A Survey of Duplicate Bug Report Detection

Swetha Murali
Department of Computer Science,
North Carolina State University,
Raleigh, NC.
smurali8@ncsu.edu

Deepak Nair
Department of Computer Science,
North Carolina State University,
Raleigh, NC.
dnair2@ncsu.edu

Glen Menzes
Department of Computer Science,
North Carolina State University,
Raleigh, NC.
gmenze@ncsu.edu

ABSTRACT
Bug fixing is a vital process in producing a high-quality software product. There are many users who use the system and report issues found in the system. These issues are captured in a bug report which is stored in a bug repository. Since, many users interact with the system it is possible for a bug to have been reported more than once leading to duplicate bug reports. Detecting duplicate bug reports reduce maintenance effort and also provides more information about the detected bug hence, the process of detecting duplicate bug reports needs to be automated. Active research is being conducted in detecting duplicate bug reports and in this paper, we have provided a brief overview about the various techniques and also our opinions on these techniques and possible improvements.

General Terms
Algorithms, Measurement, Performance, Design, Experimentation, Security, Theory.

Keywords
Duplicate Bug Reports, Automated Triaging, Document Processing, Information Retrieval, Machine Learning, Performance Evaluation.

1. INTRODUCTION
A software bug is an error, fault or failure that interferes with the proper and efficient functioning of a software system. When a bug is encountered, it is reported. A bug report encapsulates the details regarding the bug and it is stored in a bug repository. Many testers and end users interact with the system and it possible that a bug might have been encountered by more than one user. This causes a single bug to be reported multiple times. As the software produce evolves over a period of time, the volume of bugs and bug reports increase.

There are various tools that have been developed like JIRA, Bugzilla, etc. which provide an interface for users to report bugs. These tools have greatly simplified the bug reporting procedure but, they are not equipped to identify and handle duplicate bug reports. Hence, there is a manual intervention in detecting duplicate reports. The person who manually detects duplicate bug reports is called a triager and the process of detecting duplicate reports is called triaging. There is a need to automate the triaging process as it reduces the work done by triagers and also prevents manual errors.

There is a lot of research being done in the field of automated triaging and in this paper, we capture some of the important and popular techniques used in this process. We also provide our views about the techniques and possible improvements that could be made.

In section 2 we provide a brief overview about the papers we surveyed. Section 3 lists the general procedure to be followed to classify duplicate bug reports. In section 4, we give a brief overview about the datasets that were encountered. Section 5 details all the pre-processing techniques, section 6 details all the features that were extracted, section 7 gives an overview about all the information retrieval techniques encountered and section 8 provides details about machine learning techniques. Section 9 and 10 provides details about similarity measure and performance measure metrics. We conclude the discussion by listing the threats to validity and future scope in section 11 and 12. In each of the sections we have included our comments and recommendations.

2. RELATED WORK
There is a huge amount of research being conducted in the area of automated triaging. There are two approaches which can be employed to detect duplicate bug reports. The first approach is to automatically filter duplicates and prevent the triagers from ever encountering duplicates and the second approach is to provide a list of similar reports for each incoming bug report to the triagers.

One of the earliest works in duplicate bug report detection was carried out by Runeson, Alexandersson and Nyholm [1] where they evaluated the feasibility of using Natural Language Processing (NLP) techniques to identify duplicate bug reports in Sony Ericsson Mobile Communications bug repository. The authors extracted textual features from the bug reports and represented the features in a vector space model. They used NLP techniques to identify the duplicates and retrieved the top 5, 10 and 15 similar reports for an incoming bug report. This technique could recognize about 40% of the duplicates in the bug repository and had a recall rate of about 67%.

Jalbert and Weimar [2] developed a technique of filtering incoming duplicate reports in Mozilla bug repository. The authors constructed a vector space model from surface features [3] and textual semantics [1] extracted from the incoming bug reports and then used graph clustering techniques [4] to identify the duplicates in the existing bug repository. This technique filtered about 8% of the incoming duplicate reports.

X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun [5] extended the work done in [1] by incorporating execution information along with NLP. The authors calculated the Natural Language similarity and Execution similarity between the new bug report and the existing repository of old reports. The authors then extracted top-k similar reports by combining both the similarity values. The authors then compared their technique with the one proposed in [1] on Mozilla bug repository and achieved about 21-24% increase in accuracy. Even though this technique obtained an improvement, only 0.83% of all the bug reports contain execution information.

In 2010, C. Sun, D. Lo, X. Wang, J. Jiang and S. Khoo [6] developed a classifier based approach to accurately retrieve the top-k similar reports. The authors retrieved the textual features from the summary and description fields of the bug reports. The authors then divided the bug reports into pairs of duplicate reports and non-duplicate reports and trained the Support Vector Machine (SVM) classifier. For any new bug report, the authors used the trained SVM classifier to identify and retrieve the top-k duplicates. This approach was compared with the techniques mentioned in [2], [1] and [5] on the bug repositories of Mozilla, OpenOffice and Eclipse and achieved an improvement in recall of about 17-31% in OpenOffice, 22-26% in Mozilla and 35-43% in Eclipse datasets.

Sureka and P. Jalote [7] developed a technique to identify duplicate bug reports written in free text form by extracting character n-gram features. The previous techniques mentioned are word based, whereas in this paper the authors experimented with low level features based on characters. This technique identifies the top-k similar reports. The authors evaluated their technique on the Eclipse bug repository and obtained a recall rate of about 34% for 1100 randomly selected test cases and 62% for 2170 randomly selected test cases. For each of the cases the proposed technique extracted top-50 similar reports.

The previous techniques considered the bug reports to contain only textual information and processed it in a manner similar to processing English language. Generally, the users who detect the bugs included other information such as patches, stack trace and source code along with the bug reports and this information were generally ignored. N. Bettenberg and R. Premraj [8] developed a technique for extracting and processing this additional information from the bug reports. The authors tested their technique on Eclipse bug reports. All the patches were accurately detected, stack traces were detected with an accuracy of 98.5%, 98.5% of source code were detected and 97% of the enumerations detected. This technique was a breakthrough as it allowed users to extract more information from the bug reports with near perfect accuracy.

C. Sun, D. Lo, S. Khoo and J. Jiang [9] proposed the REP technique of identifying duplicate reports. This technique extracts categorical features from the bug report in addition to the word based textual features and uses the extended BM25F metric [10] to measure the similarity between the reports. The authors compared their algorithm against the ones mentioned in [6] on Eclipse, Mozilla and OpenOffice bug repositories. This technique outperformed [6] and achieved relative improvement in recall rate of about 23% in OpenOffice dataset, 20% on Mozilla dataset and 17% on Eclipse dataset and also there was a significant improvement in runtime. The authors also compared their algorithm against [7] on the exact same dataset used in [11]. This technique achieved an improvement in recall rate of about 51%.

C. Sun, D. Lo, T.T. Nguyen and A.T. Nguyen [12] developed the DBTM approach in tackling duplicate bug reports. This technique leveraged textual and topic based features. The authors constructed a model termed T-Model to represent topic based features. The T-Model is an extension of Latent Dirichlet Association (LDA) [7] which is used to identify reports having similar topics. The authors used the BM25F technique to measure the textual similarity between the documents. The authors combined the T-Model and BM25F approach to identify the duplicate reports. The DBTM technique was compared against the REP [9] technique on Eclipse, Mozilla and OpenOffice dataset and achieved 10-13% improvement in accuracy.

In [13] A. Alipour, A. Hindle and E. Stroulia leverage prior knowledge of software quality and software architecture along with contextual, topic-based and textual features to identify duplicate reports. The authors have used LDA to model and evaluated topic based features and BM25F to measure the similarity between the textual and contextual features. The authors have used various machine learners (classifiers) to identify and retrieve duplicate reports. The classifiers used are 0-R, K-NN, Naïve Bayes, C4.5 and Logistic Regression. This technique was compared against the REP [9] technique on Android bug report dataset. The consideration of contextual features improves the accuracy by about 12%.

In [14], Y. Tian, C. Sun and D. Lo refined the technique proposed in [2] to filter out the incoming duplicate reports. The authors used the technique in [9] to extract textual and categorical features and also used a newer similarity measure namely BM25F to measure the similarity between the reports. They used a SVM classifier similar to the one proposed in [6] to classify the incoming reports as duplicate or non-duplicates instead of the clustering approach used in [2]. This technique could filter about 24% of the incoming duplicates whereas [2] achieved only 8% accuracy. This technique achieves an 160% more accuracy than [2].

R.P. Gopalan and A. Krishna [15] have evaluated the feasibility of clustering in identifying duplicate bug reports. The authors have considered only textual features in identifying duplicates and have used cosine metric to compute the similarity between reports. The bug reports whose cosine values fall within a particular threshold are grouped or clustered together. The authors have used INCLUS [16] algorithm to handle clustering operations. The authors have used TP-Rate, TN-Rate and Harmonic measure to evaluate the performance of this technique on Eclipse, OpenOffice and Mozilla datasets.

In [17], A. Panichella, B. Dit, R. Oliveto, M.D. Penta, D. Poshyvanyk and A.D. Lucia have proposed the GA-IR approach that uses Genetic Algorithm along with Information Retrieval techniques like VSM and LSI to identify duplicate reports. The authors have considered textual features and execution trace of each bug and have used Jaccard and cosine metrics to evaluate similarity between the reports. For implementing Genetic Algorithm, the authors have set crossover probability to 0.8, size of chromosome to 7, uniform mutation with probability 1/7, population size of 50 with elitism of 2 individuals. The genetic

algorithm condition is stopped if the fitness value does not improve for over 10 generations. The GA-IR is compared against the VSM and LSI technique on Eclipse dataset. It achieves a 2-7% greater accuracy for the dataset considering only the title of bug reports and 2-16% better for the dataset considering the title and description.

As mentioned at the beginning there are two approaches to detect duplicate bug reports namely filtering the incoming reports as duplicates or retrieving the top-k similar reports. Only two of the algorithms [2] [14] filter the incoming reports whereas the other techniques retrieve the top-k reports. It can be seen that retrieving the top-k similar reports has higher accuracy in detecting duplicates than filtering the incoming reports. Moreover, retrieving multiple reports provides the developer assigned to fixing the bug with more information about the bug as just one report may not encompass all the necessary details.

Our recommendation is to retrieve the top-k reports even though the idea of filtering sounds more simple and desirable. We would also recommend to fix the value of k anywhere between 5-8. If the k value is too low, the information provided to the developer would be minimum and might not be adequate to understand and fix the bug. If the value of k is too large, some reports which are not duplicates might get extracted and the developer would have to sift through a lot of unnecessary information to find relevant information.

3. GENERAL PROCEDURE

After surveying and reviewing various papers we observed common pattern in detecting duplicate bug reports. In this section, we list out the steps involved and in the subsequent sections we provide a brief overview of the algorithms encountered for each of the procedures. Typically, there are six steps and are as following,

i. **Dataset Acquisition:** The first step involves acquiring the bug repository of a software product which contains duplicate bug reports.

ii. **Pre-processing:** The next step involves pre-processing the acquired data using various techniques to convert it into a suitable format for further processing.

iii. **Feature Selection:** The third step involves extracting necessary features from the pre-processed dataset and also additional features corresponding to topic and contextual information of the bug reports.

iv. **Detecting Duplicate Reports:** The next step involves applying a machine learning/information-retrieval algorithm to model the extracted features to detect duplicates.

v. **Similarity Measure:** These are the various metrics used to measure how similar/dissimilar two bug reports are.

vi. **Algorithm Evaluation:** The final step evaluating the performance of the algorithm in detecting duplicates.

4. DATASET ACQUSITION

The first step in duplicate bug report detection is to have a dataset of bug reports which contains duplicates. In software maintenance, users inform developers via bug reports which part of a software product needs corrective maintenance [18]. A typical bug report contains various fields,

- **Title/Summary:** Contains the title of the bug report and a brief one line description
- **Description:** Describes the bug and how to reproduce it
- **Product:** Software product where the bug was discovered
- **Component:** Product component where the bug was encountered
- **Bug Status:** The attribute indicates the current state of a bug
- **Resolution:** This attribute indicates what happened to this bug.
- **Assigned To:** The identifier of the developer who got assigned the bug
- **Operating System:** In which OS was the bug encountered
- **Priority:** Denotes how soon the bug should be fixed
- **Severity:** Denotes impact of bug on software system
- **Target Milestone:** When a bug is expected to be resolved
- **Attachments:** Developers usually add various attachments to a bug report namely screenshots, patches and stack traces.

During the course of this project we came across the following 5 bug repositories,

- Eclipse
- OpenOffice
- Mozilla
- Sony Ericsson Mobile Communications
- Android

4.1 Sony Ericsson Mobile Communications Bug Repository

Sony Ericsson Mobile Communications develops mobile multimedia devices for GSM and UMTS standards. It is a large company with a complex software development process. Sony Ericsson uses a Software Product Line approach when developing mobile phones. The notion is to have one single platform for a family of phones. They all share this single platform, and upon it, modules are added as needed. While this is an efficient way to develop new phone models, it is also a good source of duplicate bug reports. In [1], the authors have used a subset of the Sony Ericsson bug repository to evaluate their algorithm.

4.2 Android Bug Repository

Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices. There has been continuous version release of the Android OS. Android OS source code is open and bugs faced can be reported by anyone using the OS. Therefore,

Android has a vast bug repository with many duplicate bugs. In [13] the authors used the Android bug reports submitted from November 2007 to September 2012 as the dataset. After filtering out unusable bug reports the authors were left with 37,236 reports out of which 1063 were duplicates.

4.3 OpenOffice Bug Repository

OpenOffice is a leading open-source office software suite for word processing, spreadsheets, presentations, graphics, databases and more. It is available in many languages and works on all common computers. It stores all the data in an international open standard format and can also read and write files from other common office software packages. OpenOffice has a tool called issuezilla which allows users and developers from all over the word to submit bug reports. In [6], the authors used the bug reports submitted in the year 2008 from OpenOffice bug repository which contained 12,732 bug reports among which around 700 were duplicates. In [9] and [12], the authors used the bug reports submitted from the year 2008 to 2010 which were about 31,138 in number among which around 3400 were duplicates.

4.4 Mozilla Bug Repository

Mozilla's most famous product is the Firefox browser. Firefox is a well-known open source web browser written in C/C++ and has an enormous amount of bug reports which contain a significant number of duplicates. In [2], the authors have used the Mozilla bug reports submitted from February 2005 to October 2005 which 29,000 in number. In [5], the authors used the reports submitted from Jan 2004 to April 2004 which were around 1500 in number and 744 were duplicates. In [6], the authors used the Firefox reports from April 2004 to July 2007 which were around 48,000 in number among which 3307 were duplicates. In [9] and [12], the authors used the reports generated in 2010 which were 75,653 in number among which around 7000 were duplicates.

4.5 Eclipse Bug Repository

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages through the use of plugins. Eclipse bug repository is the most widely used repository for duplicate bug report detection. In [5], the authors randomly selected 200 reports from the list of reports submitted in June 2004. In [6], the authors used the Eclipse reports generated in the year 2008 which were around 45,000 in number among which 2013 were duplicates. In [9] and [12], the authors used the reports generated in from 2001 to 2007 which were 209,058 in number among which around 28,000 were duplicates.

4.6 Commentary

We think that there are two improvements that can be made,

- **Fix test dataset**: It is evident from the survey that the dataset varies widely from algorithm to algorithm. Due to this, there is a certain level on uncertainty while comparing the performance of the algorithms. Only [12] and [9] use the same dataset. Our recommendation is to have a standard dataset on which the algorithms can be compared.
- **Dataset from newer open-source projects**: Our recommendation is to evaluate the algorithm on bug reports other open source project like the various Apache projects (Spark, Storm, Ambari, Samsa, etc.), Git, Blender, D3 and many more. In [20], the authors have analyzed 4 open source projects namely Ambari, Camel, Derby and Wicket. This could be used as a starting point for incorporating dataset from newer projects.

5. PRE-PROCESSING TECHNIQUES

In identification of duplicate bug reports, almost all the techniques extract features from the text content of the bug report namely, the description and summary fields. Before the features and be extracted, the text content should be cleaned and represented in a format suitable for feature extraction. Pre-processing generally consists of the following 3 steps,

- Tokenization
- Stemming
- Stop-word removal
- Term Weighting

5.1 Tokenization

Tokenization is the process of breaking a stream of words into text, phrases, symbols or other meaningful elements. In all of the algorithms, tokenization is based on approach mentioned in [19]. The various constraints applied during tokenization are,

- All periods are removed except for the ones occurring in floating point numbers
- Hyphenated words are split into separate tokens
- All other punctuation marks like comma, semicolon, etc. are discarded
- All capital letters are converted to lower case letters

5.2 Stemming:

Stemming is the process of converting each of the identified tokens into their root word. This involves removing prefixes, suffixes and other lexical components of a word. For example, words like working and worked are converted to work.

5.3 Stop word removal:

This step involves removing some of the commonly occurring words from the token list. There are many words like the, and, is, was, etc. which do not carry any specific information and their presence can interfere with similarity measure. Therefore, a list containing the common words also known as stop words are applied to the text. As an alternative to stop word removal, weights are assigned to the words. This technique is called term weighting.

5.4 Term Weighting

In this technique weights are assigned to each individual token. Term weighting is intended to reflect how important a word is to a document in a collection or corpus, in this case we measure how important a word is in a bug report contained in a bug repository. In the techniques described to identify duplicate bug reports, tf-idf [23] is the common measure used to weigh

the terms. Tf-idf stands for term frequency – inverse document frequency.

- TF: It is the count of the number of times a particular word occurs in a document to the total number of words present in the document.
- IDF: It is used to measure the relative importance of a term in a corpus/collection of documents. It is usually the inversely proportional to the frequency of the term in corpus. A commonly occurring word like 'the' has a very low idf value and it indicates that the word is not important.

Tf-idf of a word is the product of its tf and idf values. So, for a word to have high tf-idf value, it should have high frequency in a given document and a low document frequency of the term in the whole collection of documents.

5.5 Commentary

The pre-processing steps especially tokenization and stemming is constant throughout all papers except [7], where character level features are extracted instead of words. In papers [1] and [5] stop word removal technique is used. In the subsequent papers, term weighting (calculation of tf-idf) values is used. This change lead to a significant improvement in accuracy. Hence, our recommendation is not to use stop word removal and use term weighting instead.

6. FEATURE SELECTION

Feature selection is the process of selecting a subset of relevant variables or attributes from the data. In this survey, we encountered the following features,

- Textual
- Character level n-grams
- Categorical
- Topic Based
- Contextual

6.1 Textual Features

Bug reports that are submitted contain two text fields namely summary and description. Textual features are the words extracted from these fields after necessary pre-processing and term weighting. The textual features obtained from the reports are generally represented in bag of words format and similarity is computed between the bag of words of a pair of reports. In all of the papers surveyed, the authors have extracted the textual features from the summary and description fields.

6.2 Character Level n-grams features

N-gram means a subsequence of N contiguous items within a sequence of items. Word n-grams represent a subsequence of words and character n-grams represent a subsequence of characters. In paper [7], the authors have extracted n-grams characters from the description and summary fields instead of words from the bug reports to compute similarity. Extracting character level n-grams has advantages such as handling misspelt words, hyphenated words, matching words to their common root, etc.

6.3 Categorical Features

A bug report has several fields like priority, version, component, product, etc. in addition to the summary and description fields. Categorical features are the information extracted from these various other fields. In [9] five categorical features are selected from the bug reports namely product, component, type, priority and version. For product, component and type their equality is checked between a pair of bug reports and for priority and version the reciprocal between their distances is computed.

6.4 Topic Based Features

Each system or software product is assumed to have a certain number of technical functionalities. Each functionality is considered as a topic represented by a certain words or terms. In [12], the authors have considered such topic based features. They have assigned certain words along with frequencies to a topic and each topic can be represented as a vector of words. In the summary and description fields of the bug report based on the occurrence of certain words each bug report is assigned to a set of topics. A bug report is then represented as a vector of topics and topic similarity between a pair of bug reports is computed by measuring the similarity between the topic vectors.

6.5 Contextual Features

Contextual features of a software products are the features associated with prior knowledge of software quality and software architecture. In [13], the authors have used contextual features in addition to textual, categorical and topic based to determine duplicate bug reports on the Android bug repository. The authors have maintained a list of contextual words related to topics in Android Architecture (Application, Framework, Kernel, etc.), Non-Functional Requirements (Efficiency, Functionality, Maintainability, etc.) and Android Topic words (camera, Bluetooth, browser, etc.). The authors consider each of the word lists as text and compute the similarity between the word list and the textual features obtained from the bug reports.

6.6 Commentary

From the survey, it is evident that extracting only textual features is not sufficient. There was a dramatic increase in performance as when new features were extracted. With textual, categorical, topic based and contextual features the maximum accuracy obtained was around 92% [13]. Extraction of character level features is not necessary as it did not give a great improvement in performance and also it complicates the process of feature extraction. Hence, our recommendation is to extract word based textual features, categorical features, topic based features and contextual features and disregard character level features altogether.

7. INFORMATION RETREIVAL TECHNIQUES

Information retrieval (IR) is the activity of obtaining information resources relevant to an information need from a collection of information resources. IR aims to extract useful information from unstructured documents, most of which are

expressed in natural language. IR methods typically treat documents as "bags of words" and subsequently represent them in a high-dimensional vector space where each dimension corresponds to a unique word or term. During the course of this survey we came across three IR models which have been employed to represent the extracted features. The IR models encountered are,

- Vector Space Model (VSM)
- Latent Semantic Indexing (LSI)
- Latent Dirichlet Allocation (LDA)

7.1 Vector Space Model

Vector Space Model [24] is an algebraic model which is used to represent text documents as vectors of identifying keywords or terms so that these vectors can be used to filter, retrieve, index and rank these documents based on queries. The model considers a multidimensional space where the documents are conceptualized as vectors. The query is treated just like a document and considered to be a vector in the multi-dimensional space. The terms housed in the documents and queries have weights allocated using a weighing scheme, which are used to assign coordinates for the document in the space. The query is then matched to a document by matching the query terms to the terms found in the document. But this model assumes term independence which can be brought about by synonymy. This is where the model fails. In identification of duplicate bug reports the textual features obtained from the summary and description fields is represented in the Vector Space Model. This model is used in all the papers encountered in the survey.

7.2 Latent Semantic Indexing (LSI)

LSI [25] improved on VSM by considering the documents to be categorized under knowledge domains. LSI makes use of synonyms to club words together to be under a specific domain. So, the keywords in the document and the count of occurrences of those keywords are analyzed to calculate the semantic distances between two documents. If the documents contain more words that fall under the same knowledge domain, their semantic distance is lesser otherwise the documents are semantically distant. In [17], the authors have compared their GA-IR technique against LSI to identify duplicate bug reports.

7.3 Latent Dirichlet Allocation (LDA)

LDA [26] is a generative statistical model which uses unobserved groups of words to explain the occurrence of observed sets of words in a document. LDA works on the concept of topic modelling by defining that the occurrence of certain words in the document is the result of presence of data on that topic in that document. LDA considers documents to be a mixture of topics, and also categorizes words to be parts of certain topics. This categorization of words into topics and search of count of words allows LDA to categorize documents into topics or domains. In [12], the authors extract topic based features and have used LDA to model the topic based features since LDA is tailor made for topic modelling.

7.4 Commentary

In almost all of the papers VSM has been used to model the textual features. Since LSI is an improvement over VSM, we think that the performance might improve if LSI were used. LDA was specifically meant to handle topics and they are perfect for topic based features. Our recommendation is to stick with LDA for topic based features and compare the performance of LSI and VSM for textual features and then choose the better one.

8. MACHINE LEARNING TECHNIQUES

Machine learning techniques employ classifiers or discriminative models to identify duplicate bug reports. Classifier is an algorithm or mathematical function that maps input data to a category (in our case it classifies bug reports as duplicates/non-duplicates). In the papers surveyed various machine learning algorithms have been used to identify duplicates. A brief overview of the following algorithms is provided in the upcoming sub-sections,

- Support Vector Machine
- 0-R
- Logistic Regression
- Naïve Bayes
- C4.5
- K-NN clustering
- INCLUS clustering

8.1 Support Vector Machine

In machine learning, support vector machines [27] are learning models that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. In [6], the authors have trained their SVM classifier with textual features extracted from bug reports in pairs of <duplicate, duplicate> and <duplicate, non-duplicate> reports. Any incoming bug report is classified as duplicate or non-duplicate by using the trained SVM classifier. After the incoming report, has been classified, the SVM classifier is re-trained with the new data. The only problem with SVM is the runtime.

8.2 0-R

0-R is the simplest classification algorithm which relies on the target (duplicate/non-duplicate) and ignores all other factors. It simply predicts the majority class. It is generally used for determining a baseline performance as benchmark for other classifiers. In [13], the authors have used 0-R as the benchmark classifier and have evaluated the performance of other classifiers against this. The selection of features has no impact on the 0-R classifier. The authors have evaluated the performance of 0-R under 2 conditions namely considering only textual and categorical features and considering textual, categorical and contextual features. In both the cases 0-R performs the same and is 80% accurate in detecting the duplicate reports.

8.3 Logistic Regression

Regression analysis is a machine learning technique used for estimating the relationships among variables. It includes techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables. Logistic regression is a regression model where the dependent variable (DV) is categorical. In duplicate bug report detection, binary dependent variables are handled that is duplicate and non-duplicate. In [13], the authors have used this technique to identify duplicate reports. This technique gives an accuracy of around 82% when textual and categorical features are considered and 88% when textual, categorical and contextual features are considered.

8.4 Naïve Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. In the duplicate bug report detection problem, the class variables are duplicate and non-duplicate. In [13], the authors have used this technique to identify duplicate reports. This technique gives an accuracy of around 78% when textual and categorical features are considered and 79% when textual, categorical and contextual features are considered. This algorithm performs worse than the baseline 0-R algorithm in identifying duplicates.

8.5 C4.5

C4.5 [28] is a decision tree based algorithm. A decision tree is a decision support tool that uses a tree-like graph to model decisions and their possible consequences. Decision tree learning uses a decision tree as a predictive model which maps observations or features about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). C4.5 builds decision trees from a set of training data using the concept of information entropy. At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sub-lists. In [13], the authors have used this technique to identify duplicate reports. This technique gives an accuracy of around 85% when textual and categorical features are considered and 92% when textual, categorical and contextual features are considered. This algorithm performs the best for duplicate bug report detection.

8.6 K-NN Clustering

In machine learning, the *k*-Nearest Neighbors algorithm (or *k*-NN for short) is a non-parametric algorithm used for classification. For every vector, the k-nn algorithm calculates the k-nearest neighbors for that input vector. In [13], the authors have used this technique to identify duplicate reports. This technique gives an accuracy of around 82% when textual and categorical features are considered and 91% when textual, categorical and contextual features are considered.

8.7 INCLUS Clustering

INCLUS [16] is an incremental clustering algorithm. It is used for high dimensional sparse transactional data and it uses a newly defined similarity measure and a notion of cluster representatives based on locally frequent items of each cluster. INCLUS seeks structures in transactional data with respect to the support and similarity constraints specified by the users. In [15], the authors have considered only textual features in identifying duplicates and have used cosine metric to compute the similarity between reports. The bug reports whose cosine values fall within a particular threshold are grouped or clustered together using the INCLUS algorithm.

8.8 Commentary

The authors have used well known Machine Learning algorithms to detect duplicate reports. From this survey, it seems like C4.5 is the best classifier for duplicate bug report detection but it cannot be conclusively ascertained because most of the classifiers have been tested on Android dataset with contextual features. Our recommendation is to test the performance of various classifiers on different datasets to obtain a conclusive result. We also suggest to stick with 0-R as the baseline classifier because of its simplicity and to discard any algorithm that performs worse than 0-R (Naïve Bayes in case of [13]).

9. SIMILARITY MEASURE METRICS

To identify duplicate documents, it is important to measure the similarity between documents. There are various similarity metrics that can be used to indicate how similar a given pair of documents are. Throughout the course of this survey we have encountered three standard similarity metrics,

- Cosine Similarity
- BM25F

The other techniques not mentioned above employ their own method of similarity measurement which is not a standard metric.

9.1 Cosine Similarity:

In this metric, the similarity between two bug reports represented in Vector Space Model is calculated. In Vector Space Model, each bug report, is represented in a bag of words format, which is a vector of all the terms present in the bug repository. Given the vector representation of words in 2 documents, the formula for cosine similarity between them is,

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

Higher the cosine value, more similar the documents are. This similarity metrics has been used in [1], [2], [5], [17] and [15].

9.2 BM25F:

BM25F is an advanced document similarity function based on weighted word vectors of documents [22]. BM25F computes the similarity between a query and a document based on the common words that are shared between them. It ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document. TF and IDF measures are used while calculating the BM25F score of a document. The formula for computing the BM25F score to identify whether two given documents, document d and a document q are duplicates is as follows,

,

$$BM25F(d,q) = \sum_{t \in d \cap q} IDF(t) \times \frac{TF_D(d,t)}{c + TF_D(d,t)}$$

This similarity metric has been used in [12], [9], [6] and [13]

9.3 Commentary

In a recent research conducted [21], cosine similarity was proved to be the worst performing similarity measure. Hence, it is a surprise that papers published recently [17] still use cosine similarity measure. Cosine similarity measure can be used for initial analysis but as the algorithms evolve it is better to switch to BM25F similarity measure.

10. PERFORMANCE MEASURE METRICS

There were many performance metrics encountered during the course of the survey and they were all confusion matrix based metrics. The metrics used are as follows,

- True Negative Rate
- Recall
- Precision
- Accuracy
- Harmonic

10.1 Confusion Matrix based Metrics

The use of confusion matrix is very prominent in data science and machine learning. All the papers surveyed used at least one confusion matrix based metric. A confusion matrix allows visualization of performance of algorithm. The structure of a confusion matrix is as follows,

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

With respect to automated detection of duplicate reports, the definitions of various terms in the confusion matrix is as follows,

- **True Positive (TP):** It is the number of reports that have been correctly identified as duplicates
- **False Positive (FP):** It is the number of reports that have been identified as duplicates but are actually not duplicates.
- **True Negative (TN):** It is the number of reports that have been correctly identified as non-duplicates.
- **False Negative (FN):** It is the number of reports that have been identified as non-duplicates but in reality, they are duplicates.

There are various metrics that can be obtained from the confusion matrix values. Some of the metrics encountered during the survey are,

- **Accuracy:** It is the measure of the overall correctness of the algorithm. It is the ratio of the sum of TP and TN to the total number of reports. Accuracy was used in [12], [13] and [17] to evaluate the performance of the algorithm.
- **Recall:** Recall is a measure of, if predicted duplicate how often it is correct. It is the ratio of TP to the sum of TP and FP. This is the most common measure encountered in the survey. Papers [1], [5], [9], [7], [2] and [15] use recall.
- **Precision:** It gives an estimate of the actual number of duplicates identified. It is the ratio of the cardinality of TP to the sum of TP and FN. This measure is used in [2] and [15].
- **True Negative Rate:** It is defined as the ratio of the number of non-duplicate bug reports less the number of false positives as identified by our algorithm to the actual number of non-duplicates. It is the ratio of TN to the sum of TN and FP. This metric is used in [15].
- **Harmonic:** This metric is used to balance the tradeoff between precision and true negative rate. The formula for harmonic is,
  Harmonic = (2*Precison*TNRate)/(Precision+TNRate)
  This metric is used in paper [15].

10.2 Commentary

Confusion matrix based metrics are the most widely used for machine learning algorithms. Hence, it is no surprise that all the papers have used at least one of the confusion metrics. Different papers have used different metrics and it becomes cumbersome to compare the performance of different techniques. Our recommendation is to stick to a standard metric probably recall or accuracy and then compare the performance of algorithms. This would greatly improve the readability and understandability of the various techniques.

11. THREATS TO VALIDITY

In this section, we give a brief overview of some of the future problems that could be faced.

All the papers have assumed that the bug reports would have the description and summary fields and have extracted textual features from them. While extracting the categorical features, the authors have assumed that the field names would be constant forever and never change. Change in any of the field names can produce unpleasant results. Hence, there is a need to generalize the extraction of features from the bug reports without

dependence on field names. With respect to context based features, the authors have maintained a list of context specific words. There is a high probability that the context specific words might change in the future versions of the product. A technique should be developed such that the context list can be extracted from the technical documents associated with the product.

In all the experiments, only subsets of bug reports in the original bug repositories. This factor may be a threat to the external validity, as the experimental results may be applicable only to the selected bug reports but not to the entire bug repositories.

The main advantage of extracting the top-k reports is for the developer to obtain more information about the bug to fix it. There is a possibility that additional information can mislead the developer and distract the developer from the root cause of the problem. Techniques should be developed to ensure that the developers do not get overwhelmed by the information overload and can find relevant information to fix the bug.

In almost all the algorithms, the authors have used a machine learning technique to identify duplicate bug reports. The authors have trained their classifier with pairs of duplicate and non-duplicate reports. The training data is triaged manually. The authors assume that the training data is completely valid. No room for handling training data error is mentioned in any of the algorithms.

12. CONCLUSION AND FUTURE WORK

Automated bug triaging has made tremendous progress in the past few years. It has come a long way from considering only textual features in the summary and description fields of the bug report to considering contextual features. However, there is still a long way to go and considerable room for improvement.

First and fremost, there should be a standard dataset on which all the algorithms can be tested. In some of the papers, the authors have just evaluated their algorithm on a dataset size of just 200 which is not at all sufficient. Introducing a standardized dataset would make it much simpler to compare the performance of various algorithms.

The authors have filtered invalid bug reports and completely neglected them. The authors could spend more time on analyzing the invalid bug reports and try to reproduce the bugs. This would ensure that the duplicate bug report detection system is fully functional.

The authors should perform an initial survey of testers, developer and users before moving onto detecting duplicate bug reports. The survey would give the authors a clear idea about bug reporting habits. This human behavior factor could be incorporated into duplicate detection algorithms to detect duplicate bug reports with higher accuracy.

Only for Android dataset, the contextual features are analyzed. This technique should be extended to the Sony, Mozilla, Eclipse and OpenOffice bug reports repository. Moreover, the authors should incorporate bug reports from newer open source projects.

Almost all the papers completely ignored time efficiency while analyzing the performance of their algorithm. The size of the bug repository increases exponentially every day. Hence, it is necessary for the algorithm to detect duplicates with good accuracy in a limited period of time.

Even though there is a technique to extract source code, patches and screenshots from the description field [8], very few algorithms have leveraged this functionality. Source code, stack traces and patches contain valuable information about a bug and should not be ignored. Algorithms should be modified to handle the above-mentioned information.

Techniques need to be developed that can obtain information from images. The recent trend is to submit a screenshot of the bug instead of manually filling up a bug report. Image processing techniques need to be incorporated in feature extraction algorithm to extract relevant features from the screenshot.

13. REFERENCES

[1] Runeson, P., Alexandersson, M., & Nyholm, O. (2007, May). Detection of duplicate defect reports using natural language processing. In *29th International Conference on Software Engineering (ICSE'07)* (pp. 499-510). IEEE.

[2] Jalbert, N., & Weimer, W. (2008, June). Automated duplicate detection for bug tracking systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)* (pp. 52-61). IEEE.Fröhlich, B. and Plate, J. 2000.

[3] Hooimeijer, P., & Weimer, W. (2007, November). Modeling bug report quality. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (pp. 34-43). ACM

[4] Mishra, N., Schreiber, R., Stanton, I., & Tarjan, R. E. (2007, December). Clustering social networks. In *International Workshop on Algorithms and Models for the Web-Graph* (pp. 56-67). Springer Berlin Heidelberg.

[5] Wang, X., Zhang, L., Xie, T., Anvik, J., & Sun, J. (2008, May). An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering* (pp. 461-470). ACM.

[6] Sun, C., Lo, D., Wang, X., Jiang, J., & Khoo, S. C. (2010, May). A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1* (pp. 45-54). ACM.

[7] Sureka, A., & Jalote, P. (2010, November). Detecting duplicate bug report using character n-gram-based features. In *2010 Asia Pacific Software Engineering Conference* (pp. 366-374). IEEE.

[8] Bettenburg, N., Premraj, R., Zimmermann, T., & Kim, S. (2008, May). Extracting structural information from bug reports. In Proceedings of the 2008 international working conference on Mining software repositories (pp. 27-30). ACM.

[9] Chengnian Sun, David Lo, Siau-Cheng Khoo and Jing Jiang. 2011. Towards More Accurate Retrieval of Duplicate Bug Reports. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering.

[10] Robertson, S., Zaragoza, H., & Taylor, M. (2004, November). Simple BM25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management* (pp. 42-49). ACM.

[11] Čubranić, D. (2004). Automatic bug triage using text categorization. In *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*.

[12] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N. Nguyen, David Lo and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012).

[13] Alipour, A., Hindle, A., & Stroulia, E. (2013, May). A contextual approach towards more accurate duplicate bug report detection. In Proceedings of the 10th Working Conference on Mining Software Repositories (pp. 183-192). IEEE Press.

[14] Tian, Y., Sun, C., & Lo, D. (2012, March). Improved duplicate bug report identification. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on* (pp. 385-390). IEEE.

[15] Gopalan, R. P., & Krishna, A. (2014, April). Duplicate bug report detection using clustering. In 2014 23rd Australian Software Engineering Conference (pp. 104-109). IEEE.

[16] Li, Y., & Gopalan, R. P. (2006, May). Clustering high dimensional sparse transactional data with constraints. In *2006 IEEE International Conference on Granular Computing* (pp. 692-695). IEEE.

[17] Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., & De Lucia, A. (2016, March). Parameterizing and Assembling IR-based Solutions for SE Tasks using Genetic Algorithms. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) (Vol. 1, pp. 314-325). IEEE.

[18] Bettenburg, N., Premraj, R., Zimmermann, T., & Kim, S. (2008, September). Duplicate bug reports considered harmful… really? In *Software maintenance, 2008. ICSM 2008. IEEE international conference on* (pp. 337-345). IEEE.

[19] Minnen, G., Carroll, J. and Pearce, D. “Robust, Applied Morphological Generation”, Proceedings of the First International Natural Language Generation Conference, Mitzpe Ramon, Israel, pp. 201-208, 2000.

[20] Ohira, Masao, et al. "A dataset of high impact bugs: manually-classified issue reports." *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015.

[21] Whissell, J. S., & Clarke, C. L. (2013, October). Effective measures for inter-document similarity. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management* (pp. 1361-1370). ACM.

[22] Zaragoza, H., Craswell, N., Taylor, M. J., Saria, S., & Robertson, S. E. (2004, November). Microsoft Cambridge at TREC 13: Web and Hard Tracks. In *TREC* (Vol. 4, pp. 1-1).

[23] Aizawa, A. (2003). An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, *39*(1), 45-65.

[24] Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, *18*(11), 613-620.

[25] Hofmann, T. (1999, August). Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 50-57). ACM.

[26] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, *3*(Jan), 993-1022.

[27] Suykens, J. A., & Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, *9*(3), 293-300.

[28] Quinlan, J. R. (1996, August). Bagging, boosting, and C4. 5. In *AAAI/IAAI, Vol. 1* (pp. 725-730).