A Bug Mining Tool to Identify and Analyze Security Bugs using Naive Bayes and TF-IDF

(A Comparative Analysis)

Diksha Behl, Sahil Handa, Anuja Arora
CSE / IT Department
Jaypee Institute of Information Technology Noida, India
dikshabehl5392@gmail.com, sahil_handa1992@yahoo.co.in , anuja.arora@jiit.ac.in

Abstract— **Bug report contains a vital role during software development, However bug reports belongs to different categories such as performance, usability, security etc. This paper focuses on security bug and presents a bug mining system for the identification of security and non-security bugs using the term frequency-inverse document frequency (TF-IDF) weights and naïve bayes. We performed experiments on bug report repositories of bug tracking systems such as bugzilla and debugger. In the proposed approach we apply text mining methodology and TF-IDF on the existing historic bug report database based on the bug's description to predict the nature of the bug and to train a statistical model for manually mislabeled bug reports present in the database. The tool helps in deciding the priorities of the incoming bugs depending on the category of the bugs i.e. whether it is a security bug report or a non-security bug report, using naïve bayes. Our evaluation shows that our tool using TF-IDF is giving better results than the naïve bayes method.**

Key Words — **Bug, TF-IDF, Naïve Bayes, security bug reports, non-security bug report, mining, text analysis**

I. INTRODUCTION

The advancement in software technology led to an increase in the development of software projects thereby increasing the requirement of an efficient bug tracking system such as bugzilla [7], an open source system used in Mozilla [7] released as open source software by Netscape communications. The bug tracking system should provide an interactive platform for bug reporting and tracking the progress. The generic process of the reporting a bug generally contain the information like bug id, priority, description, category, resolution, status, component and product.

In a bug tracking system, some bug reports are mislabeled by bug reporters as security bug reports, where its description satisfies a non-security bug report characteristics. A security related bug report requires a higher priority for fixing than a Non-security bug reports. Any type of delay in identifying and fixing the priority bugs can cause damage to software users or stakeholders so, labeling the bug reports correctly as security or non-security bugs is important. There are chances that the reporter of the bug reports the bug incorrectly, if bug reporters recognize a precise security bug as an irreverent non-security bug, then they may label the BUG REPORT as a SECURITY BUG REPORT. As a result, manual inspection of Non-security bug reports to identify Security bug reports is infeasible.

Therefore, an efficient bug detection tool for reducing human efforts and for analyzing bug reports and supporting bug tracking system in the process of identifying bug reports as security or non-security is required. To satisfy such a strong need, in this paper, we propose a new tool incorporating Term frequency-inverse document frequency (TF-IDF) method wherein mining on text descriptions of Bug reports is done to reach a model to classify a BUG REPORT as either a SECURITY BUG REPORT or a NON-SECURITY BUG REPORT. Our approach also compares the results with the probabilistic naïve bayes approach. In a probabilistic Naïve Bayes we can use different models like Bernoulli or multinomial event model for classification purposes. Thereby, making TF-IDF based bug Mining tool feasible to use as a complimentary tool in the BUG TRACKING SYSTEM.
This research paper has two most important endeavors:

- The bottom line of our proposed tool is to retrieve useful information of Bug reports from a BUG TRACKING SYSTEM database using text mining through developing a natural language based on description of the bug reports.
- The comparison of our proposed tool model with the naïve bayes approach. The results of our tool prove to be fruitful, showing more accuracy than the former approach.

II. RELATED WORK

Various research works have been done in the field of bug tracking systems. An approach undertaken by Neelima, Annapurna, V. Alekhya and Dr. B. M. Vidyavathi [15] to detect bugs in C programs via matching and mining techniques helped in easy detection of bugs, wherein, the input for the system is a text file containing errors, primarily syntax errors from a C program which is matched for similar but slightly different code fragments in the database that acts a repository of errors aptly classify them and generate an analysis report predicting solutions for the same.

978-1-4799-2995-5/14/$31.00©2014 IEEE

Cubranic and Murphy [4] proposed to apply machine learning techniques to assist in bug triage by using text categorization to predict the developer that should work on the bug based on the bug's description. They demonstrated their approach on a dataset of 15,859 bug reports taken from a large open-source project. Their evaluation shows that their prototype, using supervised Bayesian learning, can correctly predict 30% of the report assignments to developers. They applied these techniques to a selection of bug reports from the Eclipse project and tested their accuracy in assigning reports to developers.

Anvik[9] elaborated the research work of Cubranic and Murphy [4] and helped in determining a suitable software engineer for a working on a specific Bug Report. Support vector machines were mainly used to reach to valuable information from the summary and text description of Bug Reports to create term vectors. These term vectors are then used to predict the name of the developer who should fix the higher priority bug accordingly. Their work got a precision level of 64% for Firefox and 57% for the Eclipse project.

Leon Wu, Boyi Xie, Gail Kaiser and Rebecca Passonneau [11] developed a tool that derives useful information using data mining from an existing and historic bug report database and then uses the derived information to do completion checks and redundancy checks on either a new bug report or a given bug report, and also helps in estimating the trend of the bug report using statistical analysis. The empirical studies of their tool using several real-world bug report repositories shows that it is easy to implement, effective and has relatively high accuracy despite of a low quality data.

Nicholas Jalbert and Westley Weimer [12] have developed a system that saves the developer's time by automatically detecting the duplicate bug reports. Their developed system used features such as graph clustering and textual semantics to find duplicate bug reports out of the database. Using a dataset of 29,000 bug reports from the Mozilla project, they performed experiments that included a simulation of a real-time bug reporting environment. Their system was able to reduce development cost by filtering out 8% of duplicate bug reports while allowing at least one report for each real defect to reach developers.

Michael Gegick, Tao Xie and Pete Rotella [6] proposed an approach that used mining on text descriptions of bug reports to classify Bug Report as either a Security Bug Report or a Non-Security Bug Report. They implemented their solution using a well renowned commercial text mining tool called SAS Text Miner and they evaluated their proposed system on a large database of Cisco software system that contains more than ten hundred million source lines of code. Their approach identified SBRs mislabeled as NSBRs as high as with a percentage of 78% for Cisco system.

Sangeeta Lal and Ashish Sureka [13] investigated similaritiesand differences between different types of bug reports (cleanup, crash, polish, performance, regression, security and usability) across multiple dimensions within the same project to increase our understanding of different bugreport types. They reported experimental results on dataset from Google Chrome browser project.

III. EMPIRICAL ANALYSIS

A. Experimental Dataset

The standard bug report dataset used is taken from the bugzilla repository of bug reports [7]. This database contains 10000 instances and 7 attributes. The data set consists of detailed information about each bug. This includes a unique bug id, the name of the product in which the bug originated, the component of the product in which the bug originated, status of the bug i.e. whether it is a new, resolved, verified, reopened or closed bug, the assignee of the bug, summary of the bug having complete description of it and the category of the bug inclusive of security bug reports and non-security bug reports. But owing to the presence of many missing values and also zero valued columns, it became imperative to apply the concept of Dimensional Reduction i.e. Feature Selection and Feature Extraction and resize the dataset while maintaining the reliability and relevance of the dataset. Thus, to obtain accurate results, pre-processing was done on the bug reports data set by filling the missing values with the average value of the attribute. An attribute with all the missing values and all similar values was removed. Hence, final dataset used for the analysis contained 4 attributes and 10000 records.

B. The Bag-of-Words Representation

Given a database of bug reports, first and foremost we need to retrieve a collection of words by considering all the words used at least once in the bug report database. This collection of words is called the vocabulary of bug reports. Generally, it is vast in size and hence minimized by taking words that are used in at least two bug reports. This vocabulary is alphabetically sorted to fix its ordering of words so that a word 1 can be easily retrieved through V words, where V is the size of the vocabulary. On complete generation of vocabulary, each and every bug report is depicted as a vector of words with integer entries of size length V = n. For an example, if the vector is x then its i^{th} component is x_i which depicts the number of occurrences of word j in the bug report database. The length of the bug reports is l, which can be mathematically written as

$$l = \sum_{i=1}^{n} x_i \qquad (1)$$

Each vector can be assigned weighting functions corresponding to their terms and frequencies. The aggregate weight of a term is determined addition of its term weight and its frequency weight. There are two models to give weights namely the Log frequency weight function and Entropy term weight function.

We are using the Log frequency weight function to reduce the weight of a single term occurring frequently in a single bug report. The log frequency weight of term t in bug report d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d}, & \text{if } tf_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

We define the IDF (inverse document frequency) of t by

$$\mathrm{idf}_t = \log_{10} N/\mathrm{df}_t \quad (3)$$

Where, df_t is the document frequency of t i.e. the number of bug reports that contain t .

According to log frequency formulation, the final weight of a term is the product of its inverse document frequency weight and its term frequency weight and its: [10]

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log N/\mathrm{df}_t \quad (4)$$

Our proposed model can estimate whether a bug report belongs to a security bug category or a non-security bug category referring to the weighted values in its complete word vector.

C.Stop List and Start List

In computing the terms, we also used the stop-words removal to remove short function words that are of least importance present in most of the bug reports and these words do not relate to any specific subject matter. The stop words can include connectives (and, because, however), pronouns (you, he, it), auxiliaries (have, been, can), prepositions (to, of, before), and generic nouns (amount, part, nothing). In this way, only a smaller set of keywords are used to query all the bug descriptions using the standard vector-space

However, these generic start words comprises of valuable information for many jobs, including detecting bug reports generated from security or non-security domains or identifying the software engineer to handle a bug report . A set of bug reports (shown in Table 1) is represented as a two-dimensional matrix in which each column corresponds to a word out of the complete vocabulary generated and each row comprises of count of words describing a bug report. Each entry in the matrix shows the occurrence of a word in a bug report as an integer count.

TABLE I. BUG VECTOR SPACE MODEL

Bug ID \ Word	Ability	Security	manage	network	failure
1	1	0	2	0	0
2	0	2	0	3	1
3	0	1	0	0	1
4	1	0	0	0	0
5	1	1	0	0	0

A start list contains words that are of maximum importance as these contain information that gives indication of categories of bug reports. Terms from complete dataset of bug reports are entered in the matrix if they exist in the start list. The terms in the start list are not exclusive to Non-Security Bug reports or security bug reports, the frequency of existence along with the presence with other security related terms increase the chances that the considered Bug report is a Security bug report.

IV. SOLUTION APPROACH

Our solution approach involves developing a statistical model using term frequency- inverse document frequency weights and developing a vector space model to find relationships between vocabulary words and category of bug reports (security bug reports or non-security bug reports). Finally, comparing our proposed model to Bernoulli document model and finding differences in success rate and precision rate.

A. Naïve Bayes Approach

Consider a bug report B, whose categories are given by C. In the bug report analysis, there are two classes C = S (security bug report) and C = N (Non-Security bug report). We classify B as the class which has the highest posterior probability P (C|D), which can be re-expressed using Bayes' Theorem:

$$P(C|B) = (P(B|C)\,P(C))/P(B) \propto P(B|C)\,P(C) \quad (5)$$

We shall look at two probabilistic models of documents, both of which represent bug reports as a bag of words, using the Naive Bayes assumption. Both models represent bug reports using feature vectors whose components correspond to word types. If we have a vocabulary V, containing |V| word types, then the feature vector dimension d=|V|.

Bernoulli document model: a bug report is represented by a feature vector with binary elements taking value 1 if the corresponding word is present in the bug report and 0 if the word is not present.

Multinomial document model: a bug report is represented by a feature vector with integer elements whose value is the frequency of that word in the bug report.

For the purpose of comparison, we used Bernoulli document model.

1) The Bernoulli document model

In the Bernoulli model [14] a bug report is represented by a binary vector in the space of words. On developing a vocabulary V containing a set of |V| words, then the d^{th} dimension of a bug report vector corresponds to word w_t in the vocabulary. Let F_i be the feature vector for the i^{th} bug report D_i ,then the d^{th} element of F_i, written F_id, is either 0 or 1 representing the absence or presence of word w_d in the i^{th} bug report. Let $P(w_d|C)$ be the probability of word w_d occurring in a bug report of class C; the probability of w_d not occurring in a bug report of this class is given by $(1-P(w_d|C))$. If we consider the naive Bayes assumption that the probability of each word occurring in the bug report is independent of the occurrences of the other words, then we can write the bug report likelihood $P(D_i|C)$ in terms of the individual word likelihoods $P(w_d|C)$:

$$P(D_i|C) \sim P(F_i|C) = \pi [F_i d P(w_d|C) + (1 - F_i d)(1 - P(w_d|C))] \quad (6)$$

(3)

This product goes over all words in the vocabulary. If word wt is present, then $F_i d = 1$ and the required probability is $P(w_d|C)$; if word w_d is not present, then $F_i d = 0$ and the required probability is $1 - P(w_d|C)$.

We can imagine this as a model for generating bug report feature vectors of class C, in which the bug report feature vector is modeled as a collection of $|V|$ weighted coin tosses, the d_{th} toss having a probability of success equal to $P(w_d|C)$. The parameters of the likelihoods are the probabilities of each word given the bug report class $P(w_d|C)$; the model is also parameterized by the prior probabilities, P(C). We can learn these parameters from a training set of bug reports labeled with class C = k. Let $n_k(w_d)$ be the number of bug reports of class C = k in which w_d is observed; and let N_k be the total number of bug reports of that class. Then we can estimate the parameters of the word likelihoods as,

$$P(w_d| C=k) = n_k(w_d) / N_k \quad (7)$$

The relative frequency of bug reports of class C = k that contain word w_d. If there are B bug reports in total in the training set, then the prior probability of class C =k may be estimated as the relative frequency of bug reports of class C=k:

$$P(C=k) = B_k/B \quad (8)$$

B.Text Mining

The Proposed solution involves the implementation using term weighing as described in Figure 1 right portion. Finally our proposed tool's results are compared with the results of naïve bayes approach on the same test dataset. Detailed explanation of figure 1 is given in the following points.

1) Preparation Of Data

First, we obtain Bug Reports from the database that was retrieved from different sources. Second, we distinguish between Security Bug Reports and Non-Security Bug Reports among the obtained Bug Reports. In the database, a Bug Report contains a category field indicating the genre of the report i.e. whether the Bug Report is a Security Bug Report or a non-security Bug Report. Applying a query on this field in the system leads all known SBRs as a result. At the bottom line, we use labeled Bug Report training data to build our proposed predictive model.

2) Preparation Of Start Lists and Stop Lists

We obtain the terms from the text description of Bug Reports and prepare the start list using all these terms. We add terms to the start list from the Bug Reports using the tool. We manually added prepositions, conjunctions and articles to the stop list as they are likely to have least important information for indicating the genre of a bug report. We used both start lists and stop lists for our proposed tool. We generated a term-by-bug report frequency matrix from the words present in the description of the Bug Reports using the stop list and the start list. The above shown matrix is a quantified format of the natural language descriptions in the Bug Reports. As the number of bug reports increases, the term-by-bug report frequency matrix becomes large.

3) Development of the Model

We used the term-by-bug report matrix as the predictor variable or controlled variable (i.e., the input variable) in our proposed model. The response variable (i.e., the outcome variable) is the category (Security Bug Report or Non- Security Bug Report) of a Bug Report. We developed a trained predictive model based on the term-by-bug report matrix. When the precision and success rates are satisfactory, then the model is used by providing a new Bug report data set as an input variable to the model for predicting their outcome variable i.e. category.

We finally found that words present in bug report descriptions are not random and we see association between terms and bug types. And on the basis of this relationship we segregated security bug reports and non security bug reports.

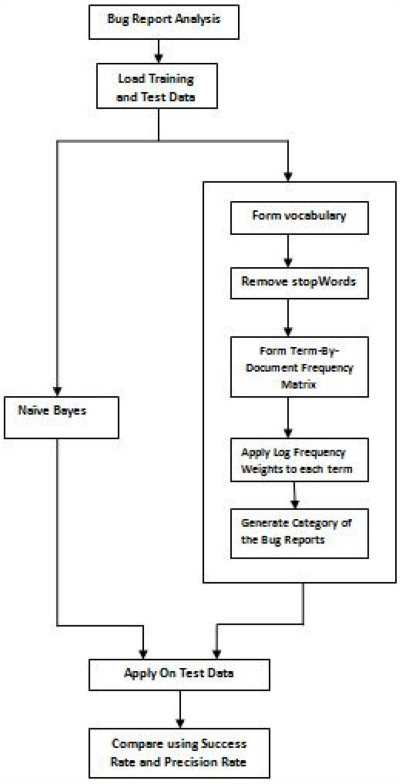

Fig. 1.Process Flow Diagram

V. RESULTS

Our model changed the mislabeled bug reports with a high success rate. We used the concept of true positives, false positives, false negatives, and true negatives. Taking the first case wherein an SBR is correctly classified by the model as asecurity related bug then this is a true positive(TP). The next one comprises of NSBR that is incorrectly classified to be an SBR leading to the generation of a false positive (FP). The third case includes SBR incorrectly classified as an NSBR, leading to the existence of a false negative (FN). The last case includes NSBR correctly classified to be an NSBR, leading to the generation of a true negative (TN).Misclassification can occur in the model if it classifies an SBR as an NSBR, or if it classifies an NSBR as SBR. The Information Extraction metrics of success and precision are used. Model success is the number of successful classifications divided by the number of all classifications. The precision rate of the model is the number of successful classifications divided by the number of bug reports attempted to be classified. These metrics for the success rate and precision rate are defined as follows [6]

$$\text{SUCCESS RATE} = \frac{((TP+TN)*100)}{TP+FP+TN+FN} \quad (9)$$

$$\text{PRECISION} = \frac{TP*100}{TP+FP} \quad (10)$$

Where, TP stands for true positive, FP for false positive, TN for true negative, and FN for false negative. In context of classification, TP represents Normal classified as Normal, FP represents Abnormal classified as Normal, TN represents Normal classified as Abnormal and FN represents Abnormal classified as Abnormal.

Our implemented tool based on TF-IDF is giving better results than the tool based on naïve bayes in terms of precision and success rate in accordance with the formulae mentioned above.

Table 2 below shows success rate and precision rate of our proposed tool over different number of datasets in comparison to naïve bayes approach. The table demonstrates better results of the tool in comparison to Naïve bayes algorithm. Moreover, it shows constant performance even on increasing the number of reports.

Table II . Results Of Comparison Using Varying Number Of Records

No of records		Naïve Bayes (%)	Proposed Tool (%)
1000	Success Rate	91.0	95.69
	Precision Rate	82.3529	93.186
5000	Success Rate	90.0566	93.499
	Precision Rate	83.0	92.144
10000	Success Rate	90.0506	93.989
	Precision Rate	82.45	92.56
20000	Success Rate	90.034	94.23
	Precision Rate	81.456	93.54

Figure 2 shows statistical view of comparison between naïve bayes and proposed tool's success rate. Figure 3 shows statistical view of comparison between naïve bayes and proposed tool's precision rate.

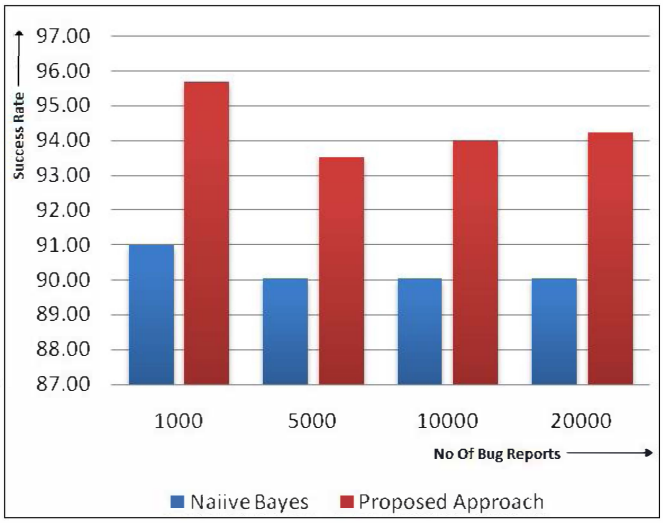

Fig. 2.Success Rate Comparison Graph

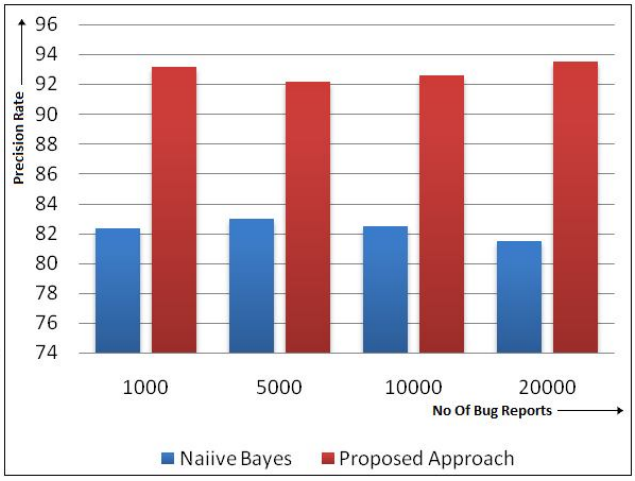

Fig. 3.Precision Rate Comparison Graph

VI. CONCLUSION

In this paper, we presented a tool that is able to retrieve and analyze bug reports from a bug report training database. It is able to derive vocabulary from the description of bug reports using text mining and log frequency text weights .Thereby, generating a relationship between words and bug report type (security or non-security) and modifying the mislabeled bug reports by generating a vector space model using the conclusions and inference from text weights and TF-IDF measurements.

On testing our tool on the same test data as of naïve bayes, we did a comparative analysis. Our approach gave a high success rate of 93.989 % and precision rate of 92.56 % with a dataset of 10,000 bug reports. This research shows the benefits of applying text mining and log frequency weights over naïve bayes approach in terms of both success rates and precision rate.

The future scope of the work includes:

- The use of statistical topic models which can be highly useful in summarizing huge collection of bug reports and analyzing topics and entities in the bug reports.
- The proposed approach can be extended further to deal with more categories of bug reports. For example: Crashbugs, Cleanup bugs, Security bugs, Performance bugs, Polish bugs and Regression bugs.

REFERENCES

[1] Čubranić, Davor. "Automatic bug triage using text categorization." In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. 2004.

[2] V. Neelima, Annapurna, V. Alekhya, and Dr. B. M. Vidyavathi, "Bug Detection through Text Data Mining", Volume 3 Issue 5, May 2013.

[3] Chaturvedi, K. K., and V. B. Singh. "Determining Bug severity using machine learning techniques." Software Engineering (CONSEG), 2012 CSI Sixth International Conference on. IEEE, 2012.

[4] Leon Wu Boyi Xie Gail Kaiser Rebecca Passonneau, "BugMiner: Software Reliability Analysis Via Data Mining of Bug Reports," in Proc. of the 23rd Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), Jul. 2011, pp. 95-100.

[5] Sangeeta Lal and Ashish Sureka, Empirical Analysis of Bug Report Types: A Case-study of Google Chrome Browser Project, 19th Asia-Pacific Software Engineering Conference, (APSEC), 2012

[6] Ashish Sureka and P. Jalote, Detecting Duplicate Bug Report Using Character N-Gram-Based Features, The 17th Asia-Pacific Software Engineering Conference (APSEC), 2010

[7] Jeong, Gaeul, Sunghun Kim, and Thomas Zimmermann. "Improving bug triage with bug tossing graphs." Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering. ACM, 2009.

[8] Denzil Correa and Ashish Sureka, Integrating Issue Tracking Systems with Community-Based Question and Answering Websites, 22nd Australasian Software Engineering Conference (ASWEC) 2013

[9] Ashish Sureka, Learning to Classify Bug Reports into Components, 50th International Conference on Objects, Models, Components, Patterns (TOOLS Europe), 2012

[10] Gegick, Michael, Pete Rotella, and Tao Xie. "Identifying security bug reports via text mining: An industrial case study." Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE, 2010.

[11] Sunil Joy Dommati, Ruchi Agrawal, Prof. Ram Mohana Reddy.G, and Sowmya Kamath, "Bug Classification: Feature Extraction and Comparison of Event Model using Naïve Bayes Approach" In proceeding of International Conference on Recent Trends in Computer and Information Engineering (ICRTCIE'2012) April 13-15, 2012 Pattaya

[12] John Anvik, Lyndon Hiew and Gail C. Murphy"Who Should Fix This Bug?" Proc of the ICSE, pp. 371-380, 2006.

[13] Nicholas Jalbert and Westley Weimer. Automated duplicate detection for bug tracking systems. In DSN 2008: 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 52–61, Anchorage, Alaska, USA, 2008.IEEE Computer Society.

[14] Lectures by: Prabhakar Raghavan (Yahoo and Stanford) and Christopher Manning (Stanford)). http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

[15] https://bugzilla.mozilla.org/