, NY, USA, 2010, pp. 301–310.

[6]. C. D. Manning, P. Raghavan, and H. Sch€utze, Introduction to Information Retrieval. New York, NY, USA: Cambridge Univ. Press, 2008.

[7]. N. Bettenburg, S. Just, A. Schr€oter, C. Weiss, R. Premraj, and T. Zimmermann, “What makes a good bug report?” in Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng., New York, NY, USA, 2008, pp. 308–318.

[8]. F. Rahman and P. Devanbu, “How, and why, process metrics are better,” in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2013, pp. 432–441.

[9]. P. Melville and V. Sindhwani, “Recommender systems,” in Encyclopedia of Machine Learning, C. Sammut and G. Webb, Eds., New York, NY, USA: Springer, 2010, pp. 829–838.

[10]. Anvik, J., Hiew, L., and Murphy, G. C. (2005). Coping with an open bug repository. In Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, pages 35–39, New York, NY, USA. ACM Press.

[11]. Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In Proceeding of the 28th International Conference on Software Engineering (ICSE'06), pages 361–370, New York, NY, USA. ACM Press.

[12]. Song, Q., Shepperd, M. J., Cartwright, M., and Mair, C. (2006). Software defect association mining and defect correction effort prediction. IEEE Transactions on Software Engineering, 32(2), 69–82.

[13]. Ko, A. J., Myers, B. A., and Chau, D. H. (2006). A linguistic analysis of how people de-scribe software problems. In Proceedings of the Visual Languages and Human-Centric Computing (VLHCC'06), pages 127–134, Washington, DC, USA. IEEE Computer Science.

[14]. Sandusky, R. J., Gasser, L., and Ripoche, G. (2004). Bug report networks: Varieties, strategies, and impacts in a f/oss development community. In Proceedings of the 1st International Workshop on Mining Software Repositories (MSR'04), pages 80–84, University of Waterloo, Waterloo.

[15]. Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., and Wang, B. (2003). Automated support for classifying software failure reports. In Proceedings of the 25th International Conference on Software Engineering (ICSE'03), pages 465–475, Washington, DC, USA. IEEE Computer Society.

[16]. Hiew, L. (2006). Assisted Detection of Duplicate Bug Reports. Master's thesis, The University of British Columbia.

[17]. Runeson, P., Alexandersson, M., and Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. In Proceedings of the 29th International Conference on Software Engineering (ICSE'07), pp.499–510. IEEE Computer Science Press.

[18]. Pohl, K., Böckle, G., and van der Linden, F. (2005). Software Product Line Engineering: Foundations, Principles, and Techniques.

[19]. Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. In Proceedings of the 13th International Conference on Software Engineering (ICSE'08), pp. 461– 470. ACM Press.

[20]. Canfora, G. and Cerulo, L. (2006). Supporting change request assignment in open source development. In Proceedings of the 2006 ACM Symposium on Applied Computing (SAC'06), pages 1767–1772. ACM Press.

[21]. Anvik, J. and Murphy, G. C. (2007). Determining implementation expertise from bug reports. In Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR'07). IEEE Computer Society.

[22]. Sandusky, R. J., Gasser, L., and Ripoche, G. (2004). Bug report networks: Varieties, strategies, and impacts in a f/oss development community. In Proceedings of the 1st International Workshop on Mining Software Repositories (MSR'04), pages 80–84, University of Waterloo, Waterloo.

[23]. Antoniol, G., Penta, M. D., Gall, H., and Pinzger, M. (2005). Towards the integration of versioning systems, bug reports and source code meta-models. Electronic Notes in Theoretical Computer Science, 127(3), 87–99.

[3]. Koponen, T. and Lintula, H. (2006). Are the changes induced by the defect reports in the open source software maintenance? In H. R. Arabnia and H. Reza, editors, Proceedings of the 2006 International Conference on Software Engineering Research (SERP'06), pp. 429–435. CSREA Press.

[4]. S. Rao and A. Kak, "Retrieval from software libraries for bug localization: A comparative study of generic and composite text models," in Proc. 8th Working Conf. Mining Softw. Repositories, New York, NY, USA, 2011, pp. 43–52.

[5]. J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports," in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2012 pp. 14–24.

[6]. F. Rahman and P. Devanbu, "How, and why, process metrics are better," in Proc. Int.

Conf. Softw. Eng., Piscataway, NJ, USA, 2013, pp. 432–441.

[7]. R. Saha, M. Lease, S. Khurshid, and D. Perry, “Improving bug localization using structured information retrieval,” in Proc. IEEE/ ACM 28th Int. Conf. Autom. Softw. Eng., Nov. 2013, pp. 345–355.

[8]. L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web,” Stanford InfoLab, Stanford University, Tech. Rep. 1999-66, 1999.

[9]. J. M. Kleinberg, “Authoritative sources in a hyperlinked environment,” J. ACM, vol. 46, no. 5, pp. 604–632, 1999.

[10]. S. Ahmad, A. Kalra, and H. Stephen, “Estimating soil moisture using remote sensing data: A machine learning approach,” Advances in Water Resources, vol. 33, no. 1, pp. 69–80, 2010.

[11]. S. Yin, X. Xie, J. Lam, K. C. Cheung, and H. Gao, “An improved incremental learning approach for kpi prognosis of dynamic fuel cell system,” Cybernetics, IEEE Transactions on, 2015.

[12]. D. S. Broomhead and D. Lowe, “Radial basis functions, multi-variable functional interpolation and adaptive networks,” DTIC Document, Tech. Rep., 1988

[13]. J. Zhou, H. Zhang, and D. Lo, “Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports,” in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2012 pp. 14–24.