A Systematic Study of Duplicate Bug Report Detection

Som Gupta[1]
Research Scholar
AKTU Lucknow

Sanjai Kumar Gupta[2]
Associate Professor
Computer Science Engineering Department BIET Jhansi

***Abstract*—Defects are an integral part of any software project. They can arise at any time, at any phase of the software development or the maintenance phase. In open source projects, open bug repositories are used to maintain the bug reports. When a new bug report arrives, a person called "Triager" analyzes the bug report and assign it to some responsible developer. But before assigning, has to look if it is duplicate or not. Duplicate Bug Report is one of the big problems in the maintenance of bug repositories. Lack of knowledge and vocabulary skills of reporters sometimes increases the effort required for this purpose. Bug Tracking Systems are usually used to maintain the bug reports and are the most consulted resource during the maintenance process. Because of the Uncoordinated nature of the submission of bug reports to the tracking system, many times the same bug report is reported by many users. Duplicate Bug Reports lead to the waste of resources and the economy. It creates problems for triagers and requires a lot of analysis and validation. Lot of work has been done in the field of duplicate bug report detection. In this paper, we present the researches systematically done in this field by classifying the works into three categories and listing down the methods being used for the classified researches. The paper considers the papers till January 2020 for the analysis purpose. The paper mentions the strengths, limitations, data set, and the major approach used by the popular papers of the research in this field. The paper also lists the challenges and future directions in this field of research.**

***Keywords*—AUSUM; feature-based; deep learning; semantic; unsupervised**

I. Introduction

Bug Report is one of the artifact which is produced during the software development, testing and the maintenance phase of the software process. It has been found that in any normal software development process, maintenance phase accounts for about two-thirds of the total efforts [1]. Mainly bug reporting systems are used for maintaining the bug reports of large software projects like Bugzilla. Nowadays, with the increasing competition and rapid development with quick time to release, it is common among the software community. But this quick time to release also leads to a lot of issues and the remaining features that make the users submit their expectations and issues. Also, these bugs lead to the release of another version of the software. As software defects add to the cost to the testing process. It is very important to detect them as soon as possible. As the software teams are usually geographically distributed, for the collaboration purpose web-based systems are used [2]. Bug Tracking Systems also allows users to report the bug they encounter. It has been found that finding whether the bug report is duplicate or not is more expensive than creating the new bug report.

In a paper by [3] they performed an exploratory study of 8 open-source projects and 1 private company project to analyze what percentage of the corpus consists of the duplicate bug reports, how much time a submitter spends in identifying the duplicate bug report before opening the new bug report, how the time is spent in resolving the duplicate bug report and the valid bug report, the average frequency of duplicate bug reports coming up in the repository on daily basis, how much vocabulary the master and duplicate bug report shares, how the type of bug report impacts the frequency of duplicate bug reports and why the duplicate bug report problem arises. They found that the submitter productivity is impacted by the duplicate bug reports problem as many times almost 48 man-hours need to be spent daily for performing this activity. The same study was later conducted by [4] where their focus was to predict the effort required by developers to identify the duplicate bug reports. They used Peers, Workload, Bug Report, Working Habits, and Triager experience to predict their effort. In another paper by Xie et al. [5], they also analyzed the percentage of duplicate bug reports in various popular platforms like Mozilla Core, Firefox, Thunderbird, Eclipse Platform, JDT, Hadoop, etc. They found that for these popular projects, on an average 12 percent bug reports are duplicate. The statistics of nine famous projects in terms of bug reports and the number of duplicate bug reports are mentioned in Table I. From the table, it is clear the duplicate bug reports comprises of almost 20 percent of the total bug reports.

Because of the following issues, the duplicate bug reports exist in a large number in these bug tracking systems.

1) free availability to the users to report the defect
2) Millions of users exists for the large projects
3) Bug Reports are submitted by both the developers and the users
4) Frequent release of software versions
5) User Inexperience
6) Poor Search Functionality in the Bug Tracking Systems

Many times even if the bug report is duplicate, it gets attached to the master report which is also known as triaging.

Bug Triaging is a process in which the person assigns the bug to a particular person for the resolution. But before assigning, has to go through these bug reports to find which one is duplicate and which one is not. Along with this, the triager has to look whether the bug report is related to the problem or the modification to the software. Because if the same problem is assigned to more than one person, it leads to

TABLE I. DATASET INFORMATION [6]

Project Name	No of Issue per day	Unique Components	No of Duplicates	Duplicate Percentage
Mozilla Core [7]	33	130	44691	21.8
Firefox22		52	35814	30.9
ThunderBird	23	7	12501	30.9
Eclipse Platform [8]	19	21	14404	16.9
JDT [9]	10	6	7688	17
Cassandra [10]	4	24	2083	14.8
MapReduce [11]	2	63	977	13.9
HDFS [12]	3	71	1659	13
Spark [13]	8	29	3077	13.6

the waste of resources. It has been found that many times more than 300 [14] bugs on a day are reported. Not only this but duplicate bugs are also there. This makes the finding of a duplicate bug report a challenging task. They lead to redundancy in the work and thus increase the workload for engineers. For a large project, bug triaging increases the cost to software maintenance, and manually doing this process is almost impractical. Thus there is a need to find the automatic approach which can help find the duplicate bug reports among all.

Because of the above-mentioned problems, need for assistance in detecting duplicate bug reports eases the task of bug triager. In a paper by [15], they observed that detecting duplicate bug reports consume a lot of time. Even though lot of approaches have been proposed but getting the full score for precision is very difficult. The reasons for these results is the diversity of natural language text in the bug reports and the diversity of the features in large software. The poorly written content in the bug reports by the users also poses another challenge to the triager.
Many authors [16] claim that the duplicate bug reports also help find more information. Thus many have proposed the techniques which finds the top-k most duplicate bug reports for the triager to have the additional information also. From our survey we have observed that following techniques are very popular among the research community in this field:

1) Natural language based Researches [17] [18]
2) Execution Trace Based Researches [19]
3) Clustering based models with similarity measures[14] [20]
4) Classifier-based machine learning approach [2]
5) Convolutional and LSTM Based Deep Learning Models [1]

In most of the projects, the first step to further process the bug report for any analysis is feature vector representation. For the Feature vector representation following natural language processing techniques are mainly used by various researchers: Word Embedding, ,Bag of Words, Skip-Gram, CBOW (Continuous Bag of Words), Latent Semantic Analysis, Latent Dirichlet Analysis, N-Gram.

On the Basis of above based information, we divide our work into three parts mainly Information-Retrieval, Machine Learning and Natural Language Processing based, according to the approach being used.

The main contributions of the paper are as follows:

- The paper presents the classification of duplicate bug report approaches.
- Paper gives a brief overview to almost all the famous researches in this field till early 2020.
- Paper gives the comparative results of various approaches being used in this field.
- Paper also mentions the various data-sets which have been used by various researchers.
- Paper also mentions the Strengths and Limitations of the works.
- Paper also discusses the Evaluation Measures being used for various approaches.
- Paper also lists down the challenges and the directions for future research.

The paper is organized as follows: Section 2 gives a brief overview of existing approaches which is further divided into Information-Retrieval Based, Natural Language Processing Based and Machine Learning Based. Section 3 discusses the Methodology that we opted for carrying out this research. Section 4 discusses the various evaluation measures being used for measuring the efficiency of Detecting duplicate bug reports detection techniques. Finally we discuss the future directions in this field and the conclusion of the paper.

II. EXISTING APPROACHES

Bug Report is an artifact which is produced during the software testing or software maintenance phase. It contains three types of data in it:

- Textual Data: It consists of Title, Description and comments. Title is a brief introduction about the bug. Description is the detailed overview along with the steps to reproduce to clearly indicate about the bug. Comments are the text that the other users or developers write against the bug in order to either given a suggestion or improvement or solution.
- MetaData: It consists of miscellaneous information about the bug in terms of its classification like product name, component field, assignee, version, priority, reporter, create time, status and resolution. Component refers to the component of the project where the bug has occurred. Status means whether it is duplicate or not. Status means whether the bug report is “open”, “in-progress”, “resolved”, “closed”, “reopen”, “wont fix”, “not a bug”, etc. Version means in which version of the software, the bug has occurred.
- Attachment: It is the screen shot of the issue in order to give better clarity about the bug report. It may also include the execution trace.

Bug Reports not only contains the resolution for the bug but also enhancements, ideas and the change requests. Analyzing bug reports help in change management, software evolution, traceability and also during effort estimation [3]. Many researchers observed that around 10 to 30 percent of the bug reports are duplicate in these open bug repositories. Having no duplicate bug report is also a big problem as it will lead to less information. In research by [16], he observed that duplicate bug reports help improve the knowledge by adding more information bug having large number of duplicate bug reports creates problem.

Many researchers [2] stated that duplicate bug report literature can be classified into two categories: one which finds the relevant bugs and one which aims to find the duplicate bug report. Thus the process can be classified into:

- Prevention of duplicate bug reports while submitting.
- Identification of top-k bug reports during bug report triaging.

Even though the approaches are classified into the mentioned two categories but in a paper by [21] they mentioned that the former approaches can only delete 8 percent of the duplicate bug reports but 92 percent bug reports still remains in the system. Thus leading to no such good benefit to the triager. More than this preventing the duplicate bug reports is a very expensive process also.

We observed the first main research in this field started with [17] where they used the log based weighing techniques along with the NLP Based approaches to identify the duplicate bug reports. The work was extended by [19] where along with the natural language text, execution traces were also used. The shortcomings of these approaches were taken into consideration by [22] where they used SVM model after creating the features using NLP techniques. In the same year [18] used the n-grams for the same purpose. In 2010, [21] used the BM25F to calculate the correlation between the bug reports. In year 2012 the same work of [21] was extended by [23] to calculate the effectiveness of BM25F for duplicate bug report detection. After these works, we observed that most of the works involve deep learning techniques for the same.

By analyzing all the works, we classified the approaches into following categories:

A. Information-Retrieval based

Information Retrieval techniques aim to find out the structure from the unstructured data. IR Based Models are generally divided into two approaches: word-based and topic-based models. Vector Space Model (VSM) is one example which gives the weights to the words and helps identify the main themes out of the document. Mainly Logarithms, IDF and Entropy are common weighing models which are used with these word-based models.

Topic models are one of the very famous approach which is used in IR field to find the latent topics of the text. They help find the similarity scores between the documents. Topic models help capture the semantic information from the bug report. It includes approaches like Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA), Random Projections and their variants. In topic-based models, the term-document distribution and the word-topic distribution are obtained. These document-topic representation are converted into the vector form. Few hyper parameters are used with the LDA to get the cluster of relevant topics. For those hyper-parameters setting, many researchers have used techniques like Genetic Algorithms, Differential Evolution, Particle Swarm Optimization, Simulated Annealing, Random Search. Similarity between documents is usually calculated by using cosine similarity. General Steps which are used for using Information Retrieval Techniques are: first the pre-processing of the document to extract the relevant words from the document. This step includes stemming and removal of stop words. Then either the Term-by-document matrix or the probabilistic models are generated which are then used to calculate the textual similarities [24]. Term-by-Document matrix includes vocabulary which is also referred to as Terms as rows and the documents as the columns. Weights are assigned to the terms in the documents by either TF-IDF or their variants.

The author in [19] used the Vector Space Model to transform the execution trace into the vector form for getting the information from the execution trace of a bug report for duplicate bug report detection. The author in [25] used the VSM in their information retrieval module in the JDF server to represent the document for the further similarity calculation.

The author in [26] compared the word-based model VSM with the topic-based model over the Firefox and Eclipse Project. They also analyzed how different weighing models work with the VSM. They found that for the task of duplicate bug report identification, VSM models outperformed the Topic-Based models and Log-Entropy outperformed among the weighing schemes.In the paper of [26], they compared the Vector Space Model with the topic models and observed that the Vector Space Model was performing better than the topic models.

The author in [2] in their paper, used the LDA along with the gibbs sampling to extract the topics. They used the comments to find whether the two bug reports are same or not. Their notion was that many times when the title and description are vague, comments supplement the information and help find out the similarity of bug reports. First they classified the comments into useful and un-useful. For the classification of comments, they used SVM along with the RBF kernel. For the the classification, they used four features namely IDF, bug status, length of comment and unigrams to create the vectors for training purpose. They found the weighted comment similarity to find the similarity between the comments and the bug report title and description. They used LDA topics along with the gibbs sampling to find the main topics of the bug report. Then they used topic distance models like Jensen-Shannon and symmetric KL-Divergence to find the distance between the topic distributions. They also used the knowledge about the issuing author to determine the duplicate bug report. Their notion behind checking the issuing author is that because of some specific way of writing, there can be high similarity between the two bug reports from the same author but chances of being duplicate will be very less.

The author in [27] used the information retrieval model to find the top-k similar bug reports for BlackBerry Systems.

They first preprocessed the bug report then important features were extracted for indexing purpose. The indexing was specified in terms of Term-Vectors. Indexed data was fed to searching and ranking module. They used the Lucene features for finding the similar documents. The ranked documents were then fed to the selection and filtering module to find the top-k similar bug reports.

The author in [28] used the topic model LDA implemented in MALLET along with the machine-learning based approaches to identify the duplicate bug report. In their paper, they first paired the bug reports using metrics like difference of words in the summary and description respectively, number of shared words between the summaries and descriptions respectively, number of shared identical topics between summary and description, priority field, time between two submissions, component field, type of bug report severity, etc. In order to find these metrics, they used LDA with alpha value as 50 and beta value as 0.01 in the Weka.

The author in [29] used vector space model to detect the duplicate bug reports by considering the title and the description.

The author in [30] used the topic-based model LDA for identifying the topics from the bug report which were further used for the feature vector construction for the purpose of training the model. The author in [31] along with the NLP used the SVM to analyze whether the Expected Behavior and steps2reproduce are missing or not. They also trained their model with 10 fold SVM after finding the discourse patterns, N-grams and Part Of Speech.

The author in [32] considered the domain-specific information of the bug report to identify the duplicate bug reports. They utilized the power of IR techniques along with the contextual keywords which includes architectural words, non-functional words, topic words extracted using LDA and labeled LDA, and random dictionary words.

The author in [33] classified the duplicate bug reports as one which shares the vocabulary and one which does not share the vocabulary. They proposed an approach which deals with both type of duplicate bug reports. For the first type of bug reports where the vocabulary is shared, they used the Vector Space Model and Clustering techniques. They used the cross product between the vectors and the vector lengths were computed using eucledian distance. They used the concept that smaller the angle between the two vectors, higher is the similarity. They used the TF-IDF weighing scheme to assign the weight to each term of the vector. For clustering, they used K-Means technique. For finding the dissimilar bug reports, first they identified the dissimilar duplicate bug reports by vector space model with TF-IDF where they analyzed the angle between the bug reports. After finding dissimilar reports, they created the word co-occurrence model. After creating the co-occurrence model, they computed the similarity using BM25 and language modeling techniques like Jelinek-Mercer smoothing and Dirichlet smoothing.

The author in [34] in their mentioned how powerful the word embeddings and the LDA approach is while calculating the similarity between two documents. [35] used the LDA and LSI approaches to find how continuously querying the bug report like how it happens in Google Search Engine helps find the duplicate bug reports.

The author in [24] in their paper compared five meta-heuristics GA, DE, Particle Swarm Optimization, Simulated Annealing and Random Search, to analyze how the LDA works when applied.

B. Natural Language Processing based

The author in [17] used the natural language text of bug reports, performed the pre-processing by tokenization, stemming and stop word removal, and then converted the text into bag of word models and modeled as feature vectors. For the features calculation, Term frequency was used. Similarity of the bug reports is calculated by calculating the similarity between the features vectors by using cosine, Jaccard and dice similarity measures. The word was extended by [19] where they used the execution information along with the natural language information and in the bug report to compare it with the other reports. For the feature vector construction, they used Inverse Document Frequency (IDF) along with the Term Frequency (TF). They used the cosine similarity measure to calculate the similarity between the feature vectors. They detected the top-k similar bug reports with their approach.

The author in [36] used the event-component similarity approach to find the duplicate bug reports for the bugs having their component as GUI. In their approach they transformed the report into event, component and requirements. They used the Longest Common Subsequence approach for estimating the similarity between the two event, component and requirements.

N-Gram models have been used to analyze the title and the description of the bug report. The author in [18] et al. have used n-gram characters to identify the top-N similar bug reports. They used the number of shared n-gram characters and the number of character-n grams of title present in the other report's description to find the similarity between the two bug reports.

Extending this work, [25] used the above approach to create a tool for Jazz bug repository to identify the duplicate bug reports. They used Natural Language Based Similarities(NL-S) and Execution-information based similarities(E-S) too calculate the similarity of new bug report with the existing bug reports. The author in [21] extended the concept of BM25F to calculate the textual similarity accurately. For BM25F, they used the property of weight of terms in the query. The author in [37] used the bag of diagrams of title and description along with the extension of BM25F to compute the similarity to find the top-k similar bug reports. The author in [38] used the N-grams to find the scores for 25 features for the bug text for finding the semantic similarity between the two texts.

The author in [39] also used the features and textual similarity to help assist in the duplicate bug report detection while writing the bug report so that user also does not continue writing the bug report. In their approach they have used some fixed amount of word to calculate the features. In their paper, they first pre-processed the titles and descriptions by picking only few important words from them. Then they chosen few bug reports which are non duplicate and few which are duplicate for creating the training set. TakeLab tool is used to calculate the features of the bug report. And then using machine learning classifier approach, they detect whether the bug report is duplicate or not.

The author in [32] used the textual, categorical and contextual similarity measures by calculating the similarity between the title, description, product feature, component feature, bug type, priority and version to find the whether the bug report is duplicate or not. For comparison, cosine similarity measure is being used by them. They used the dictionary having the architectural words which they extracted from project documentation, non-functional requirements which they classified into they efficiency, functionality, maintainability, portability, reliability and usability; LDA-topic words which they extracted by using Vowpal Wabbit online learning tool; and Random English words. For the comparison purpose, they used seven comparison features which includes BM25F with unigram and bigram characters, product, component, difference in versions and priorities, and type of bug report which includes enhancement or defect.

The author in [31] used the discourse based analysis of observed behavior, expected behavior, steps2reproduce to analyze the semantic and syntax of the bug report. They used the part of speech tagging and dependency parsing for finding the discourse patterns. They used Stanford CorenNLP to perform these tasks at the sentence and the paragraph level.

The author in [40] proposed an approach which they called as DURFEX where they used the feature extraction technique by analyzing the stack trace and converting them to a particular format.N-Grams were used to map the stack traces to generate feature vectors. TF-IDF was used to generate the package names. Along with the stack trace, they alsoo used severity and component information of bug report to identify if the bug report is duplicate or not.

The author in [41] used the Longest Common Subsequence approach to find out the most similar substring between the two bug reports to create the feature vector for bug report. The author in [42] used the Hidden Markov Models (HMM) to classify the bug reports as duplicate or not. Their approach included first identifying the stack traces from bug reports. These stack traces were splitted into the duplicate groups. Each group was then separately trained by using HMM. The incoming bug reports were then compared with the trained HMM models to find the scores.

C. Machine Learning based

Machine Learning techniques involve the supervised, unsupervised and semi-supervised approaches. Supervised approaches involve the use of classifiers to either predict binary class problems or multi-class problems. Unsupervised approaches involves the use of clustering or associations where the data is unlabeled. Here we divide the works according to the approaches used:

1) Supervised approaches: The author in [21] proposed a new machine learning model after applying textual similarity to fine tune the parameters of BM25F which not only consider the natural text but also product, component, priority and other aspects of the bug report as well. Similar to this work, [37] also used the SVM approach to learn the REP parameters. In order to address the issues of imbalancing of data, for the training purpose they reduced the number of non-duplicates. The author in [2] used the SVM along with the RBF kernel for the classification of comments of the bug reports. They classified the comments describing the root cause or solution or the bug phenomena as the useful class and rest are classified as less useful. For the classification purpose, they used the idf, status of bug, length of text and unigram features of the comments to create the vector for the learning purpose. The author in [15] in the same year conducted the research where they used the stack trace along with its structure to train the machine learning model. They used the Eclipse project, found that roughly 10 percent of the total corpora contained stack traces but they observed that it was very easy to detect if the bug report is duplicate or not for those 10 percent bug reports in comparison to others where the text needed to be analyzed for the same. They identified the duplicates even before the complete bug report writing was over. The author in [28] used the machine learning classifiers after applying the topic models to pair the bug reports according to the duplicate metrics and used ten-fold cross validation to test their classifiers. Along with ZeroR, NaiveBayes, Logistic Regression, C4.5, K-NN; they also used BootStrap Aggregating Technique to understand how their approach is working. [38] also uses the supervised learning approaches to find the semantic similarity between two texts after finding the feature sets in the vector form. For the binary classification, supervised learning they used the following methods for K-NN(K Nearest Neighbor, Linear SVM, RBF, SVM, Decision Tree, Random Forest and Naive Bayes.). The author in [39] used LibSVM after finding the features to find whether the bug report is duplicate or not.

The author in [32] first created the feature vector by finding the comparison between various features of the bug report and then undersampled the training data and used the Naive-Bayes, K-NN, Logistic Regression and C4.5 for the classification purpose.

2) Unsupervised approaches: In the supervised approaches, as it involves the training and the prediction of model depends upon the training data. Thus there is a need of large and good quality data. The author in [43] used the clustering based weighing approach to improve the performance of SVM Based approaches.

3) Deep learning approaches: Deep Learning is the part of machine learning which involves neural networks with the hidden layers involving the use of activation functions, max-pooling to train and test the bug reports. They also help extract the non-linear features. Deep Learning techniques in the context of duplicate bug report detection involves first the conversion of text into the numerical terms. Though there a number of ways to do so like One-Hot Encoding, Word Embedding, Vocabulary set creation, etc but word embedding is one of the most famous approach for this purpose.

The author in [1] used Siamese style Neural Network which used CNN and LSTM Based approach to detect whether the Bug Report is duplicate or not and find the top-k similar bug reports. They considered 3 parts of the Bug Report for the purpose namely structure information like Component, Title and Description. They used Vanilla single layered neural network for the structured information.

$$\sigma(W.b)$$

They used Bi-LSTM for Title and CNN for Description so as to convert it to small text. For Description, they used Filters and Max-pooling layer. For finding top-k similar bug reports they used Max margin training loss objective function along with backproapagation and stochastic gradient. For Classification purpose, they used two layer neural network along with the softmax function. They used Adam optimizer and cross entropy as the loss function for finding the accuracy of the model.

The author in [5] used the word embedding and the convolutional neural networks(CNN) to find the similarity of bug reports and in turn to identify the duplicate bug reports. Word Embedding is the part of language modeling where the words and phrases are converted into features and vectors. Mainly Randomly-generated embedding, Glove and word2vec are used for word embedding. Convolution Neural Networks are feed-forward neural network. They used these techniques to capture both the syntactic and semantic features of the bug report to increase the performance of Natural language processing. Along with the CNN, they used the domain-specific features of the bug reports. For the activation function, hyperbolic tangent function is used by researchers.

The author in [34] proposed a tool POSTER which uses the high precision property of word-embedding and the high recall property of LDA to capture the semantic and syntactic information about the words in the text. Word-embedding was used to convert the text of bug report into the vector form and then the deep learning model was applied to learn the distribution of duplicate and non duplicate bug reports.

After studying the research work in this field, we find the strengths and limitations of the individual papers. Table II and Table III gives the tabular representation of strengths and limitations of the papers.

III. Methodology

We use the keywords and the citations of the searched papers to find out the relevant documents. We have used IEEE Xplore [50] Elsevier [51], ACL Anthology [52], Cornell University library, Springer [53], ACM [54] and Google Scholar [55] for finding out the concerned research papers. We have used only top 50 research articles for consideration as we observed after these searches the relevance of papers with the query started declining. Table IV describes the searching done to find the important papers in this field.

Table V gives the number of papers which we selected from various libraries for the inclusion into this paper.

Table VI gives the distribution of papers year-wise. The statistics help understand how the evolution of the research in this field has taken place by the research community.

After finding the Papers selected for the literature purpose, we divide them into categories according to the Approach they have used. Table VII gives the summary of papers in terms of which approach was used, what type of Bug Report Categorization was done and the Corpus used for the purpose.

IV. Evaluation Measures

For the evaluation how effective the approach is in terms of identifying the duplicate bug report,

- True Positive Rate: It is the number of actual duplicate bug reports which are classified as duplicate.
- True Negative Rate: It is the fraction of number of non-duplicate bug reports which are classified as non-duplicate.
- Accuracy: It is the proportion of true results to the total number of observations. It is not a very stable metrics. It works well for balanced-set of data only. In case of bug reports, the ratio of duplicate to non-duplicate bug reports is very skewed as number of duplicate bug reports are very less than the total number of non-duplicate bug reports.
- AUC (Area Under Curve) or ROC (Area Under Receiver Operating Characteristic Curve): It measures the capability of machine classifiers to discriminate between instances of certain class. It is the plotting between the True Positive Rate and False Positive Rate. It describes the trade-off between these two. It is also known as specificity. Closer the curve is to 0, lesser is the trade-off. It helps tell what is the probability that the classifier will choose the positive instances over the negative instances.
- Kappa Statistics: It tells how the model fits the data. It is the statistical measure which calculates the inter-rated agreement. If the judges are completely agreeing to one condition, the kappa value comes to 1 and if complete disagreement is there, the value becomes zero. It is used for agreement between the gold set summaries and the machine learning classifier.
- Precision: It is the ratio of number of bug reports which are duplicate and are classified as duplicates to the total number of bug reports which are classified as duplicates. Precision=True Positives/(True Positives + False Positives)
- Recall: It is the ratio of number of bug reports which are duplicate and are classified as duplicate to the number of duplicate bug reports available in the repository. Recall=True Positives/(True Positives + False Negatives)
- F-Measure: It is the harmonic mean of Precision and Recall. F Score=(2* Precision * Recall)/ (Precision + Recall)
- Fisher Score: It is one of the very popular measure of statistics. It is used to find out the effectiveness of different features for distinguishing between duplicate and non-duplicate bug reports. Here mean and standard deviation for the feature values of a particular class are used.

TABLE II. SUMMARY OF STUDIES: STRENGTHS AND LIMITATIONS

(Table Continues in Table III)

Author	Pros	Cons
[17]	Approach does not require training data. Thus can be used for any project.	Their approach could not achieve good results.
[19]	First available approach where the execution trace is also included along with the textual and categorical features of a bug report.	Considered only Recall as the evaluation Measure. More Evaluation measures would have helped give better insights. Their approach depends upon the external tools to collect the execution trace. The approach suffers from privacy issues also.
[20]	They analyzed how the individual feature performs. Thus gives the important features for further research.	Their approach requires the training data. Constructing the training data with large sample size is not an easy task. Very few datasets are available to be used for training for duplicate bug report detection.
[36]	First Paper for analyzing the GUI Bugs by transforming them into Event, Component and Requirement.	Approach does not consider the semantic information of the Bug Report.
[18]	Utilized the word-level information, handled super word features, expansion of short forms, and hyphenated phrases	N-grams introduces the unessential noisy features.
[22]	They used 54 features of bug report title and description along with the SVM to detect the duplicates.	Requires the computation of lot of features.
[21]	Use of BM25 approach along with the stochastic Gradient Descent helped introducing control weights to the features.	Lot of hand-crafted features need to be considered.
[37]	Introduced the concept of relative similarity among the Bug Reports. Approach also addressed the imbalancing problem for Machine Learning .	Only SVM is considered for learning from features.
[44]	They used the longest common subsequence approach which helps preserve the word order. Word ordering is usually neglected by IR approaches.	As the approach is based upon the word ordering, the presence of synonyms and alternate spellings create the problem
[23]		They did not considered the clustering information to their approach.
[2]	First paper where along with just the information from Bug Report, comments and user profile are also considered.	Evaluated only on Naive Bayes, Decision Tree and SVM. More Classifiers can also be considered. More Similarity Measures can be used to improve the feature scores.
[27]	Approach is generic and thus can be used for any software repository.	Their approach does not display the results in ordered manner.
[15]	They used only Stack Traces to find the duplicate Bug Reports. Thus even if the Bug Report is not written properly, it will not impact the performance of the approach.	Very few Bug Reports contain the stack Trace. Thus limiting the approach.
[45]	Achieved the good results with very simple approaches LSI and VSM	Their approaches are experimented on small dataset. Thus results can not be taken as for benchmark.
[46]	They added the contextual information to create the topics which improved their results a lot.	More features can be included for improving the results.
[14]	The approach uses clustering, thus requires no training data	Synonyms and phrases with similar meaning needs to be considered to improve the accuracy of the approach.
[43]	They used the cluster information to their TF-IDF weighing mechanism.	
[29]	The approach does not require training data and can be easily generalized to any project. Rather than just proving their approach, they integrated their approach to the the existing bug tracking system and observed the results.	Their approach only used conventional Vector Space Model (VSM) when their exists many other approaches which give better results than the VSM.
[28]	Along with the contextual information and the existing approaches till their time, their approach also used another metric known as Bootstrap Aggregation which gave better results than previous ones.	The approach does not consider for Top-k duplicate bug reports.
[47]	The approach used the contextual information of software engineering for the duplicate bug report detection. The approach reduced the time and effort to detect the duplicate bug report	The contextual information used was software-level specific not the project-specific.
[30]	They involved the use of both the Classification and the Clustering to detect the duplicates. Classification helps get the topic definition while Clustering helps evaluate the degree of correlation between the topics.	Pre processing requires the intervention of experts. Approach first use classification then clustering, Classification requires the goo
[33]	Their paper not only included the techniques to find the Bug Reports which resembles the textual similarity but also those which do not exhibit textual similarity.	The approach needs the properly labeled triaged dataset as it involves the machine learning approach.
[31]	Discourse Based analysis to explore the Observed Behavior, Expected Behavior and Steps to Reproduce helped improve the NLP Based Techniques	Noisy data and typographical mistakes imposes the challenge
[48]	Reduced the machine learning bias by inclusion of Fidelity Loss Function.	Only includes statistical and textual Similarity Features. Semantic Information is not included. Considered the VSM approach. Approach can be evaluated over other models also.

- Mean Reciprocal Rank: This approach is used by [35] to find the rank of correct answer. As mentioned in their paper, as this approach supports only one correct result, this evaluation measure is not very appropriate for duplicate bug report context. The reason is that for a bug report, many duplicate bug reports can exist. It has no lower limit. This metric gives the mean of reciprocal ranks of the correct results. If the rank of correct result is low, then a problem appears.
- Mean Average Precision: It is based on the ranks of correct results. Here the multiple matches are allowed. Average Precision is calculated for each query and then mean is taken for all the queries.

Table VIII gives the tabular representations of various metrics being used by researchers for evaluating the effectiveness of their approach.

V. FUTURE DIRECTIONS

- Need of testing the approaches with large and different dataset: Almost in all the papers of duplicate bug report detection, few includes [38] [25] [34] [37], mentioned the need of testing their approach with the large dataset to find the reliability of the approaches. They also mentioned the need to extensively train the approaches on various types of software projects.
- Extensive Training and Tuning for Deep Learning models: In a paper by [1],they mentioned the need of performing the extensive training for Deep Learning models by involving the use of various approaches like batch normalization. As in DL Based Models, hyperparameters need to be trained for the learning process. Thus need to fine tune the parameters is also required. For ML and DL Based Models, the

TABLE III. SUMMARY OF STUDIES: STRENGTHS AND LIMITATIONS: CONTINUATION OF TABLE 6 SUMMARY

Author	Strengths	Limitations
[1]	Use of LSTM and CNN Models helped achieve highly accurate duplicate bug reports. No handcrafted features were used by them.	The approach needs to be tested with large training set and the attention mechanism should also be added for improving the results.
[34]	Inclusion of word embedding with deep learning improved the results significantly very high.	The approach only uses simple Deep Learning Model along with the Word Embedding while the RNN and CNN gives better results in terms of syntax and semantic aspects. The model includes two step training.
[5]	Simple Convolutional Model is used for learning the long descriptions of the Bug Report.	Domain-specific features are not used for the approach which limits its results.
[35]	Instead of giving the top-k results, their approach continuously queries the similar bug reports and help be alerted before submitting the bug report. Thus the approach helps users stop at any time.	As the approach uses the words for retrieving the results, many times because of the use of synonyms and other similar words, the results do not come effectively.
[24]	Usually the approaches which have used LDA, used number of topics as the paramter. But in their paper,they studied the parameter tuning techniques which included Genetic Algorithms, Differential Evolution, Particle Swarm Optimization, Simulated Annealing and Random Search. Thus it opens the area of improving LDA to achieve better results.	There are many more meta heuristics which are available which can be explored further to see how the LDA models works with them.
[42]	The approach not only helps in detecting the similar bug reports but also helps assign the new bug report to the appropriate group of previous Bug Reports.	Their approach does not work for those bug reports which do not have prior duplicates in the repository. Their approach involves the analysis of Stack Trace Information but only few bug reports contain Stack Trace, thus impacts the validation of efficiency of the approach.
[49]	Query reformulation Strategy used by them is independent of other bug reports other than itself.	Query Reformulation does not consider the description of Bug Report. It only considers the Title and Observed Behavior for finding the duplicates.
[41]	They included all the temporal, categorical, textual and contextual information of the bug report to detect the duplicates. It is the first paper where they used the Manhattan Distance Similarity Measure and found that the accuracy has improved because of this measure.	Use of domain-specific knowledge can improve the accuracy of the model. More work on reducing the search space for feature calculation is also needed.

TABLE IV. KEYWORDS USED FOR SELECTING THE RESEARCH PAPERS

Application	Strings Used
Duplicate Bug Report: NLP	Duplicate bug report detection using Natural Language Processing, Duplicate Bug Report, Duplicate Bug Report NLP, Bug Report
Duplicate Bug Report: Information Retrieval	Duplicate Bug Report Detection: Information Retrieval, Duplicate Bug, Bug Report
Duplicate Bug Report: Machine Learning	Duplicate Bug Reports Machine Learning, Duplicate Bug, Bug Report, Bug Report Duplicate
Duplicate Bug Report: Deep Learning	Deep Learning for Duplicate Bug Reports, Duplicate Bug Report

TABLE V. PAPER DISTRIBUTION: SOURCE WISE

Link	No of Papers	Duplicate Detection: NLP	Duplicate Detection: IR	Duplicate Detection: ML	Duplicate Detection: DL	Total Used in Paper
IEEE	28	11	9	3	2	28
Springer	7	1	3	2	0	7
ACM	6	2	1	2	1	6
ACL Anthology	2	1	0	1	0	1
Elsevier	2	0	0	2	0	2
Total Papers found	45	15	13	10	3	45

TABLE VI. PAPER DISTRIBUTION: YEAR WISE

Year	No Of Papers
2008	3
2009	2
2010	2
2011	2
2012	6
2013	6
2014	6
2015	3
2016	5
2017	3
2018	3
2019	4
Total	45

approaches need to be trained on large datasets for finding their efficiencies.

- The author in [3], mentioned the following things to take care of for avoiding bug reports duplication:
  - Analyze the submitter profile while analyzing the bug report.
  - Use of controlled vocabulary for the bug report writing.
  - Initial display of Related Search Results
  - Automatic Duplicate Bug Report analysis for the display of only top few duplicate bug reports
  - Visualization of bug reports
- The author in [26] mentioned the need to reformulate and expand the queries to get the better results for duplicate bug report detection.
- Even though lot of works have been done in the field of machine learning, properly addressing the issue of imbalanced data is also very important. The author in [37] modified the training set by reducing the number of non-duplicate instances in the training dataset.
- Need to integrate the approaches to the bug reporting management system is also the need of time to reduce the efforts of Triager.
- Most of the works have used Title and Description for identifying the duplicate bug reports, in a paper by [2], they used the comments also to identify the potential duplicate bug reports. There is a need to include all the

TABLE VII. SUMMARY OF STUDIES

Author	Major Approach Used	Type of Duplicate Bug Report Categorization	Corpus	Main Techniques Use
[17]	Natural Language Based	Top-k	Sony Mobile Erricson Mobile Communications	Similarity Measures, TF
[19]	NLP Based	Top-k	Eclipse, Firefox	Similarity Measures, TF-IDF
[20]	NLP Based	Top-k	Mozilla [56]	Cosine Similarity, TF
[36]	NLP	Top-k		Event Extraction, Longest Common Subsequence(LCS)
[18]	NLP Based	Top-k	Eclipse	Unigram and Bigrams
[22]	Machine Learning Based	Prevention	Openoffice, Firefox, Eclipse	SVM
[21]	Information Retrieval	Top-k	Eclipse, Mozilla, Openoffice	BM25F, Gradient Descent
[37]	Machine-Learning Based	Top-k	Mozilla	SVM
[44]	NLP	Top-k	Eclipse, Firefox	Longest Common Subsequence
[23]	NLP	Top-k	Apache, AgroUML, SVN	BM25
[2]	ML(SL)	Top-k	MeeGo	SVM with RBF Kernel, LDA, Jensen-Shanon, symmetric KL Divergence
[27]	NLP	Top-k	BlackBerry	BM25F+ Smoothing techniques
[15]	NLP	Top-k	Eclipse	TF-IDF
[45]	IR	Top-k	Google Chrome Browser	VSM, LSI
[46]	IR	Prevention	Android [57]	BM25F, LDA
[14]	ML	Top-k	Eclipse, Mozilla, Open Office	K-Means
[43]	ML(USL)	Top-k	Apache, AgroUML,SVN	
[29]	IR	Top-k	-	Vector Space Model
[28]	IR+ ML	Prevention	Android	ZeroR, Naive Bayes, Logistic Regression, C4.5, K-NN, Bagging: REP Tree
[47]	NLP + ML	Top-k	Eclipse, OpenOffice, Mozilla	BM25F
[30]	IR+ML	Top-k	Apache, Eclipse, Mozilla	Naive Bayes, LDA
[33]	IR + ML	Top-k	Chrome Dataset	Vector Space Model, Cosine Similarity, TF-IDF, K-Mean Clustering, Nearest Neighbor Classifier
[48]	IR	Top-k	Eclipse	LDA, Similarity Calculation(Cosine)
[31]	NLP	Top-k	9 different software projects from Github and Jira	SVM+Discourse-Based+Dependency Parsing
[1]	ML(DL)	Top-k, Classification	Open Office, Eclipse, NetBeans	CNN+LSTM +Word Embedding+Feed Forward Neural Network
[34]	DL	Top-k	Firefox, Openoffice	Word Embedding, Deep Learning Feed Forward Neural Network
[5]	DL	Top-k	Hadoop, HDFS, MapReduce, Spark	Word Embeddings, Convolutional Neural Networks
[35]	IR+NLP	Prevention	[58]	LDA, LSA, TF-IDF, BM25F
[24]	IR	TOp-k	Bench4BL	LDA with GA, DE, PSO,SA,Random Search
[42]	NLP	Top-k	Firefox and Gnome	Hidden Markov Model
[49]	NLP	Top-k	Accumulo,Ambari, ActiveMQ, Cassandra, Cordova, Continuum, Drill, Eclipse, Groovy, Hadoop, Hbase, Hive, Maven, Firefox, My Faces, OpenOffice, PDFBox, Spark, Wicket, Struts	Ontology
[41]	NLP	Top-k	Android,Mozilla Eclipse, Open Office	LCS , N-gram

aspects of a bug report including comments to solve this problem.

- The author in [4] observed that finding those reports which are textually similar requires less effort by developers but the reports which does not exhibit similar textual similarity are more difficult to be identified by developer. Thus more work is required to identify the approach where the automatic techniques identify the issue reports which do not or exhibit less textual similarity.

VI. CONCLUSION

Duplicate Bug Report Detection is one of the very important and frequent task that is performed on the daily-basis by a person called triager. It is a tedious and a time-consuming activity apart from the main software development and other tasks. In this paper, we have identified the main works that have been performed for this problem. We classified the works into the Natural Language Processing Based, Information-Retrieval Based, Machine Learning Based and Deep Learning Based. We systematically mentioned all the works in the classified manner. We also present the statistics about this work which gives an insight about how much work has been done in this field. We have mentioned almost all the popular evaluation measures which have been used by various researchers. We have also mentioned the Strengths and Limitations of all the major works in this field along with their experimental results. This will enable the researchers to easily identify the gaps and compare their results. Apart from mentioning just the works done, we also present the challenges and pointers for the future research in this field.

TABLE VIII. RESULTS FROM RESEARCHES

Author	Extension of Research	Evaluation Measure1	Evaluation Measure2	Any Other
[20]: Mozilla		TPR: 8 %	TNR: 100 %	Harmonic: 15%
[37]: Mozilla		TPR: 24%	TNR: 91%	Harmonic: 39%
[14]: Mozilla		TP Rate: 27%	TN Rate: 86%	Harmonic: 41%
[21]		MAP(BM25Fext): OpenOffice-45.21 %, Mozilla-46.22%, Eclipse-53.21%, Large Eclipse-44.22%	Recall: Top-1: 37%, Top-20: 71%	
[44]		Recall: Firefox- 73% (Top-20), Eclipse - 85%		
[5]: Spark Dataset		F Measure: Random: 0.964, GloVe: 0.944, word2Vec:925	Accuracy: Random: 0.951, GloVe: 0.935, word2Vec: 0.920	
[18]		Recall Rate: 0.2 (Top-1), 0.35 (Top-5), 0.49 (Top-20)		
[23]		Recall Rate:0.112(Top-1), 0.2 (Top-5), 0.3 (Top-20)		
[34]		Recall Rate: 0.25 (Top-1), 0.5 (Top-5), 0.7 (Top-20)		
[1]		Recall: 80% (Top-20)	Accuracy: 90 % (Top-20)	
[30]: Classification +LDA		Accuracy: Naive Bayes: 38.74(T-1), 46.39 (T-2), 60.52 (T-3) Bayesian Network: 27.93(T-1), 36.98 (T-2), 47.43(T-3) C4.5: 41.31(T-1), 48.5 (T-2), 60.52 (T-3) SVM: 22.96(T-1), 41.33 (T-2), 43.72 (T-3)		
[22] Textual and Categorical		Accuracy: 80%(ZeroR), 79.655% (Naive Bayes), 88.125% (Logistic Regression), 92.105% (C4.5), 91.55 %(K-NN)	AUC: 0.500(ZeroR), 0.904(Naive Bayes), 0.788(Logistic Regression), 0.888(C4.5), 0.7561%(K-NN)	Kappa: 0.0(ZeroR), 0.3508(Naive Bayes), 0.5967(Logistic Regression), 0.7553(C4.5), 0.0.7561(K-NN)
[46]: Textual, Categorical and Labeled-LDA	[22]	Accuracy: 80%(ZeroR), 79.655% (Naive Bayes), 88.125% (Logistic Regression), 92.105% (C4.5), 91.55 %(K-NN)	AUC: 0.500(ZeroR), 0.904(Naive Bayes), 0.788(Logistic Regression), 0.888(C4.5), 0.7561%(K-NN)	Kappa: 0.0(ZeroR), 0.3508(Naive Bayes), 0.5967(Logistic Regression), 0.7553(C4.5), 0.0.7561(K-NN)
[47]: Labelled LDA, Android Textbook	[46]	Accuracy: 80%(ZeroR), 87.01% (Naive Bayes), 94.22% (Logistic Regression), 92.75% (SVM), 94.22 %(C4.5)	Kappa: 0.0(ZeroR), 0.583(Naive Bayes), 0.753(Logistic Regression), 0.750(SVM), 0.799(C4.5)	
[28]	[46]	Accuracy: 80%(ZeroR), 93% (Naive Bayes), 94.5% (Logistic Regression), 94.7% (C4.5), 94.75 %(K-NN), 95.17 (Bagging)	AUC: 0.5(ZeroR), 0.958(Naive Bayes), 0.972(Logistic Regression), 0.941(C4.5), 0.955 %(K-NN), 0.977(Bagging)	Kappa: 0.0(ZeroR), 0.77(Naive Bayes), 0.824(Logistic Regression), 0.832(C4.5), 0.830(K-NN), 0.845(Bagging)
[49]		Recall: Observed Behavior: 56.6 %, Bug Title: 59.6%, Title +Observed: 78%		
[41]		Precision: Mozilla-97.14, Android-99.12, Eclipse-96.86, Open Office-98.45	Recall: Mozilla-97.78, Android-99.45, Eclipse-90.12, Open Office-97.59	Accuracy: Mozilla-97.14, Android-99.47, Eclipse-96.58, Open Office-97.10 %

REFERENCES

[1] J. Deshmukh, A. M, S. Podder, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," 09 2017, pp. 115–124.

[2] L. Feng, L. Song, C. Sha, and X. Gong, "Practical duplicate bug reports detection in a large web-based development community," in *Web Technologies and Applications*, Y. Ishikawa, J. Li, W. Wang, R. Zhang, and W. Zhang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 709–720.

[3] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira, "The bug report duplication problem: an exploratory study," *Software Quality Journal*, vol. 21, no. 1, pp. 39–66, Mar 2013. [Online]. Available: https://doi.org/10.1007/s11219-011-9164-5

[4] W. S. Mohamed Sami Rakha and A. E. Hassan, "Studying the needed effort for identifying duplicate issues." *Empir Software Eng*, vol. 21, pp. 1960 – 1989, 2016.

[5] Q. Xie, Z. Wen, J. Zhu, C. Gao, and Z. Zheng, "Detecting duplicate bug reports with convolutional neural networks," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, Dec 2018, pp. 416–425.

[6] A. Lamkanfi, J. Perez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information," in *MSR '13: Proceedings of the 10th Working Conference on Mining Software Repositories, May 18–-19, 2013. San Francisco, California, USA*, 2013.

[7] [Online]. Available: https://bugzilla.mozilla.org

[8] [Online]. Available: https://bugs.eclipse.org/bugs/describecomponents.cgi?\\product=Platform

[9] [Online]. Available: https://bugs.eclipse.org/bugs/describecomponents.cgi?\\product=JDT

[10] [Online]. Available: http://issues.apache.org/jira/browse/CASSANDRA

[11] [Online]. Available: http://issues.apache.org/jira/browse/MAPREDUCE

[12] [Online]. Available: http://issues.apache.org/jira/browse/HDFS

[13] [Online]. Available: http://issues.apache.org/jira/browse/SPARK

[14] R. P. Gopalan and A. Krishna, "Duplicate bug report detection using clustering," in *Proceedings of the 2014 23rd Australian Software Engineering Conference*, ser. ASWEC '14. Washington, DC, USA:

IEEE Computer Society, 2014, pp. 104–109. [Online]. Available: https://doi.org/10.1109/ASWEC.2014.31

[15] J. Lerch and M. Mezini, "Finding duplicates of your yet unwritten bug report," 03 2013, pp. 69–78.

[16] J. He, N. Nazar, J. Zhang, T. Zhang, and Z. Ren, "Prst: A pagerank-based summarization technique for summarizing bug reports with duplicates," *International Journal of Software Engineering and Knowledge Engineering*, vol. 27, pp. 869–896, 06 2017.

[17] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *29th International Conference on Software Engineering (ICSE'07)*, May 2007, pp. 499–510.

[18] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *2010 Asia Pacific Software Engineering Conference*, Nov 2010, pp. 366–374.

[19] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *2008 ACM/IEEE 30th International Conference on Software Engineering*, May 2008, pp. 461–470.

[20] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, June 2008, pp. 52–61.

[21] C. Sun, D. Lo, S. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, 2011, pp. 253–262.

[22] C. Sun, D. Lo, X. Wang, J. Jiang, and S. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, May 2010, pp. 45–54.

[23] C. Yang, H. Du, S. Wu, and I. Chen, "Duplication detection for software bug reports based on bm25 term weighting," in *2012 Conference on Technologies and Applications of Artificial Intelligence*, 2012, pp. 33–38.

[24] A. Panichella, "A systematic comparison of search algorithms for topic modelling—a study on duplicate bug report identification," in *Search-Based Software Engineering*, S. Nejati and G. Gay, Eds. Cham: Springer International Publishing, 2019, pp. 11–26.

[25] Y. Song, X. Wang, T. X. L. Zhang, and H. Mei, "Jdf: Detecting duplicate bug reports in jazz," 2010.

[26] N. Kaushik and L. Tahvildari, "A comparative study of the performance of ir models on duplicate bug detection," in *2012 16th European Conference on Software Maintenance and Reengineering*, March 2012, pp. 159–168.

[27] M. Amoui, N. Kaushik, A. Al-Dabbagh, L. Tahvildari, S. Li, and W. Liu, "Search-based duplicate defect detection: An industrial experience," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 173–182.

[28] N. Klein, C. S. Corley, and N. A. Kraft, "New features for duplicate bug detection," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 324–327. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597090

[29] F. Thung, P. S. Kochhar, and D. Lo, "Dupfinder: Integrated tool support for duplicate bug report detection," 2014.

[30] C. Jingliang, M. Zhe, and S. Jun, "A data-driven approach based on lda for identifying duplicate bug report," in *2016 IEEE 8th International Conference on Intelligent Systems (IS)*, 2016, pp. 686–691.

[31] O. Chaparro, "Improving bug reporting, duplicate detection, and localization," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 421–424.

[32] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empirical Software Engineering*, vol. 21, no. 2, pp. 368–410, Apr 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9387-3

[33] P. Anjaneyulu, G. Sarbendu, A. Gopichand, P. B. Satya, and P. Srinivas", *An Analytics-Driven Approach to Identify Duplicate Bug Records in Large Data Repositories*. Cham: Springer International Publishing, 2016, pp. 161–187. [Online]. Available: https://doi.org/10.1007/978-3-319-31861-5_8

[34] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "Poster: Dwen: Deep word embedding network for duplicate bug report detection in software repositories," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, May 2018, pp. 193–194.

[35] A. Hindle and C. Onuczko, "Preventing duplicate bug reports by continuously querying bug reports," *Empirical Software Engineering*, vol. 24, no. 2, pp. 902–936, Apr 2019. [Online]. Available: https://doi.org/10.1007/s10664-018-9643-4

[36] N. K. Nagwani and P. Singh, "Bug mining model based on event-component similarity to discover similar and duplicate gui bugs," in *2009 IEEE International Advance Computing Conference*, 2009, pp. 1388–1392.

[37] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *2012 16th European Conference on Software Maintenance and Reengineering*, March 2012, pp. 385–390.

[38] A. Lazar, S. Ritchey, and B. Sharif, "Improving the accuracy of duplicate bug report detection using textual similarity measures," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 308–311. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597088

[39] A. Tsuruda, Y. Manabe, and M. Aritsugi, "Can we detect bug report duplication with unfinished bug reports?" in *2015 Asia-Pacific Software Engineering Conference (APSEC)*, Dec 2015, pp. 151–158.

[40] K. K. Sabor, A. Hamou-Lhadj, and A. Larsson, "Durfex: A feature extraction technique for efficient detection of duplicate bug reports," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2017, pp. 240–250.

[41] B. S. Neysiani and S. Morteza Babamir, "Improving performance of automatic duplicate bug reports detection using longest common sequence : Introducing new textual features for textual similarity detection," in *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, 2019, pp. 378–383.

[42] N. Ebrahimi, A. Trabelsi, M. S. Islam, A. Hamou-Lhadj, and K. Khanmohammadi, "An hmm-based approach for automatic detection and classification of duplicate bug reports," *Information and Software Technology*, vol. 113, pp. 98 – 109, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095058491930117X

[43] M. Lin and C. Yang, "An improved discriminative model for duplication detection on bug reports with cluster weighting," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 117–122.

[44] S. Banerjee, B. Cukic, and D. Adjeroh, "Automated duplicate bug report classification using subsequence matching," in *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*, 2012, pp. 74–81.

[45] I. Chawla and S. K. Singh, "Performance evaluation of vsm and lsi models to determine bug reports similarity," in *2013 Sixth International Conference on Contemporary Computing (IC3)*, 2013, pp. 375–380.

[46] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," 05 2013, pp. 183–192.

[47] K. Aggarwal, T. Rutgers, F. Timbers, A. Hindle, R. Greiner, and E. Stroulia, "Detecting duplicate bug reports with software engineering domain knowledge," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 211–220.

[48] J. Zou, L. Xu, M. Yang, M. Yan, D. Yang, and X. Zhang, "Duplication detection for software bug reports based on topic model," in *2016 9th International Conference on Service Science (ICSS)*, 2016, pp. 60–65.

[49] O. Chaparro, J. M. Florez, U. Singh, and A. Marcus, "Reformulating queries for duplicate bug report detection," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 218–229.

[50] [Online]. Available: https://ieeexplore.ieee.org/

[51] [Online]. Available: https://www.elsevier.com

[52] [Online]. Available: http://aclweb.org/anthology/

[53] [Online]. Available: https://link.springer.com

[54] [Online]. Available: https://dl.acm.org/

[55] [Online]. Available: https://scholar.google.co.in/

[56] [Online]. Available: https://bugzilla.mozilla.org/buglist.cgi?quicksearch=Mozilla

[57] [Online]. Available: https://source.android.com/setup/contribute/report-bugs

[58] [Online]. Available: https://archive.org/details/2016-04-09ContinuousQuery