A Novel Two-Step Classification Approach for Runtime Performance Improvement of Duplicate Bug Report Detection*

Research Article

Behzad Soleimani Neysiani[1] Seyed Morteza Babamir [2]

Abstract: Duplicate Bug Report Detection (DBRD) is one of the famous problems in software triage systems like Bugzilla. There are two main approaches to this problem, including information retrieval and machine learning. The second one is more effective for validation performance. Duplicate detection needs feature extraction, which is a time-consuming process. Both approaches suffer runtime issues, because they should check the new bug report to all bug reports in the repository, and it takes a long time for feature extraction and duplicate detection. This study proposes a new two-step classification approach which tries to reduce the search space of the bug repository search space in the first step and then check the duplicate detection using textual features. The Mozilla and Eclipse datasets are used for experimental evaluation. The results show that overall, 87.70% and 89.01% validation performance achieved averagely for accuracy and F1-measure, respectively. Moreover, 95.85% and 87.65% of bug reports can be classified in step one very fast for Eclipse and Mozilla datasets, respectively, and the other one needs textual feature extraction until it can be checked by the traditional DBRD approach. An average of 90% runtime improvement is achieved using the proposed method.

Keywords: Duplicate Detection, Bug Report, Machine Learning, Runtime Performance, Search Space Reduction

1. Introduction

Duplicate detection is one of the essential and time-consuming operations in social communities like software repositories of bug reports (e.g., Bugzilla) or question and answering forums (e.g., Stack Overflow). There has been about 30% to 60% duplicate bug reports in various software repositories, especially open-source projects, and it is growing every day with growing their communities [1]. Duplicate detection needs to compare a new bug report to all bug reports of the repository. The comparing process is not straightforward because bug reports contain many data fields with various domains (e.g., identity, temporal, categorical, and textual domains). The textual data fields cannot be compared simply because two texts may have the same content but different forms and words. So, feature extraction should be used to convert bug reports as unstructured data to structured data [2]. There are many efforts on feature extraction, like using time difference of temporal data fields [3], textual features considering term frequencies [4], and subsequence matching [5, 6], using similarity of bug reports to specific topics as contextual features [7, 8, 9]. By the way, there are some issues for feature extraction, especially for textual data fields, e.g., stemming, removing the stop words, correcting typos [10, 11, 12, 13, 14], which can improve the validation performance of duplicate bug report detection (DBRD).

After feature extraction, the features of a pair of bug reports, including a new bug report and another one from the repository, should be checked for duplication. The Information Retrieval (IR) approach checks the similarity of these features to the features of other pairs of bug reports. If the two feature vectors were very similar, they would be reported as duplicates. Machine Learning (ML) approach tries to learn the features of duplicate pairs and predict the label of a new pair without comparing it to other pairs, usually [15]. ML approach is a little faster than IR approach because, after feature extraction, it uses the ML algorithm to predict the duplication, but IR approach compares the feature to other features that take a long time again [16].

Duplicate detection is a binary operator that needs two bug reports, and we cannot say a bug report is duplicated without considering other bug reports. It is challenging because of the massive number of bug reports in the repository. If we suppose every feature extraction and duplicate detection using ML algorithms take just 1 second –even though it can take more time based on the feature extraction methods, especially for textual features-, for a bug report repository containing 10,000 bug reports, it takes 10,000 seconds, which is about 2.7 hours. So, this approach cannot be used for online DBRD. Besides, some feature extractors like the longest common subsequence sometimes take more than 1 second to calculate.

Offline DBRD has no time limit. It has a repository of bug reports and tries to find duplicate bug reports like a clustering problem that categorizes data in some clusters. Here, the clusters contain those bug reports that are related and duplicated. Online DBRD tries to find a duplicate of new bug reports as it wants to be submitted in the repository and even helps the writer avoid submitting duplicated bug report real-time. The continuous query is a kind of online DBRD that repeatedly checks duplications [17, 18], and the time complexity is very important in the online DBRD versus the offline version. Such complexity is the major problem of online DBRD, which is currently lacking knowledge.

This study focuses on the runtime performance as a significant online DBRD objective to avoid comparing a new bug report to all bug reports of the repository in online DBRD. The significant difference between this study and related works is that this stufy considers runtime challenges for online DBRD, not just the validation performance. The main contributions of this study are:

1. Introducing a novel two-step classifying approach for improving the runtime performance of DBRD based on two light and full classifiers. The first one uses faster and

* Manuscript received: 2020 May 2, Revised, 2020 November 2, Accepted, 2022 May 17.

[1] PhD Candidate, Department of Software Engineering, University of Kashan, Kashan, Iran.

[2] Corresponding author. Professor, Department of Software Engineering, University of Kashan, Kashan, Iran.
Email: babamir@kashanu.ac.ir

easier features, and the second one uses all the time-consuming features;

2. Using voting ensemble approach to improve the validation performance of the proposed online and two-step DBRD.

This study's fundamental hypothesis is that a two-step filtering-based classification approach reduces the feature extraction runtime for online DBRD.

Section 2 will review the machine learning approach, and Section 3 introduces our proposed machine learning algorithm. Section 4 includes the results of the experiments, and Section 5 concludes the study.

2. Literature review

The following first sub-section will introduce methodology of Duplicate Bug Report Detection (DBRD) and then the feature extraction methods will be illustrated to clear demonstration of examples about proposed method. Moreover, a comparative tabular review on the related works will be summarized to show lack of runtime improvement in state-of-the-arts.

2.1. The methodology of Duplicate Bug Report Detection (DBRD)

Figure 1 shows the traditional approach of duplicate bug report detection (DBRD). The bug reports of a dataset will be pre-processed in the first step (box 2). There are many pre-processing operations such as dealing with null values, homogenizing data types, cleaning textual fields, and preparing for feature extraction. Then pairs of bug reports should be selected for duplicate checking (box 4).

This methodology is for offline DBRD, but it can also be used for online DBRD. In online DBRD, there is no need to select pairs of bug reports. The new bug report can be paired with all bug reports of the repository instead. Then, feature extraction will be used to extract various types of features such as categorical, temporal, textual, and contextual features (box 6) [2]. The feature selection [19] or instance-based learning [20] can be used at this time after feature extraction and before the train and test process. The feature vector sets will be divided into two parts for offline usage, including some vectors for training a machine learning (ML) algorithm and others for testing the ML (outputs of box 7). Now it is time to use an ML for training the features of duplicated pairs (box 8). The trained ML will then be used to predict the test set label (box 10). Four modes occurr here based on the prediction label and the real one, which is tabulated in Table 1. The validation performance metrics of the evaluation process can be calculated based on Table 1. There are many reliable and robust metrics for this purpose.

Accuracy refers to true predictions for duplication or non-duplication status of all bug reports (1).

$$Accuracy = \frac{True\ Prediction\ (TP)}{Total\ (TT)} \qquad (1)$$

Precision indicates the exactitude of duplication detection of duplicates as (2).

$$Precision = \frac{True\ Duplicates\ (TD)}{Predicted\ Duplicates} \qquad (2)$$

Recall shows the memory of an ML algorithm for actual duplicates as (3).

$$Recall = \frac{TD}{Actual\ Duplicates = TD + FND} \qquad (3)$$

F1-measure or F1-score is a harmonic mean of precision and recall as (4).

$$F1 - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \qquad (4)$$

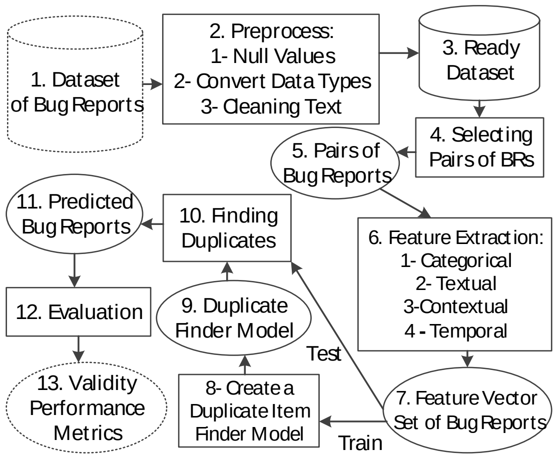

1. Dataset of Bug Reports
2. Preprocess: 1- Null Values 2- Convert Data Types 3- Cleaning Text
3. Ready Dataset
4. Selecting Pairs of BRs
5. Pairs of Bug Reports
6. Feature Extraction: 1- Categorical 2- Textual 3-Contextual 4 - Temporal
7. Feature Vector Set of Bug Reports
Train
Test
8- Create a Duplicate Item Finder Model
9. Duplicate Finder Model
10. Finding Duplicates
11. Predicted Bug Reports
12. Evaluation
13. Validity Performance Metrics

Figure 1. The methodology of Duplicate Bug Report Detection using Machine Learning Algorithms [15]

Table 1. Modes of the duplicate detection

Actual → /Predict ↓	**Actual Dup (AD)**	**Actual Non-Dup (AND)**	**Total Actual Status**
Predicted Duplicated	True Dup (TD)	False Dup (FD)	AD = TD+FND
Predicted Non-Duplicated	False Non-Dup (FND)	True Non-Dup (TND)	AND = FD+TND
Total Prediction	True Prediction (TP=TD+TND)	False Prediction (FP=FD+FND)	Total (TT = TP+FP=AD+AND)

2.2. Feature extraction methods

The most crucial bug reports' fileds are textual which can not be used by machine learning techniques and need to be converted to nominal or numerical data which is called feature extraction. There are many feature extraction types in state-of-the-art, which can be categorized as:

1. Textual features extract the similarity of textual fields of bug reports using natural language processing and information retrieval techniques. The tokenizing sentences and extracting words, removing useless and frequent words known as stop words, removing conjunctions and punctuation, removing redundant words, and stemming words to find the pure form of each noun or verb. The process of counting the same words in two bug reports requires pre-processing.

The *N-gram* model compares the *n-sequence-word* of two text fields. Increasing *n* in *n-gram* indicates greater similarity between two documents. TF and IDF refer to the frequency of a term in a document and in a set of documents, respectively. Equation 1 and 2 are commonly used in a DBRD context, and the BM25F model is built based on these equations [4]. An occurrence of a term *t* in document *d*, which can be a textual field of a bug report in a bug triage system is checked through Equation 1. Parameter *K* is the number of textual fields in document *d*, and *f* is an index of the textual fields of a bug report. The weight factor w_f is based on the importance of each text field, the length is the number of characters in term t, and average_length*f* is the average length of all words in this field. The importance of a term t of document D is calculated using Equation 2 in all bug reports of the software repository, which contains many documents *d*, and each document contains many terms *t*. The result of BM25F is an aggregated value representing the weighted average of the TF and IDF approaches for all standard terms in both texts *d* and *q*, and K1 is a constant for preventing division by zero in Equation 3.

$$TF_D(t,d) = \sum_{f=1}^{K} \frac{w_f \times occurrences(d[f],t)}{1-b_f+\frac{b_f \times length}{average_{length_f}}} \quad (1)$$

$$IDF(t,D) = log\frac{N}{|\{d \in D : t \in d[f]\}|} \quad (2)$$

$$BM25F_{ext}(d,q) = \sum_{Mt \in d[f] \cap q[f]} IDF(t, Total\ Text\ Fields\ of\ Bug\ Reports) \times \frac{TF_D(t,d[f])}{K_1 + TF_D(t,d[f])} \quad (3)$$

There are two major text fields in bug reports: title and description. It should be noted that at least one of the title and description fields is non-empty [10]. Comparing the different combination of these two fields requires more computational overhead and is time-consuming for feature extraction.

Sometimes, simple features can also be extracted from texts, such as text size (length of text in characters or number of words in the text) [21], which is shown in Equation 4 and where the norm (||) refers to the size of bug reports in words, and *abs* is the absolute value. There are many typos in bug reports [10], which have adverse side impacts on textual features and should be corrected as a pre-processing phase.

$$SizeDiff(d,q) = abs(|d[f]| - |q[f]|) \quad (4)$$

The interconnected typos are usual in software bug reports because of the identity of variables and methods in the stack traces, or sometimes they record user-typing mistakes. Some algorithms proposed a correction of these typos [11, 12], but this phase is more complicated and needs additional effort to find the best candidates among the suggested corrections based on the context of a bug report. A new labeled dataset is introduced for typo corrections in the bug report context in which the correction algorithms have about 80% accuracy, and the effectiveness of typo correction on DBRD is an unresolved issue [13].

It is also possible to use other textual features, such as extracting the length of the longest common subsequence (LCS) in two texts as a textual feature and some other derived features (such as the number of words in LCS [5, 6]), or by using word embedding vectors [22]. The bag of words is another textual feature extraction method that considers different textual fields of bug reports as a bag and compares textual features of each bag with other bags.

The time complexity of textual feature extraction methods is greater than other feature types. The bag of words produces many textual features that are very time-consuming. Additionally, the extracted features may be unnecessary and need dimension reduction to select the best features, which causes a further deceleration of the workflow, and is not used in state-of-the-art features [23].

The technique of word embedding has been used regularly to extract the frequency of each term considering nearby terms in a bug report textual field [22, 24]. This technique suffers from high dimensionality and a sparsity problem, because it considers all terms in the bug report repository as vectors and counts the frequency of each term in a specific bug report for nearby terms to convert the unstructured textual field to a numeric structured vector. This method is a type of word2vec model. It is a very time and memory intensive and is appropriate for training neural network models, especially deep models. The neural network models are especially appropriate for solving non-linear problems, but related works showed that DBRD is a rule-based problem which can be solved by linear models as well [5, 7]. Therefore, it is better to avoid using this technique until it becomes necessary.

2. Temporal feature is a type of feature that shows an interval time between two bug reports (Equation 5 and 6) in the seconds or milliseconds [3, 25]. Usually, when a new release of the software is published, many users report duplicate bugs, and so there is a relationship between the submission dates of bug reports. The lesser value of these features indicates the highest probability of similarity of two bug reports. However, some researchers use a timing window instead of temporal features to limit the search space of the duplicate finder and find the duplication in a specific period [26].

$$f_{Id}(d,q) = abs(d.BugId - q.BugId) \tag{5}$$

$$f_{Date}(d,q) = abs(d.OpenDate - q.OpenDate) \tag{6}$$

3. *Structural features* are calculated based on runtime information [27] and stack traces [28] in bug reports. Only some bug reports have this type of information in their description; therefore, it is not possible to calculate these features for all of the bug reports. Textual similarity techniques can also be used to calculate these features; new methods convert the stack trace to a graph and extract some graph-based features like number of nodes, number of incoming and outgoing edges of nodes, and similar metrics. The hidden Markov model can also be used to investigate the similarity of chain on method calls in stack traces as a feature [29].

4. *Categorical feature* is a type of feature that shows how much two bug reports are related [4] using equality comparisons or subtraction of categorical fields. These features can be calculated by either checking the equality of two nominal values like (7), (8), and (9) or subtracting two ordinal or interval values like (10), (11), (12), and (13) in two bug reports d and q [4, 25]. Both (10) and (11) or (12) and (13) are similar, and both pairs always generate a number less than 1. However, (11) and (13) sometimes may be invalid because of division by zero, for the same priorities or variations, which can be considered as zero. Perhaps Lazar et al. [25] wrote or misused the equations, but these new features can also be studied. The letters "A" and "S" at the end of these equations refers to "Addition" and "Subtraction" in their denominators, respectively.

$$f_{Product}(d,q) = \begin{cases} 1 & if\ d.Product = q.Product \\ 0 & otherwise \end{cases} \tag{7}$$

$$f_{Company}(d,q) = \begin{cases} 1 & if\ d.Company = q.Company \\ 0 & otherwise \end{cases} \tag{8}$$

$$f_{Type}(d,q) = \begin{cases} 1 & if\ d.Type = q.Type \\ 0 & otherwise \end{cases} \tag{9}$$

$$f_{PriorityA}(d,q) = \frac{1}{1+|d.Priority - q.Priority|} \tag{10}$$

$$f_{Pr\ iorityS}(d,q) = \frac{1}{1-|d.Priority - q.Priority|} \tag{11}$$

$$f_{VersionA}(d,q) = \frac{1}{1+|d.Version - q.Version|} \tag{12}$$

$$f_{VersionS}(d,q) = \frac{1}{1-|d.Version - q.Version|} \tag{13}$$

5. Topical or contextual feature is a type of feature that is used to compare textual fields of a bug report with a word list containing exclusive content, like (1) security, (2) performance of software [30], (3) the anonymous topics made by latent Dirichlet analysis (LDA) [31], or (4) latent semantic indexing (LSI). The results obtained from these semi-textual features indicate how much the report involves specific contexts; thus, the conceptual category for bug reports. Contextual features of two bug reports can be compared as a vector by the cosine similarity equation or the Manhattan similarity individually to expand the feature space of bug reports [7].

2.3. Machine learning algorithms

As Table 2 shows, so much effort has been done over the past decade to detect duplicate bug reports based on the above descriptions. The numbered columns refer to textual, identical, temporal, structural, categorical, and contextual features. Some features (textual, temporal, and categorical) are essential, and the duplicate detection process should mention them, while contextual features are less important [7]. Textual features can also cover structural features, even though structural features represent another aspect of similarity between bug reports. Further, all bug reports cannot calculate them, except those that have stack trace(s).

Contextual features need contextual attributes as some derived attributes based on textual fields can be calculated and stored in preprocessing phase in clearning texts section to reduce feature extraction runtime.

Most state-of-the-art approaches use ML algorithm. Almost all related works have focused on improving the validation performance using new feature extraction methods [5, 7, 8, 32, 33] and/or using various ML algorithms like deep learning [24, 34, 35, 36]. Table 2 shows a brief review of related and state-of-the-art works using ML algorithms. As Table 2 hows, none of these related works mentioned the search space and runtime challenges of duplication-checking, except a continuous query study that tried to improve validation performance on the continuous query as an online challenge [17]. The related works usually choose a part of pairs of bug reports randomly to evaluate their methods without considering runtime challenge, although if they want to compare a new bug report with the entire database, it was very time-consuming. As there is no related work for the DBRD runtime problem, the literature review is limited to state-of-the-art studies' general parameters and features.

Reviewing the literature showed that runtime challenge is considered for the first time for DBRD. Therefore, we will choose the best current methods for comparison. Besides, the literature review shows that the selected parameters for experiments are almost the same as state-of-the-art.

Table 2. Review of related works in state-of-the-art using machine learning algorithms

Row	Ref	Year	Machine Learning Algorithms	Dataset	Validation Metrics
1	Bettenburg et al. [38]	2008	SVM, Naïve Bayes,	Eclipse	Accuracy
2	Sun et al. [39]	2010	SVM	Eclipse, Mozilla, OpenOffice	Recall
3	Nguyen et al. [31]	2012	LDA, Ensemble Averaging	Eclipse, Mozilla, OpenOffice	Accuracy
4	Tian et al. [40]	2012	SVM	Mozilla	F1-measure, TP and TN Rates
5	Liu et al. [41]	2013	SVM	Eclipse, Mozilla	F1-measure, MAP
6	Alipour et al. [30], [42]	2013	0-R, C4.5, kNN, Logistic Regression, Naïve Bayes	Android	Accuracy, Kappa, ROC, AUC
7	Feng et al. [43]	2013	SVM, Naïve Bayes, Decision Tree	MeeGo	Accuracy, Precision, Recall, MAP, TP and TN Rates
8	Lazar et al. [25]	2014	kNN, Linear SVM, RBF SVM, Naïve Bayes, Decision Tree, Random Forest	Eclipse, Mozilla, OpenOffice, NetBeans	Accuracy, Precision, Recall, AUC
9	Tsuruda et al. [44]	2015	SVM	Eclipse, OpenOffice	Accuracy, Precision, Recall
10	Aggarwal et al. [8], [9]	2015, 2017	0-R, Naïve Bayes, Logistic Regression, SVM, C4.5	Eclipse, Mozilla, OpenOffice	Accuracy, Kappa
11	Sharma and Sharma [45]	2015	SVM	Bugzilla	ROC, TP and FP Rates,
12	Hindle et al. [46]	2016	0-R, C4.5, kNN, Logistic Regression, Naïve Bayes	Android, Eclipse, Mozilla, OpenOffice	Accuracy, Kappa, ROC, AUC, MAP
13	Lin et al. [23]	2016	SVM	Apache, ArgoUML, SVN	Recall
14	Pasala et al. [47]	2016	kNN	Chrome	Recall
15	Rakha et al. [48]	2016	Random Forest	Eclipse, Mozilla, Bugzilla, SeaMonkey	Precision, Recall, F1-measure, AUC
16	Deshmukh et al. [34]	2017	Siamese Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM)	Eclipse, OpenOffice, NetBeans	Accuracy, Recall
17	Budhiraja et al. [24]	2018	Deep Neural Network	Mozilla, OpenOffice	Recall
18	Su and Joshi [49]	2018	Logistic Regression	Oracle	Recall
19	Xie et al. [36]	2018	Convolutional Neural Networks	Hadoop, HDFS, MapReduce, Spark	Accuracy, F1-measure
20	Soleimani Neysiani and Babamir [5]	2019	Naïve Bayes, Decision Tree, Linear Regression, Perceptron Neural Network, Bayesian Boosting by Decision Tree	Android, Eclipse, Mozilla, OpenOffice	Accuracy, Precision, Recall
21	Soleimani Neysiani and Babamir [7]	2019	Naïve Bayes, Decision Tree, Linear Regression, Auto MLP, Bagging ensemble of Decision Tree	Android, Eclipse, Mozilla, OpenOffice	Accuracy, Precision, Recall
22	Soleimani Neysiani and Babamir [14]	2019	Naïve Bayes, k-Nearest Neighborhood, Decision Tree, Linear Regression, Auto Multi-Layer Perceptron, Deep Learning with H2O	Android	Accuracy, Precision, Recall
23	Soleimani Neysiani and Babamir [16]	2020	k-Nearest Neighborhood, Linear Regression	Android	Accuracy, Precision, Recall, F1 Measure
24	Soleimani Neysiani et al. [50]	2020	Linear Regression, Decision Tree, Auto Multi-Layer Perceptron, Deep Learning with H2O	Android, Eclipse, Mozilla, OpenOffice	Accuracy, Precision, Recall, F1 Measure
25	Soleimani Neysiani et al. [20]	2020	Linear Regression, Decision Tree, k-Nearest Neighborhood	Android, Mozilla,	Accuracy, Precision, Recall
26	Kukkar et al. [51]	2020	Deep Learning (CNN)	Eclipse, Mozilla, OpenOffice, Gnome, NetBeans, Firefox	Accuracy, Precision, Recall, F1 Measure, Recall @k
27	Kim and Yang [52]	2021	Naïve Bayes, Random Forest, CNN, LSTM, CNN+LSTM	Eclipse, Mozilla, Apache, KDE	Accuracy, Precision, Recall, F1 Measure
28	Zhang et al. [53]	2022	Deep Learning (Dual Channel-CNN)	Eclipse, Mozilla, Hadoop, Spark, Kibana, VS Code	Recall @k

3. Proposed method

Calculating the textual features is very time-consuming. The main idea of the proposed approach is dividing the duplicate detection process into two phases: 1) trying to predict the duplication status using light features like non-textual features, which can be calculated quickly; 2) predicting the duplication status using all features, including the textual features that are more time-consuming. Figure 2 shows the methodology of the proposed approach.

Steps 1 to 5 as the pre-processing phase (red box) are similar to the traditional methodology of duplicate bug report detection (DBRD), but the splitting data for evaluation is held on Step 6 here (box 6). The pairs of bug reports must be divided into two different parts to train and test machine learning (ML) algorithms. It is better to split data samples by considering the distribution of duplicated pairs in both parts, which have the same percentage of duplicated pairs. After splitting the pairs of bug reports, the training phase (green box) starts, which uses the training pairs for feature extraction (box 6.1.2) but without textual features. Then an ML algorithm is used to train the non-textual features of pairs of bug reports (box 6.1.4).

On the other hand, textual features of pairs must be extracted, too (box 6.1.6), and appended to non-textual features (box 6.1.8) till another ML algorithm can be trained for all kinds of features, including textual and non-textual features (box 6.1.10). Now we have two ML algorithms as duplicate item finders (DIF).

The test phase (blue box) will be started after the training phase is finished, and the test pairs will be feature extracted using non-textual features, too (boxes 6.2.1, 6.2.2, and 6.2.3). It is time to evaluate testing pairs' features using the first light DIF (box 7). The results of ML algorithms usually contain two values: 1) The predicted label, which is the status of a pair of bug reports as duplicated or non-duplicated here; 2) The confidence of the ML algorithm for this prediction. Every ML algorithm can predict the confidence level in a customized method. For example, the Naïve Bayes confidence is the algorithm's direct calculated probability when the confidence value is real. The k-NN confidence is the number of the *k* neighbors with the predicted class divided by *k*, and the single values are weighted by distance in weighted predictions. The SVM has a reasonable estimation of a binomial class problem's positive class (14), where the function_value is the SVM prediction. This approach is also used by the RapidMiner tool [54].

$$Confidence = \frac{1}{1+e^{-function\_value}} \tag{14}$$

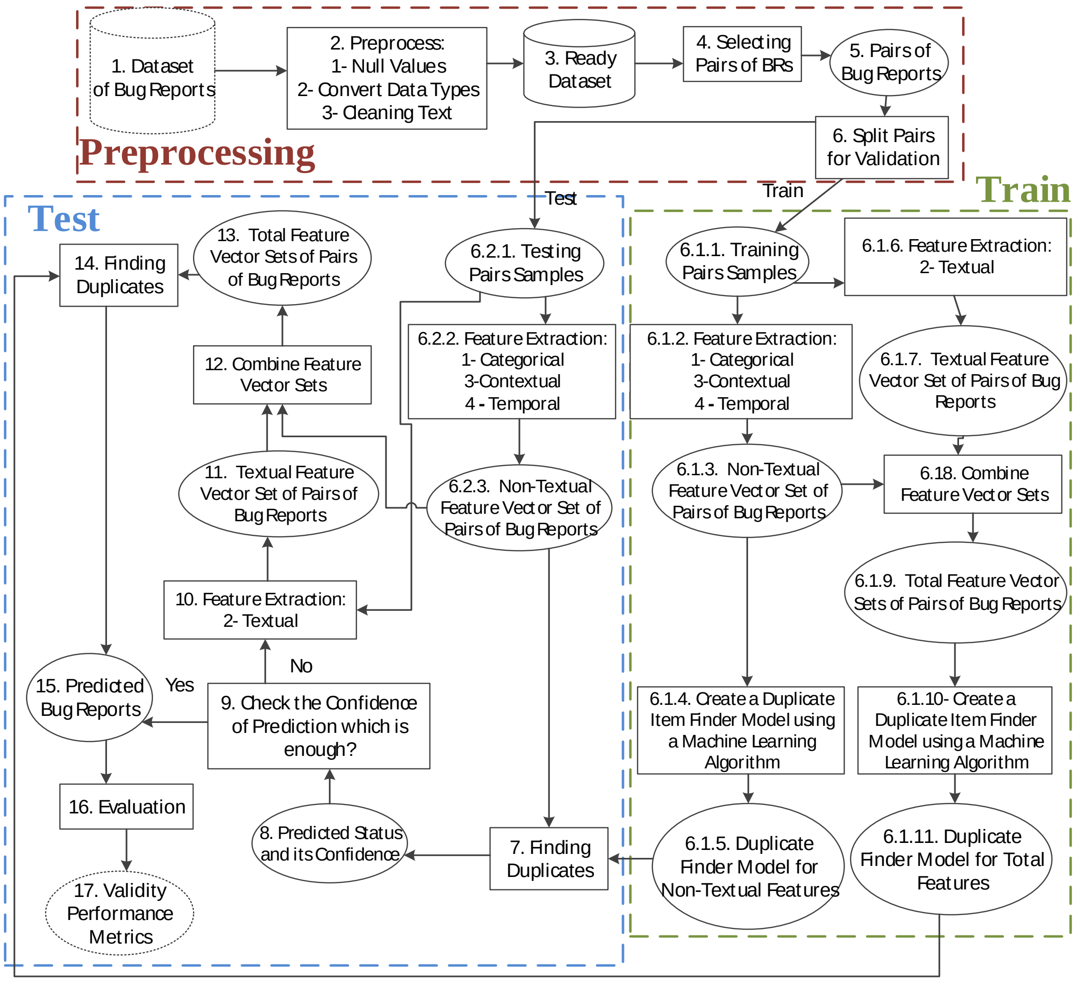

Preprocessing
1. Dataset of Bug Reports
2. Preprocess: 1- Null Values 2- Convert Data Types 3- Cleaning Text
3. Ready Dataset
4. Selecting Pairs of BRs
5. Pairs of Bug Reports
6. Split Pairs for Validation
Test
Train
6.1.1. Training Pairs Samples
6.1.6. Feature Extraction: 2- Textual
6.1.2. Feature Extraction: 1- Categorical 3-Contextual 4 - Temporal
6.1.7. Textual Feature Vector Set of Pairs of Bug Reports
6.1.3. Non-Textual Feature Vector Set of Pairs of Bug Reports
6.18. Combine Feature Vector Sets
6.1.9. Total Feature Vector Sets of Pairs of Bug Reports
6.1.4. Create a Duplicate Item Finder Model using a Machine Learning Algorithm
6.1.10- Create a Duplicate Item Finder Model using a Machine Learning Algorithm
6.1.5. Duplicate Finder Model for Non-Textual Features
6.1.11. Duplicate Finder Model for Total Features
6.2.1. Testing Pairs Samples
6.2.2. Feature Extraction: 1- Categorical 3-Contextual 4 - Temporal
6.2.3. Non-Textual Feature Vector Set of Pairs of Bug Reports
7. Finding Duplicates
8. Predicted Status and its Confidence
9. Check the Confidence of Prediction which is enough?
No
Yes
10. Feature Extraction: 2- Textual
11. Textual Feature Vector Set of Pairs of Bug Reports
12. Combine Feature Vector Sets
13. Total Feature Vector Sets of Pairs of Bug Reports
14. Finding Duplicates
15. Predicted Bug Reports
16. Evaluation
17. Validity Performance Metrics

Figure 2. The methodology of Duplicate Bug Report Detection using the proposed two-step Classification Approach of Machine Learning Algorithms

Now it is time to check the confidence of predicted status (box 9). If the confidence is more than a specific threshold, e.g., 90%, the prediction can be accepted; otherwise, the textual features of that pair should be extracted too (box 10), and the combined features (boxes 12 and 13), including textual and non-textual features of that pair of the testing set must be used for second DIF (box 6.1.11), which considers all kinds of features to predict the status of this pair (box 14). Then the predicted status (box 15) is used to compare the real status of that pair (box 16) and evaluate the validation performance metrics of DIF as the final result of DBRD (box 17).

For example, consider following three real bug reports in Table 3 from Eclipse dataset [4, 46, 55], where the two first ones already exist in the bug reports database and the third one is a new or target bug report that is compared to other existing bug reports. Their identical fields include bug report id and master id which is the bug report id of main bug report for duplicate bug reports and it is null for those bug reports which are not duplicate. The categorical fields determine detail categories for each bug report like the software product and component, The status field shows the last state of bug report which can be new, assigned to developer for fixing, fixed, duplicate, and so on. The textual fields are the main fields of every bug report because they are the main fields to find uniqueness or duplication of each bug report.

The contextual fields are derived from textual fields and as mentioned before, can be calculated and stored once time to be used later for feature extraction in comparison with other bug reports. There can be more contextual fields in various domains based on the software triage system modules or external aspects like software engineering topics. The selected example consider four general, networking, cryptography, and java conexts to calculate contextual fields, but it is possible to build dictionaries based on each module of software triage system and calculate contextual fields for those topics based on built dictionaries.

Table 4 shows some extracted features for comparing the new bug report to existing bug reports in the database. The class label shows that the selected pair is really duplicate or not based on master ID field in the dataset. For training dataset, the master ID fields are filled, so the training dataset including training pairs have deterministic label. In test phase, the label should be predicted using machine learning algorithms. Various types of features based on state-of-the-art are calculated and determined in Table 4 and their name and equations are referenced in second column. The values of the features are shown in two last columns.

Table 3. Real Sample Bug Reports Data

Field Type	**Field**	**Bug Report 1**	**Bug Report 2**	**Target Bug Report**
Identical	Bug ID	240427	258365	258935
	MasterID	233269	-	258365
Categorical	Product	Equinox	Equinox	Equinox
	Component	P2	P2	P2
	Type	Normal	Major	Normal
	Priority	3	3	3
	Version	3.4	3.5	3.5
	Status	Duplicate	Fixed	Duplicate
Temporal	Open Date (GMT)	11/7/2008 00:25:00	10/12/2008 21:33:00	16/12/2008 14:18:00
	Close Date (GMT)	14/7/2008 15:27:48	21/1/2010 07:12:35	18/12/2008 03:52:48
Textual	Title	software update dialog / filter field blocks user input	[fwkadmin][shared] shared install eclipse.ini not read	[shared] shared tests are failing on mac
	Description	menu: help->software updates displays a dialog with available software to install. in the top part there is a filter field. when typing a text into this filter eclipse starts to immediately applying the filter and whole dialog is block - what blocks possibility to continue typing into the filter field. there is certain delay but it is too short to be able to type in you filter expression. it almost imposible to use the filter field resonably.	i20081210-0800 when i install something in shared install i loose my p2 menus.	n20081215 testreadonlydropinsstartup and testuserdropinsstartup are failing
Contextual (Derived from textual)	General	26.858	8.973	4.298
	Networking	23.838	6.821	4.298
	Crypto graphy	21.514	5.501	2.325
	Java	22.946	3.291	4.298

Table 4. Real sample extracted features

Features Type	**Features**	**Target Bug Report vs Bug Report 1**	**Target Bug Report vs Bug Report 2**
Label	Duplicate	No	Yes
Textual	f_{BM25F}-1G (3)	1.407	1.786
	f_{BM25F}-2G (3)	1.270	2.395
	$f_{SizeDiff}$ (4)	386	17
	f_{LCS} [5, 6]	83	59
Temporal	f_{Id} (5)	18508	570
	f_{Date} (6) (sec)	13387812	34620875
Categorical	$f_{product}$ (7)	1	1
	$f_{company}$ (8)	1	1
	f_{type} (9)	1	0
	$f_{PriorityA}$ (10)	1	1
	$f_{PriorityS}$ (11)	1	1
	$f_{VersionA}$ (12)	0.9	1
	$f_{VersionS}$ (13)	1.1	1
Contextual (Distance)	Cosine	0.984	0.937
	General Manhattan	22.560	4.674
	Networking Manhattan	19.539	2.522
	Cryptography Manhattan	19.188	3.175
	Java Manhattan	18.647	1.006

It should be mentioned that textual features are very important for DBRD, but their time complexity is more than other feature types and depends on the length of texts. For example, the minimum, average and maximum text length of Eclipse dataset is 8, 1080, and 65,054 characters, and 2, 136, and 10,762 words, respectively. A pretest for comparing bug report 259801 with 697 characters and 130 words to more than 18,000 other bug reports shows that the minimum, average, and maximum runtime of all non-textual features calculation were 0, 2.8 and 100.8 micro seconds for each pair. These times for textual features were 0.6, 11.3 and 968.1 milli seconds which are four thousand times more than non-textual features times. If the selected bug report length is more, the runtime will be increased a lot too. So, textual features are harmful for runtime performance, and useful for validation performance of DBRD.

After feature extraction, the feature vectors including some numerical values and a label will be made and given to a ML algorithm to be trained and learn features of duplicated pairs. In the test phase, a feature vector will be provided in comparison of new bug report with other existing bug reports in the triage system. Then each feature vector will be given to the trained ML algorithm to predict its label. In the proposed approach, just non textual features will be calculated in the first stage and they will be given to simple or light or non-textual ML algorithm. Adside the predicted label by ML algorithm, the confidence of ML algorithm will be checked. If the confidence is more than a certain threshold (e.g., 90%), the predicted label based on non-textual features will be accepted and reported as the final result. Otherwise, the textual features will be calculated and appended to feature vector and the new full feature vector will be given to heavy or full ML algorithm and its result will be reported as the final result.

Furthermore, the DBRD process is divided into two parts based on the textual features. There are some considerations to improve the validation performance of this two-step classification approach as a DBRD:

1. Using more robust non-textual features to improve the validation performance of non-textual DIF, e.g., using more topics for contextual features [7];
2. Using robust and powerful ideas for ML algorithms of first DIF, e.g., ensemble algorithms like using some ML algorithms and voting their results to improve the validation performance of non-textual DIF;
3. Using an ML algorithm like linear regression to predict the best value for the threshold of confidence checking step (box 9).

4. Experimental results

The traditional and proposed methodologies of duplicate bug report detection (DBRD) are implemented using Takelab script [56] in Python 3.8 for textual feature extraction and RapidMiner 9.5 [57] for implementing the machine learning algorithms. The state-of-the-art approach is the most commonly used ML-based DBRD [5, 7, 8, 14, 20, 21, 30, 46, 50]. In the first experiment, we use all the ML algorithms based on [50] as the best results between related works for comparison. Besides, the ID difference feature was the most important feature that improves the validation performance results a lot, so it was eliminated because there may be some biased judgment in considering a relation between ID difference and duplication status. However, the open date difference was kept as a temporal feature. The results were much realistic, and there is hope that there is no more difference between the proposed approach and the triager needs in the real world. Figure 3 shows the results of the comparison of various ML algorithms for different scenarios. The results of the scenario of simple features (S) must be worse than the scenario of full features (F), but, interestingly, the results of the scenario of two-step classification (TSC) are in the middle of other scenarios that are more than 87% and 89% for both accuracy and F1-measure metrics, respectively. The ML algorithm of TSC is a voting-based ensemble algorithm of other ML algorithms; that is, S-Vote for non-textual duplicate item finder (DIF) and F-Vote for full DIF. Even though the deep learning ML algorithm has better performance in both simple and full feature scenarios, the TSC just tested using the voting algorithm because deep learning training is time-consuming at ist improvement is less than one percent for both simple and full feature scenarios.

Table 5 shows the experiments' parameters. Various machine learning (ML) algorithms are chosen to compare their efficiency for DBRD in three different scenarios with non-textual features, full features, or the proposed two-step classification. Three scenarios are considered for evaluating the proposed method, including: 1) Simple or light scenario just including the non-textual features as an old approach of state-of-the-art; 2) Full feature as the current state-of-the-art approach; 3) Two-Step Classification (TSC) as the proposed approach.

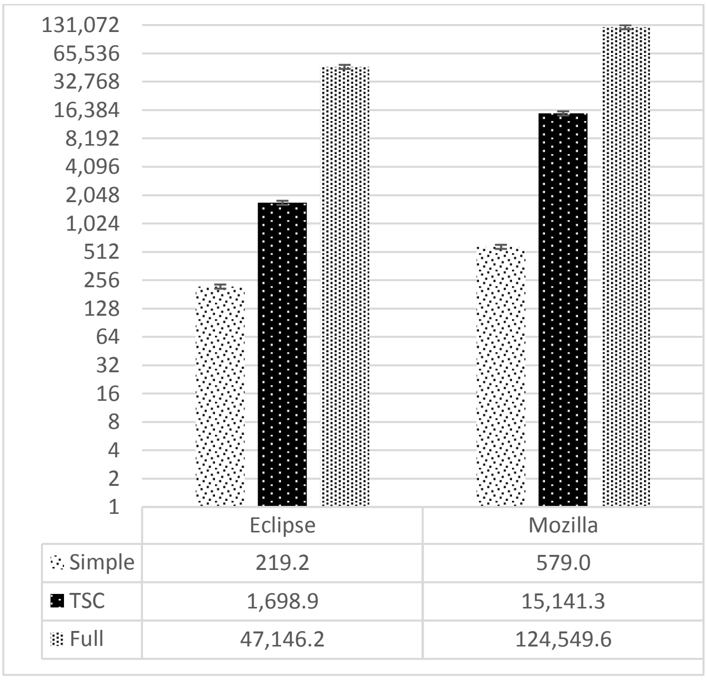
Moreover, the detailed properties of datasets are tabulated in the control variable section of The state-of-the-art approach is the most commonly used ML-based DBRD [5, 7, 8, 14, 20, 21, 30, 46, 50]. In the first experiment, we use all the ML algorithms based on [50] as the best results between related works for comparison. Besides, the ID difference feature was the most important feature that improves the validation performance results a lot, so it was eliminated because there may be some biased judgment in considering a relation between ID difference and duplication status. However, the open date difference was kept as a temporal feature. The results were much realistic, and there is hope that there is no more difference between the proposed approach and the triager needs in the real world. Figure 3 shows the results of the comparison of various ML algorithms for different scenarios. The results of the scenario of simple features (S) must be worse than the scenario of full features (F), but, interestingly, the results of the scenario of two-step classification (TSC) are in the middle of other scenarios that are more than 87% and 89% for both accuracy and F1-measure metrics, respectively. The ML algorithm of TSC is a voting-based ensemble algorithm of other ML algorithms; that is, S-Vote for non-textual duplicate item finder (DIF) and F-Vote for full DIF. Even though the deep learning ML algorithm has better performance in both simple and full feature scenarios, the TSC just tested using the voting algorithm because deep learning training is time-consuming at ist improvement is less than one percent for both simple and full feature scenarios.

Table 5, which indicates the number of bug reports in each dataset and the number of selected bug report pairs in step four of both state-of-the-art and the proposed methodologies. The K-fold cross-validation is used for the evaluation of ML algorithms to avoid biased results. Moreover, various kinds of features are extracted for duplicate detection.

The state-of-the-art approach is the most commonly used ML-based DBRD [5, 7, 8, 14, 20, 21, 30, 46, 50]. In the first experiment, we use all the ML algorithms based on [50] as the best results between related works for comparison. Besides, the ID difference feature was the most important feature that improves the validation performance results a lot, so it was eliminated because there may be some biased judgment in considering a relation between ID difference and duplication status. However, the open date difference was kept as a temporal feature. The results were much realistic, and there is hope that there is no more difference between the proposed approach and the triager needs in the real world. Figure 3 shows the results of the comparison of various ML algorithms for different scenarios. The results of the scenario of simple features (S) must be worse than the scenario of full features (F), but, interestingly, the results of the scenario of two-step classification (TSC) are in the middle of other scenarios that are more than 87% and 89% for both accuracy and F1-measure metrics, respectively. The ML algorithm of TSC is a voting-based ensemble algorithm of other ML algorithms; that is, S-Vote for non-textual duplicate item finder (DIF) and F-Vote for full DIF. Even though the deep learning ML algorithm has better performance in both simple and full feature scenarios, the TSC just tested using the voting algorithm because deep learning training is time-consuming at ist improvement is less than one percent for both simple and full feature scenarios.

Table 5. The parameters of experiments

Variable Type	**Variable Name**	**Variable States (Values)**				
Independent	Classifier	Linear Regression (LR), Decision Tree (DT), Random Forest (RF), Deep Learning with H2O (DL) [58, 59], Voting of all mentioned Classifiers as an Ensemble approach (Vote)				
	Scenarios	Simple (S): using the non-textual features Full Features (F): as the traditional approach Two-Step Classification approach (TSC): the proposed approach				
Control	Dataset	Eclipse and Mozilla [4, 46, 55]				
	Number of Bug Reports	Dataset	# Bug Reports			
		Eclipse	45,234			
		Mozilla	75,648			
	Number of Bug Pairs	Pairs→ /Dataset ↓	Duplicates	None-Duplicates	Total	Dup%
		Eclipse	15,219	5,536	20,755	26.6%
		Mozilla	40,537	14,297	54,834	26.0%
		Total	55,756	19,833	75,589	26.3%
	K-fold	10				
	Stemming	Is used				
	Features ollection	Temporal, Categorical, Contextual [7, 46], Textual [56]				



	Eclipse	Mozilla
⁘ Simple	219.2	579.0
■ TSC	1,698.9	15,141.3
⁙ Full	47,146.2	124,549.6

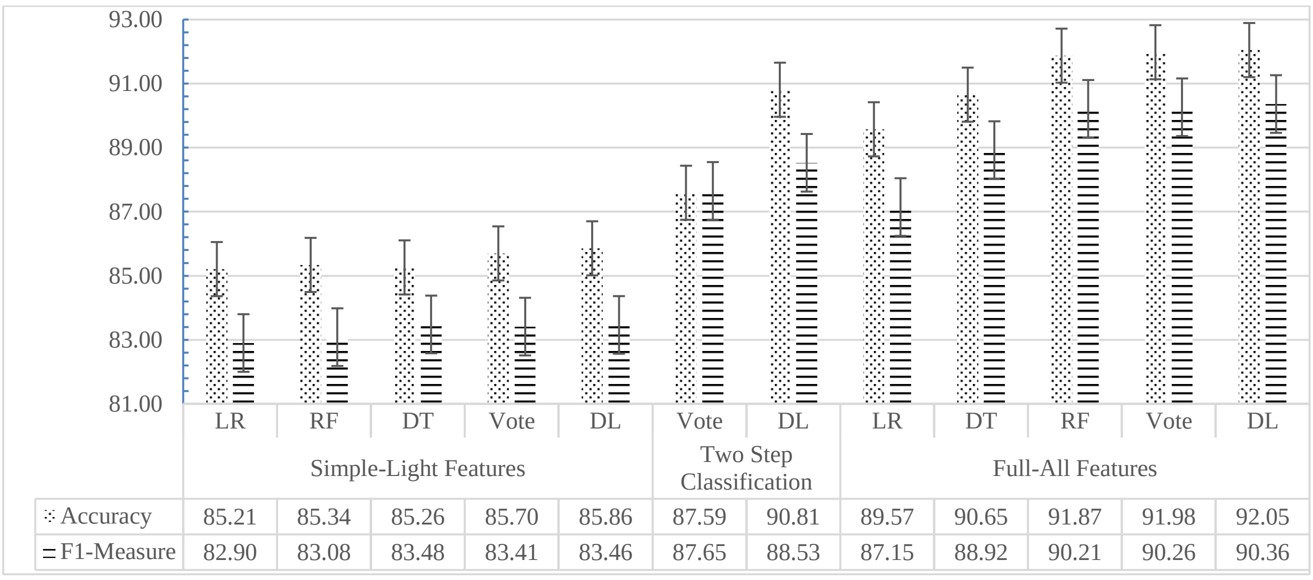
Figure 3. The runtime of three scenarios for both datasets based on seconds in logarithmic scale

Table 6. The maximum performance of different machine learning algorithms for various kinds of scenarios of classification

Scenario→	**State-ot-the-art Classification**		**Two-step Classification**		**Full Features [50]**	
Dataset ↓	Accuracy	F1-measure	Accuracy	F1-measure	Accuracy	F1-measure
Eclipse	87.33%	77.67%	88.05%	86.43%	91.53%	84.75%
Mozilla	84.26%	89.34%	87.34%	91.60%	90.98%	93.80%
Average	85.80%	83.51%	87.70%	89.01%	91.26%	89.28%

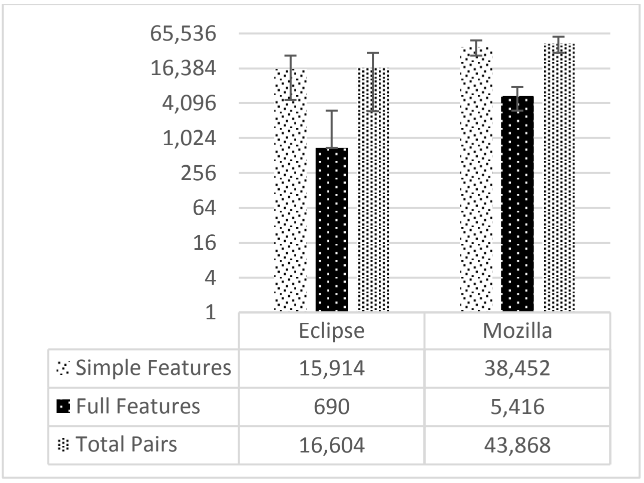
Table 6 shows the detailed results on the experiments' maximum performance for each dataset. The results show that the TSC validation performance is almost in the middle of both scenarios in various datasets even though the TSC is implemented only using the Vote-based ML, but those are compared to their best ML algorithms.

Although no one expects the results of TSC to be less than the results of simple scenario, it is time to know the impact of TSC on runtime performance. The number of pairs of bug reports is used for runtime comparison instead of execution time to eliminate hardware configuration impact on the results and have a better insight about the time complexity improvement. Using a logarithmic scale to better show value contrast, Table 4 shows the number of used features using the first and second DIFs. The first DIF uses Simple Features for classification, and the second DIF uses Full Features, as mentioned in Table 3. The results show that many bug report pairs can be classified using the first classifier, and just a few pairs need the complex textual feature extraction phase. Non-textual features can be extracted in less than a millisecond, but textual feature sometimes takes more than 5 seconds to be extracted for just a pair based on their text lengths.



	Simple-Light Features					Two Step Classification		Full-All Features				
	LR	RF	DT	Vote	DL	Vote	DL	LR	DT	RF	Vote	DL
⁙ Accuracy	85.21	85.34	85.26	85.70	85.86	87.59	90.81	89.57	90.65	91.87	91.98	92.05
= F1-Measure	82.90	83.08	83.48	83.41	83.46	87.65	88.53	87.15	88.92	90.21	90.26	90.36

Figure 4. The average validation performance of various scenarios of Table 5



	Eclipse	Mozilla
⁙ Simple Features	15,914	38,452
■ Full Features	690	5,416
⁘ Total Pairs	16,604	43,868

Figure 5. The number of bug reports which can be detected fast using simple features versus full features scenario (including textual features)

Table 7. Percentage of bug reports predicted for classification

Time Complexity Improvement → / Dataset ↓	**using Non-Textual Features**	**using All Features**	**Total Number of Bug Reports (100%)**
Eclipse	95.85%	4.15%	16,604
Mozilla	87.65%	12.35%	43,868
Average of Results	89.90%	10.10%	

The values are converted to percentage in Table 7. It shows that 87% and 95% of pairs of bug reports can be classified faster than the traditional approach using non-textual features for Eclipse and Mozilla datasets, respectively. These predictions' average validation performance was 87% and 89% for accuracy and F1-measure, respectively, which are relatively more than many related works [48].

Furthermore, 89.9% of pairs of bug reports averagely in both datasets can be classified with more than 87% accuracy and F1-measure, using simple checking of the categorical, temporal, and contextual features. The contextual features can be calculated. At first a new bug report is inserted in the repository to improve the performance of DBRD. So, the DBRD can be implemented merely using a SQL query in the repository for almost all bug reports, and those which are suspicious and need more checking, can be sent for textual feature extraction and give the full features vector of those to the full DIF.

5. Conclusion

This study focused on the runtime performance of the process of duplicate bug report detection (DBRD). A novel two-step classification method was proposed for DBRD, which uses non-textual features in the first step to check the duplication of a pair of bug reports. A machine learning (ML) algorithm is trained as a duplicate item finder (DIF) to predict the duplication status of non-textual feature vectors of pairs of bug reports. If the first DIF has low confidence in its prediction, the textual features should be extracted, and the second DIF is used to predict the status of the pair based on all features, especially textual features. The experiments show that the validation performance results of the proposed approach are better than those using the first non-textual DIF alone. Moreover, the runtime performance results of the proposed approach are better than using the second DIF alone. So, the proposed approach has a good runtime and validation performance in comparison with the traditional approaches. Every non-textual feature, like more contextual features, can improve the first DIF validation performance in the future. Also, the threshold of the first DIF for switching to the second DIF can be improved. Other datasets can be used to evaluate their validation performance. A semi-supervised [60] machine learning algorithm can be used for an incremental bug report repository of software triage systems.

6. References

[1] Zhang, J., Wang, X., Hao, D., Xie, B., Zhang, L., and Mei, H., "A survey on bug-report analysis", Science China Information Sciences, journal article vol. 58, no. 2, pp. 1-24, doi: 10.1007/s11432-014-5241-2. Science China Press, February 01, 2015.

[2] Soleimani, Neysiani, B., and Babamir, S. M., "Methods of Feature Extraction for Detecting the Duplicate Bug Reports in Software Triage Systems", presented at the International Conference on Information Technology, Communications and Telecommunications (IRICT), Tehran, Iran, 2016, 2016. [Online]. Available: http://www.sid.ir/En/Seminar/ViewPaper.aspx?ID=7677.

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

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

[5] Soleimani Neysiani, B., and Babamir, S. M., "Improving Performance of Automatic Duplicate Bug Reports Detection Using Longest Common Sequence", in IEEE 5th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, Iran, Vol. 5, 2019.

[6] Banerjee, S., Cukic, B., and Adjeroh, D., "Automated duplicate bug report classification using subsequence matching", in IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE), IEEE, pp. 74-81, doi: http://dx.doi.org/10.1109/HASE.2012.38, 2012.

[7] Soleimani Neysiani, B., and Babamir, S. M., "New Methodology of Contextual Features Usage in Duplicate Bug Reports Detection", in IEEE 5th International Conference on Web Research (ICWR), Tehran, Iran, Vol. 5, 2019.

[8] Aggarwal, K., Rutgers, T., Timbers, F., Hindle, A., Greiner, R., and Stroulia, E., "Detecting duplicate bug reports with software engineering domain knowledge", in IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER), Montreal, IEEE, pp. 211-220, doi: http://dx.doi.org/10.1109/SANER.2015.7081831, QC 2015.

[9] Aggarwal, K., Timbers, F., Rutgers, T., Hindle, A., Stroulia, E., and Greiner, R., "Detecting duplicate bug reports with software engineering domain knowledge", Journal of Software: Evolution and Process, Vol. 29, No. 3, pp. e1821-n/a, Art no. e1821, doi: 10.1002/smr.1821, 2017.

[10] Soleimani Neysiani, B., and Babamir, S. M., "Automatic Typos Detection in Bug Reports," presented at the IEEE 12th International Conference Application of Information and Communication Technologies, Kazakhstan, 2018.

[11] Soleimani Neysiani, B., and Babamir, S. M., "Automatic Interconnected Lexical Typo Correction in Bug Reports of Software Triage Systems", presented at the International Conference on Contemporary Issues in Data Science, Zanjan, Iran, 2019.

[12] Soleimani Neysiani, B., and Babamir, S. M., "Fast Language-Independent Correction of Interconnected Typos to Finding Longest Terms", presented at the 24th International Conference on Information Technology (IVUS), Lithuania, 2019.

[13] Soleimani Neysiani, B., and Babamir, S. M., "New labeled dataset of interconnected lexical typos for automatic correction in the bug reports", SN Applied Sciences, Vol. 1, No. 11, pp. 1385, 2019.

[14] Soleimani Neysiani, B., and Babamir, S. M., "Effect of Typos Correction on the validation performance of

Duplicate Bug Reports Detection", presented at the 10th International Conference on Information and Knowledge Technology (IKT), Tehran, Iran, 2020-1-2, 1157, 2019.

[15] Soleimani Neysiani, B., and Babamir, S. M., "Duplicate Detection Models for Bug Reports of Software Triage Systems: A Survey", Current Trends In Computer Sciences & Applications, Review Article, Vol. 1, No. 5, pp. 128-134, 11-22 2019, doi: 10.32474/CTCSA.2019.01.000123, 2019.

[16] Soleimani Neysiani, B., and Babamir, S. M., "Automatic Duplicate Bug Report Detection using Information Retrieval-based versus Machine Learning-based Approaches", in IEEE 6th International Conference on Web Research (ICWR), Tehran, Iran, Vol. 6, pp. 288-293, doi: 10.1109/ICWR49608.2020.9122288, 2020.

[17] Hindle, A., "Stopping duplicate bug reports before they start with Continuous Querying for bug reports", PeerJ Preprints, 2167-9843, 2016.

[18] Hindle, A., and Onuczko, C., "Preventing duplicate bug reports by continuously querying bug reports," Empirical Software Engineering, pp. 1-35, 2018.

[19] Soleimanian Gharehchopogh, F., and Mousavi, S. K., "A New Feature Selection in Email Spam Detection by Particle Swarm Optimization and Fruit Fly Optimization Algorithms", Journal of Computer and Knowledge Engineering, Vol. 2, No. 2, pp. 49-62, 2020-02-11, doi: 10.22067/cke.v2i2.81750, 2020.

[20] Soleimani Neysiani, B., Doostali, S., Babamir, S. M., and Aminoroaya, Z., "Fast Duplicate Bug Reports Detector Training using Sampling for Dimension Reduction: Using Instance-based Learning for Continous Query in Real-World", presented at the 11th International (Virtual) Conference on Information and Knowledge Technology (IKT), Tehran, Iran, 22-23 Dec. 2020, 2020.

[21] Banerjee, S., Syed, Z., Helmick, J., Culp, M., Ryan, K., and Cukic, B., "Automated triaging of very large bug repositories," Information and Software Technology, Vol. 89, pp. 1-13, 2017/09/01, doi: https://doi.org/10.1016/j.infsof.2016.09.006, 2017.

[22] Yang, X., Lo, D., Xia, X., Bao, L., and Sun, J., "Combining word embedding with information retrieval to recommend similar bug reports," in IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), IEEE, pp. 127-137, 2016.

[23] Lin, M.-J., Yang, C.-Z., Lee, C.-Y., and Chen, C.-C., "Enhancements for duplication detection in bug reports with manifold correlation features", Journal of Systems and Software, Vol. 121, No. Supplement C, pp. 223-233, 2016/11/01, doi: https://doi.org/10.1016/j.jss.2016.02.022, 2016.

[24] Budhiraja, A., Dutta, K., Reddy, R., and Shrivastava, M., "DWEN: deep word embedding network for duplicate bug report detection in software repositories", in Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ACM, pp. 193-194, 2018.

[25] Lazar, A., Ritchey, S., and Sharif, B., "Improving the accuracy of duplicate bug report detection using textual similarity measures", in MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad, India ACM, pp. 308-311, doi: 10.1145/2597073.2597088. [Online]. Available: http://icse2014.acm.org/, 2014.

[26] Wang, S., Khomh, F., and Zou, Y., "Improving bug localization using correlations in crash reports," in 10th IEEE Working Conference on Mining Software Repositories (MSR) IEEE, pp. 247-256, doi: http://dx.doi.org/10.1109/MSR.2013.6624036, 2013.

[27] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J., "An approach to detecting duplicate bug reports using natural language and execution information", in Proceedings of the 30th international conference on Software engineering, Leipzig, Germany, ACM, in ICSE '08, pp. 461-470, doi: http://doi.acm.org/10.1145/1368088.1368151, 2008.

[28] Kim, S., Zimmermann, T., and Nagappan, N., "Crash graphs: An aggregated view of multiple crashes to improve crash triage", in Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on, IEEE, pp. 486-493, 2011.

[29] Ebrahimi, N., Trabelsi, A., Islam, M. S., Hamou-Lhadj, A., and Khanmohammadi, K., "An HMM-based approach for automatic detection and classification of duplicate bug reports", Information and Software Technology, Vol. 113, pp. 98-109, 2019/09/01, doi: https://doi.org/10.1016/j.infsof.2019.05.007, 2019.

[30] Alipour, A., Hindle, A., and Stroulia, E., "A Contextual Approach Towards More Accurate Duplicate Bug Report Detection", in Proceedings of the 10th Working Conference on Mining Software Repositories, San Francisco, CA, USA, IEEE Press, pp. 183-192, doi: 10.1109/MSR.2013.6624026. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487123, 2013.

[31] Nguyen, A. T., Nguyen, T. T., Nguyen, T. N., Lo, D., and Sun, C., "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), IEEE, pp. 70-79, 2012.

[32] Bagal, P. V., *et al.*, "Duplicate bug report detection using machine learning algorithms and automated feedback incorporation", Patent US 2017/01998.03 A1, 2017.

[33] Koochekian Sabor, K., Hamou-Lhadj, A., and Larsson, A., "DURFEX: A Feature Extraction Technique for Efficient Detection of Duplicate Bug Reports", in 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, IEEE, pp. 240-250, doi: 10.1109/QRS.2017.35, 25-29 July, 2017.

[34] Deshmukh, J., Podder, S., Sengupta, S., and Dubash, N., "Towards Accurate Duplicate Bug Retrieval Using Deep Learning Techniques", in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp. 115-124, 2017.

[35] Ebrahimi Koopaei, N., "Machine Learning And Deep Learning Based Approaches For Detecting Duplicate Bug Reports With Stack Traces", Concordia University, 2019.

[36] Xie, Q., Wen, Z., Zhu, J., Gao, C., and Zheng, Z., "Detecting Duplicate Bug Reports with Convolutional Neural Networks", in 2018 25th Asia-Pacific Software Engineering Conference (APSEC), 4-7 Dec. 2018, pp. 416-425, doi: 10.1109/APSEC.2018.00056, 2018.

[37] Aminoroaya, Z., Soleimani Neysiani, B., and Nadimi Shahraki, M. H., "Detecting Duplicate Bug Reports Techniques", Research Journal of Applied Sciences, Vol. 13, No. 9, pp. 522-531, 2018/09/30, 2018.

[38] Bettenburg, N., Premraj, R., Zimmermann, T., and Kim, S., "Duplicate bug reports considered harmful… really?", in IEEE International Conference on Software Maintenance (ICSM), IEEE, pp. 337-345, doi: http://dx.doi.org/10.1109/ICSM.2008.4658082, [Online]. Available: https://www.st.cs.uni-saarland.de/softevo/, 2008.

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

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

[41] Liu, K., Tan, H. B. K., and Chandramohan, M., "Has this bug been reported?", in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ACM, p. 28, doi: 10.1109_wcre.2013.6671283, 2012.

[42] Alipour, A., "A Contextual Approach Towards More Accurate Duplicate Bug Report Detection", Master of Science, Department of Computing Science, University of Alberta, Faculty of Graduate Studies and Research, 2013.

[43] Feng, L., Song, L., Sha, C., and Gong, X., "Practical duplicate bug reports detection in a large web-based development community", in Asia-Pacific Web Conference, Springer, pp. 709-720, 2013.

[44] Tsuruda, A., Manabe, Y., and Aritsugi, M., "Can We Detect Bug Report Duplication with Unfinished Bug Reports?", in Asia-Pacific Software Engineering Conference (APSEC), IEEE, pp. 151-158, 2015.

[45] Sharma, A., and Sharma, S., "Bug Report Triaging Using Textual, Categorical and Contextual Features Using Latent Dirichlet Allocation", International Journal for Innovative Research in Science and Technology (IJIRST), Vol. 1, No. 9, pp. 85-96, Feb, 2015.

[46] Hindle, A., Alipour, A., and Stroulia, E., "A contextual approach towards more accurate duplicate bug report detection and ranking", Empirical Software Engineering, journal article, Vol. 21, No. 2, pp. 368-410, doi: 10.1007/s10664-015-9387-3, April 01, 2016.

[47] Pasala, A., Guha, S., Agnihotram, G., Prateek B, S., and Padmanabhuni, S., "An Analytics-Driven Approach to Identify Duplicate Bug Records in Large Data Repositories," in Data Science and Big Data Computing: Frameworks and Methodologies, Z. Mahmood Ed. Cham: Springer International Publishing, pp. 161-187, 2016.

[48] Rakha, M. S., Shang, W., and Hassan, A. E., "Studying the needed effort for identifying duplicate issues", Empirical Software Engineering, journal article, Vol. 21, No. 5, pp. 1960-1989, October 01, doi: 10.1007/s10664-015-9404-6, 2016.

[49] Su, E., and Joshi, S., "Leveraging product relationships to generate candidate bugs for duplicate bug prediction", in Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ACM, pp. 210-211, 2018.

[50] Soleimani Neysiani, B., Babamir, S. M., and Aritsugi, M., "Efficient Feature Extraction Model for Validation Performance Improvement of Duplicate Bug Report Detection in Software Bug Triage Systems", Information and Software Technology, vol. 126, pp. 106344-106363, 2020/10/01 2020, doi: 10.1016/j.infsof.2020.106344.

[51] Kukkar, A., Mohana, R., Kumar, Y., Nayyar, A., Bilal, M., and Kwak, K., "Duplicate Bug Report Detection and Classification System based on Deep Learning Technique", IEEE Access, Vol. 8, pp. 200749-200763, 10/23, doi: 10.1109/ACCESS.2020.3033045, 2020.

[52] Kim, T., and Yang, G., "Predicting Duplicate in Bug Report Using Topic-Based Duplicate Learning With Fine Tuning-Based BERT Algorithm", IEEE Access, Vol. 10, pp. 129666-129675, doi: 10.1109/ACCESS.2022.3226238, 2022.

[53] Zhang, T., et al., "Duplicate Bug Report Detection: How Far Are We?", ACM Transactions on Software Engineering and Methodology, doi: 10.1145/3576042, 2022.

[54] IngoRM., "Confidence values", RapidMiner. https://community.rapidminer.com/discussion/17058/confidence-values, accessed 12/10/2020, 2020.

[55] Alipour, A., Hindle, A., Rutgers, T., Dawson, R., Timbers, F., and Aggarwal, K., "Bug Reports Dataset", https://github.com/kaggarwal/Dedup, accessed.

[56] Šarić, F., Glavaš, G., Karan, M., Šnajder, J., and Bašić, B. D., "Takelab: Systems for measuring semantic text similarity", in Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation, Montréal, Canada, Stroudsburg, PA, USA: Association for Computational Linguistics, in SemEval '12, pp. 441-448, doi: 10.5555/2387636.2387708. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387636.2387708, 2012.

[57] RapidMiner Studio (9.5.1) RapidMiner Inc. [Online]. Available: rapidminer.com, , (2019).

[58] Candel, A., Parmar, V., LeDell, E., and Arora, A., "Deep learning with H_2O", H_2O. ai Inc, 2016.

[59] Cook, D., "Practical machine learning with H_2O: powerful, scalable techniques for deep learning and AI", O'Reilly Media, Inc.", 2016.

[60] Karimi Zandian, Z., and Keyvanpour, M. R., "SSLBM: A New Fraud Detection Method Based on Semi-Supervised Learning", Journal of Computer and Knowledge Engineering, Vol. 2, No. 2, pp. 10-18, 2020-02-26, doi: 10.22067/cke.v2i2.82152, 2020.