Spring 3-1-2022

Bug Triage Automation Approaches: A Comparative Study

Dr khaled nagaty
khaled.nagaty@bue.edu.eg

Madonna Mayez
The British University in Egypt, Madonna.Mayez@bue.edu.eg

Khaled Ahmed Nagay Dr.
The British University in Egypt, khaled.nagaty@bue.edu.eg

Bug Triage Automation Approaches: A Comparative Study

Madonna Mayez
The British University in Egypt, Egypt

Abeer Hamdy
The British University in Egypt, Egypt

Khaled Nagaty
The British University in Egypt, Egypt

ABSTRACT

Bug triage is an essential task in the software maintenance phase. It is the process of assigning a developer (fixer) to bug report. Triaging process is performed by the triager, who has to analyze developers' profiles and bug reports for the purpose of making a suitable assignment. Manual assignment consumes time, financial resources and human resources; to get a high-quality software with minimum cost, automating this process is necessary. Previous researchers tackled this problem as a classification problem from different perspectives, either information retrieval approach or machine learning algorithms, some researchers handled it as an optimization problem using optimization and assignment algorithms. This paper introduces deep analysis for previous studies and empirical comparison for five implemented classifiers on the same bug report repository. Results show that Linear SVM has the best performance compared with, Logistic regression, BernoulliNB, MultinomialNB and Decision Tree.

KEYWORDS: Bug triage automation, Machine leaning, Information retrieval.

1. INTRODUCTION

In development and maintenance phases in software development life cycle, issues and defects usually show up. Developers and testing team members use what is called bug report to document these defects. A bug report organizes defect information; such as bug id, summary, reporter name, severity, priority and submission date. Issue tracking systems (ITS) such as Bugzilla (Bugzilla, 2014) and Jira (Jira Software, n.d.) are used to organize and track the submitted reports. Bug triage process is a main step in handling bug reports. It assigns each bug to one of the developers in the ITS, who is qualified enough to fix the assigned bug (Alenezi, Banitaan, & Zarour, 2018).

As a real-life scenario, a coach takes the submitted bugs and starts analyzing the developers' profiles to assign each developer to a bug report based on their experience and the skills required in each bug report. Manual triage is a time-consuming process; the trigger, as a human, is not able commit to memory the qualifications of each developer and the skills required in each bug report; as a massive number of bug reports are submitted to ITS.

In a daily manner, the submitted bug reports are increasing. In November 2019, Eclipse ITS recorded 500,000 issue report (Anvik, Hiew, & Murphy, 2006) and 1,600,000 issue report are submitted in Mozilla repository. Also, 800,000 issue reports are received by Mozilla in October 2012, with around 300 new changes every day (Anjali, January 2015). These numbers indicate how this problem is sophisticated with respect to human resources and financial aspects. What makes it more complicated is tossing actions. Tossing a bug report is a process of reassigning it to another developer in case the first one failed to solve it. Around 37% of the received bug reports go in reassignment (tossing) process (Gondaliya, Peters, & Rueckert, 2018). These tossing actions not only waste financial and human resources but also it delays the fixing time. In other words, an overloaded developer can take much time to address the problem, also a less qualified developer may fail to fix the assigned bug. In both cases the bug requires longer time to be fixed. Accordingly, more human and financial recourses will be required. Therefore, tossing actions should be minimized as much as possible.

The essential point behind the triage problem is assigning the bug report to the developer who is definitely able to make the required changes. Given the above-mentioned statistics, it is clear that a triager cannot equally distribute the newly submitted bugs over the developers. In addition, there may be sever issue that have critical consequences on the whole project. Thus, incorrect assignments decisions may lead to an increase in the fixing time and cost. Also, inaccurate assignment between the developers and bug reports waist the human resources, because of the tossing actions. According to the National Institute of Standards and Technology, handling software bugs requires over $59.5 billion per year (NIST, May 2002). Additionally, in Aka company, the payment of a senior bug fixer is $129,328 per year (Glassdoor, n.d.). Because of these big numbers, triaging systems must be optimized.

Because many new approaches showed up recently, each one has different purposes and different algorithms. This paper presents a comparative study for a number of research papers exported from IEEE, Springer, ACM, IJSDR and ResearchGate, in a period from 2006 to 2022. The main contribution is summarized in a four-fold:

- Summarizing the state of art into a categorization schema; showing the model, algorithms, datasets used for each paper and results concluded by each of them.
- Providing a comparative analysis among previously published research papers.
- Introducing an experimental comparison for the performance of five classifiers in developer prediction.
- Providing some open discussion, which leads to find a gap to work on, in the future.

This paper organizes the work as followed. Section 2 provides a background and an explanation for some terminologies, that are necessary to understand the research area. Section 3 introduces a brief summary for the state of art. Section 4 discusses the previously proposed approaches in a criticizing manner. Section 5 shows experimental results of some classifiers used in bug triage problem. Section 6 introduces some ideas that could be explored in the future and conclusion.

2. BACKGROUND

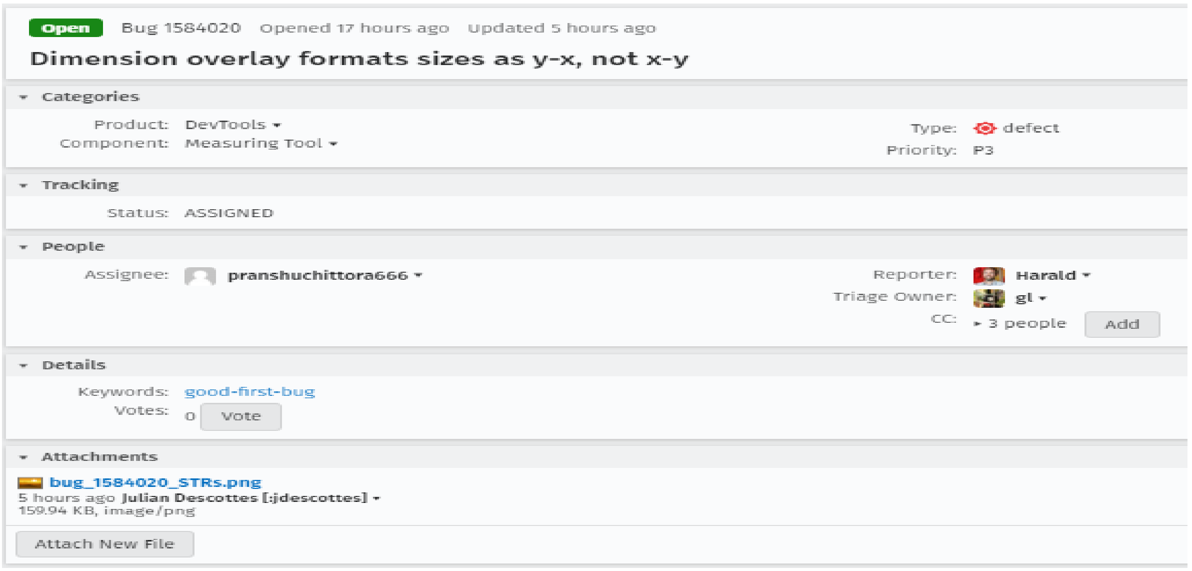
This section introduces the main terminologies and concepts used in bug triage in addition to the main algorithms adapted by previous triaging approaches.

Bug Report

A term bug can be defined as an unexpected system response, as a result from a code logical or syntax mistake. When the tester finds a bug in a software under test, he uses a bug report document to raise. Bug report is a standard form consists of a group of fields to define the raised issue, such as:

- <u>Bug ID:</u> unique number to identify the bug.
- <u>Bug Description:</u> a short paragraph describing the issue and the its module.
- <u>Open date</u>: the date when the assigned developer start working on the bug.

- Closing date: the date in which the bug is solved or closed as invalid bug.
- Bug Severity: it describes the level of impact for the bug. Bug severity is six levels; blocker, critical, major, minor, normal and traditional. Blocker is the highest one and traditional is the lowest.
- Bug Priority: it shows the urgency level of fixing the bug. It ranges between high, medium and low.
- Reporter: it shows the name of the person who raised the bug.
- Status: it shows the stage of the bug report.


Open Bug 1584020 Opened 17 hours ago Updated 5 hours ago

Dimension overlay formats sizes as y-x, not x-y

Categories

Product:	DevTools	Type:	defect
Component:	Measuring Tool	Priority:	P3

Tracking

Status: ASSIGNED

People

Assignee:	pranshuchittora666	Reporter:	Harald
		Triage Owner:	gl
		CC:	3 people

Details

Keywords: good-first-bug
Votes: 0

Attachments

bug_1584020_STRs.png
5 hours ago Julian Descottes [:jdescottes]
159.94 KB, image/png

Figure 1: Bug report form Bugzilla bug repository

After submitting a bug report, to the ITS, it goes through a sequence of sequential stages and takes a different state to each report (Bhattacharya & Neamtiu, Sept, 2010). After submitting a bug report, it is considered as NEW and it will change to ASSIGNED, if the triager assigned a developer to it, and if the developer fixed it successfully, it will be CLOSED. DUBLICATE means that there is another bug report describes the same issue. If there is no solution for the bug report, it is marked as INVALID. After solving the bug, the manger reviews it, if they decide that it is working appropriately, it will be marked as REVIEWD. After that, the manger takes the decision to close the bug and it will be CLOSED. If the programmer found that there is still a problem, the bug will be REOPEN for further modifications. Figure 1 shows the sequence of states that the bug takes starting from ASSIGNED till reaching CLOSED states at the end.

Figure 2 shows an example of a bug report from Bugzilla website (Bugzilla, 2014). Its status is ASSIGNED. The bug ID is 1584020. A description of the bug report is given by “Dimension Overly formats sizes as y-x, not x-y”. This bug report is reported by Haraid and the person who is assigned to work on this bug report has a username “Pranshuchittora666”. The reporter attached a file related to the problem to help the developer to fix this bug.

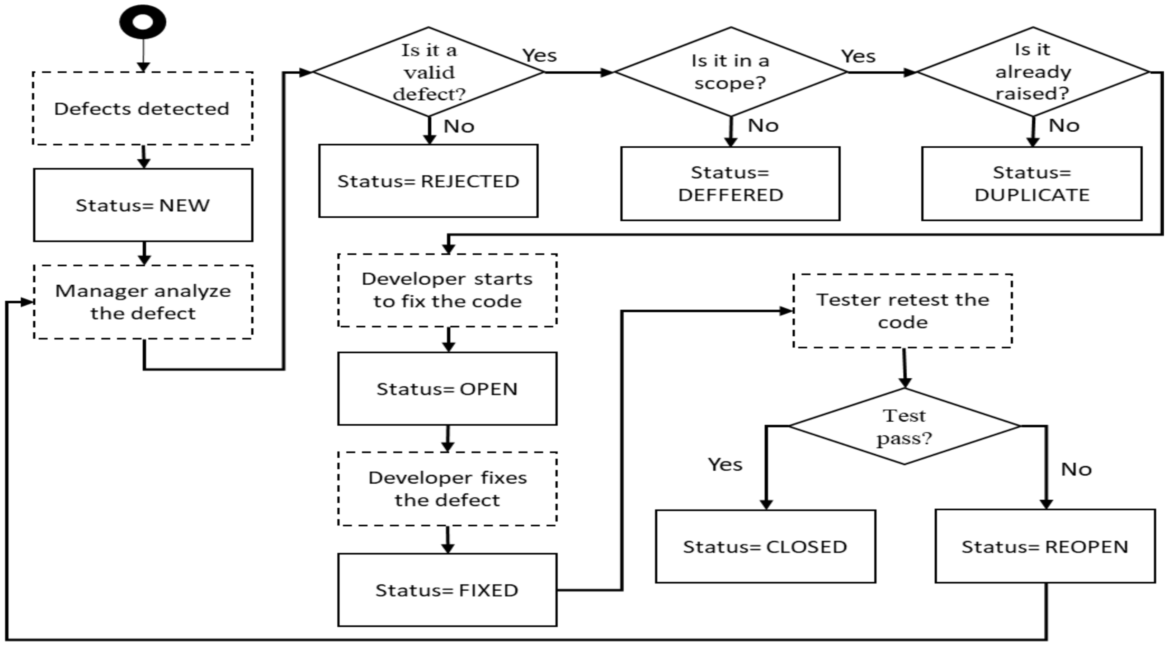


Figure 2: Bug report lifecycle (Bhattacharya & Neamtiu, Sept, 2010)

Bug Repository

It is a management system, where the developers can submit the raised issues on, using the template of bug report. Then, the submitted bugs are distributed on the available fixers by the trigger. The bug reporter and fixer can communicate on the same platform using the comment option on the bug report, that they work on. Mozilla (Mozilla bug tracking system, 2016) , GCC (Gcc bug tracking system, 2016), Eclipse (Eclipse bug tracking system, 2016), NetBeans (Netbeans bug tracking system, 2016) and OpenOffice (Openoffice bug tracking system, 2016) are the most common bug repositories.

2.1 Approaches and algorithms

Feature selection techniques

To apply machine learning algorithms, there should be a vector of features to train the model. Feature selection chooses the key subset of features that best represent the input. It is a critical step, which has high effect on the prediction accuracy. Three common methods were used in the context of bug triage (Z. Zheng, 2004):

Information gain (IG): this method evaluates the gain of a feature with respect to the category prediction. So, the target variable is used to differentiate between the feature (Z. Zheng, 2004).

Chi-square (CHI): this method measures the dependency between feature f in the feature vector and the class (category ci). It builds a probability distribution for the independency between the feature and the class given one degree of freedom. In general, machine learning models tends to use highly depended features (Z. Zheng, 2004).

Correlation Coefficient (CC): it is called Pearson's Correlation Coefficient. It is used to calculate how a feature is associated with the target. It is mainly used in continuous variables. It ranges between -1 and 1 (Z. Zheng, 2004).

Variance Threshold: this method ignores all features with zero variance and specific threshold. It is used to decide whether to consider the other features or not. The higher the feature variance, the more likely to contain more information (Isabelle Guyon, 2003)

Natural Language Processing (NLP)

Most of ML algorithms are not able to handle unstructured data, thus, NLP techniques are applied mostly to the summary and description sections of the bug report to extract the essential terms. However, these kinds of fields are best describing the bug report. Text mining works on representing this sort of information in n-dimensional space to be a structured data. This step composes three main steps:

- Tokenization: usually, it is the first step in handling the textual data. It is responsible for turning the natural language (unstructured data) into numerical structured information that can be represented as a vector.
- Stop words removal: it cleans the data from the noisy words, the irrelevant term that do not provide related information to the whole context, such as conjunctions, adjectives and pronouns.
- Stemming: it is the process of deducting a part of word and turning it to the stem version. For example, “profiling” becomes “profile”.

Topic modeling

It is used an unsupervised natural language processing technique. The input is the corpus (group of documents) and the number of topics. The output is a vector of probabilities, for each document, representing the relation of a document to each extracted topic. Many algorithms are used to apply this technique, such as Latent Dirichlet allocation (LDA) (Adam Thornton, 2020), which is the most common one, Latent Semantic Analysis and Probabilistic Latent Semantic Analysis.

Information retrieval

IR works based on a comparison between a single instant from the testing dataset and all the training dataset. Thus, it considers a new bug report as a query and using the IR algorithms try to assign it to the best developer. The documents (historical bug reports) and the newly submitted ones are represented in a similar manner and the assignment decision is taken over a matching process between them.

Document similarity methods

To measure similarity between two textual entities, they have to be represented in a similar way. This section introduces some document similarity methods.

Cosine similarity (Wael, 2013) uses the cosine of angle to measures the similarity between two textual documents.

Euclidean Distance (Puoya Tabaghi, 2020) represents the similarity between two coordinate points as a line. The shorter the line is, the similar are the two points.

BM25 (Stephen Robertson, 2009) represents the Best Match. It is used to rank the documents according to the relevance between documents.

BM25F (Tian, Lo, & Sun, 2012) is an extension for BM25. In BM25F, each document is represented as a group of fields, such as headline and main text. Each filed is represented by a different weight to imply its degree of importance in calculating the similarity.

Machin Learning (ML)

Machin learning approach uses artificial intelligence to enable software systems to learn through a training phase and implicitly apply in the testing phase without actual programming. ML algorithms are categorized into two types; supervised and unsupervised (Flach, 2012). The supervised algorithms are used to predict the model based on previous experience from a historical data. So, the input is labeled old data and the output is a forecasting for the label of a new data. This process is called classification, however, if the value of the label is continuous, this predictive process is called regression.

Bug triage problem could be handled as a classification problem; using ML algorithm. The datasets are the bug reports extracted from the bug repositories and the classes are the existing developers. The model is trained using some historical data; closed bug report with the assigned developers (labels). In the testing phase, the model is tested by new and unlabeled dataset, which called the testing dataset.

Each one in the ML algorithms works over a dataset consists of many samples or records S_i, where $i = \{1 \dots n\}$. Each record consists of a number of attributes, the problem features $\{S_{i1}, S_{i2}, S_{i3}, \dots S_{id}\}$, where $d = \{1 \dots m\}$, in addition to one attribute called, class or label $S_{i(m+1)}$. Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree are the most commonly used machine learning algorithms.

Support Vector Machine (SVM) creates an n-dimensional space, in which each feature vector is represented as a point. It separates the training instances (points) according to the associated class label. SVM aims to maximize the separation of the instance in a form of a hyperplane, which is the output model. SVM labels the testing data based on the subspace to which its features belong (Zhou, Zhang, & Lo, 2012).

k-Nearest Neighbors (KNN) uses the similarity measure to classify a new instance. Each instance in the dataset is represented as a point in an n-dimensional space. Euclidean distance is the most common technique to calculate the similarity between the new instance and k-nearest neighbors. Based on the similarity results, the new instance is labeled by the label of the nearest instance (Beyer, 1999).

Decision Tree models the training dataset in a form of tree, which consists of some internal nodes representing features, in addition to a leaf node showing the corresponding class. Each new instance in the testing dataset goes through a group of questions and conditions structured in a tree form. The output of Decision Tree algorithm is a model which is able to guide the new instance to the suitable label throughout these questions (Bahzad Taha Jijo, 2020).

Näive Bayes (NB) is a supervised algorithm, that uses Bayesian Theorem of conditional probability to determine the class of a new instance. All the features included are independent of each other, and each of them has equal contribution in predicting the class of the new instances, (Cristianini, 2000)

Graph theory

It is a mathematical representation for the instants of the problem. Each instant is represented as a node or vertices and the relation between them is represented by edges. The edge could be weighted showing the degree of relation between the two nodes.

Markov Chain Model: this model is represented as a set of states $S = \{s_1, s_2, \dots s_n\}$ and probabilities for transactions. The model moves from one state to another according to a specific probability, which is called transaction probability. Markov chain model becomes valid only if the current state (Markov property) is the only base for calculating the next transaction probability (Nguyen, Nguyen, & Lo, 2012).

Tossing graph means reassigning a bug report to another developer other than the existing one. Tossing graph is a representation for developers, who assigned to a bug report one after another. It starts with the first assigned developer and moves to one-another until reaching the actual fixer, the developer who fixed the bug. Each step is a tossing action. Tossing path consists of some tossing steps, indicating its tossing length (Yadav, KumarSingh, & Suri, 2019).

Evaluation metrics

In bug triage problem, as a prediction problem, there are four main matrices for measuring the efficiency of the proposed model; namely accuracy, precision, recall, and F-measure. Confusion matrix is the main factor to calculate the values of the four matrices (Fuli Zhang, 2019). It consists of four main values:

1. True Positive (TP): the instance is positive and the classifier predicted it as positive.
2. True Negative (TN): the instance is negative and the classifier predicted it as negative.
3. False Positive (FP): the instance is negative and the classifier predicted it as positive.
4. False Negative (FN): the instance is positive and the classifier predicted it as negative.

Where;

- True (T): all positive instance.
- False (F): all instance that are not positive.

Accuracy matrix measure how many instances are correctly classified as the real scenario among all testing instances. In bug triage problem, it calculates how many bug reports, in the testing dataset, are assigned to the correct developer according to the manual triage. Equation 1 shows the formula to calculate accuracy.

$$ACC = \frac{TP + TN}{P + N} \qquad (1)$$

Precision matrix calculates the how many correctly classified instances among all instances that are assigned to the class by the classifier. In bug triage problem, among all bug reports assigned to developer d by the classifier, how many assignments are correct, this is precision value, which is calculated by. Equation 2.

$$\text{Precision } = \frac{TP}{TP + FP} \qquad (2)$$

Recall matrix calculates how many instances are correctly classified to be of class c among all instances that are originally assigned to that class. In bug triage problem, among all bug reports that are originally assigned to developer d, how many bug reports assigned correctly by the classifier to developer d. Equation 3 shows the formula to calculate recall value.

$$Recall = \frac{TP}{TP + FN} \qquad (3)$$

F1-score is a harmonic value between the recall and precision. It ranges between 0, when either the recall or the precision is zero, and 1, which is the best scenario. Equation 4 shows how F1-score value is calculated.

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

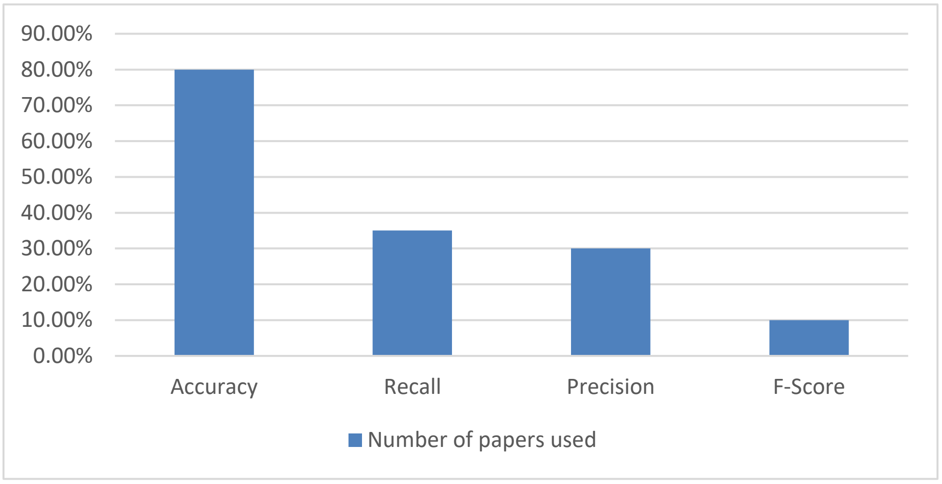


Figure 3: Statistics about the use of the evaluation matrices to evaluate the bug triage automation approach

Accuracy, recall, precision and F-measure matrices are the most common evaluating methods in bug triage problem. Figure 3 shows the percentage of using each of them across the bug triage research papers. Accuracy matrix is the most common one, it is used in more than 80% of the research papers, followed by the recall and precision. F-measure is the least. On the other hand, other researchers measure the effectiveness of the proposed model by estimating the total fixing time, calculating the tossing actions reduction, estimating the workload among developers. This step is completely dependent on the main goal of the research study.

3. BUG TRIAGE APPROACHES

Several researchers proposed different approaches to automate the triage process. Information retrieval approach is utilized by some of them. These approaches are either IR or ML techniques. Few researchers used fuzzy techniques to improve the IR approach. In addition, the graph theory is used to handle the triage process describing relations for different sorts of sets. Additionally, some researchers took the optimization approach to tackle the triage problem.

3.1 Information Retrieval Based

Many researchers leverage IR techniques to explore textual bug report attributes, for example, bug description and title to automate the triage process, where the new submitted bug report is considered a query.

Pahins et al (C´ıcero A. L. Pahins, 2019) proposed the T-REC method that combines the information retrieval approach and machine learning in order to reduce the triage time spent. VSM is applied and supported by the FastText approach to convert the textual features into embedding vectors, in addition to bag-of-words (BOW) model and Latent Semantic Analysis (LSA). Noisy-Or Classifier is used as a cross validation method to split the datasets into a number of folds. Experimental results show that the proposed algorithm increases the prediction by up to 30%. It is evaluated over an industrial dataset consists of 9.5M about mobile bug reports. He concluded that the T-REC can reach 50.9% to 89.7% accuracy.

Yin et al (Ying Yin, 2018) proposed a new hybrid bug triage method that depends on supervised leaning and ELM approach, which is a new tool that is used to train a single hidden layer based on feedforward neural network. The main focus of their solution is feature reduction. In order to not using all features, a large searching space is constructed to select only the features that clearly represent the actual content. The greedy algorithm is used for that purpose. In addition, VSM, which is a supervised learning algorithm, and TF-IDF are used for feature extraction. All in all, the ELM regression strategy is used for the assignment purpose. The bug reports dataset used to evaluate the proposed algorithm, 5200 from Bugzilla (Bugzilla, 2014), 4100 from Eclipse, 3824 from Netbeans and 2102 from GCC are exported. By comparing the proposed algorithm with other classifiers, such as SVM, C4.5, NaiveBayes, and KNN classifiers, results show that ELM concluded the best results, in terms of the triage accuracy. It reaches 71.2% using the Netbeans dataset.

Sahu et al (Sahu, Lilhore, & Agarwa, 2018) used hybrid selection method to reduce the dataset size. Feature selection and instance selection methods are used to reduce the number of features extracted from each bug report. Their proposed approach is based on K-Nearest Neighbor and Naive Base approaches (KNN+NB). By using Mozilla dataset (Mozilla bug tracking system, 2016), the accuracy percentage achieved is 85%.

Xin et al (Xia, Lo, Ding, M. Al-Kofahi, & N. Nguyen, 2017) came up with a new approach named TopicMinerMTM to handle the bug triage problem. It is an extension to the Latent Dirichlet Allocation (LDA) algorithm (Hofmann, 1999). It evaluates the relation between the between the bug report and the assigned developer. It uses the topic modeling distribution to choose a suitable developer for each bug report. GCC (Gcc bug tracking system, 2016), Eclipse (Eclipse bug tracking system, 2016), NetBeans (Netbeans bug tracking system, 2016), OpenOffice (Openoffice bug tracking system, 2016), Mozilla (Mozilla bug tracking system, 2016) are used for evaluation. The total number of bugs is 227,278 bug reports. The prediction accuracy ranges between 48% to 90%.

Sun et al (Sun, Lo, Khoo, & Jing , 2011) followed the approach of removing the duplicated bugs to reduce the dataset. Accordingly, the cost and time will be optimized. Thus, instead of two fixers are busy with the same bug report, using this automatic detector only one will be in charge. REP is a retrieval function, that extends BM25F in (Robertson, Taylor, & Zaragoza, 2004). This function takes into account the long bug reports which provide many components such as product, component, and version. To validate the proposed technique, Mozilla (Mozilla bug tracking system, 2016), Eclipse (Eclipse bug tracking system, 2016) and OpenOffice (Openoffice bug tracking system, 2016) are used. 209,058 reports are used from Eclipse. The results are concluded to be 37–71% in recall and 47% in mean average precision.

Nguyen et al (Nguyen, Nguyen, & Lo, 2012) used the mindset of Sun in reducing the bug triage cost by identifying the duplicated bugs. They proposed a model, with name duplicate bug report detection approach (DBTM), by combining the information retrieval (IR) based features and topic-based features approach as an extension for Sun's approach. Each bug report is represented as a textual document to describe the technical issues in the system. Textual similarity is used to detect the duplicate reports. In feature extraction, they combined both IR and topic modeling by applying ensemble averaging technique. This technique is defined as the mean of quantity. Gibbs sampling is a sampling technique known also with name Markov chain Monte Carlo (MCMC) algorithm. It uses multivariate probability distribution in order to achieve observations in a sequential manner. It is used to train DBTM on the dataset. Results show that DBTM reduced the triage cost by up to 20% compared with the state of art.

Matter et al (Matter, Kuhn, & Nierstrasz, May 2009) used vector space model to extract information from bug reports source code contributions for the purpose of modeling a developer's expertise. In the triage process, authors compared the vector of the upcoming new bug vector with the vectors of developers' expertise. They used bug reports of eight years from Eclipse (Eclipse bug tracking system, 2016) development bug data. They included 130,769 bug reports for the case study. Using top-10 recommendations list, they achieved triage accuracy up to 71.0%

3.2 Machine learning Based

Researchers used ML algorithms to handle the triage problem as a classification problem. The historical bugs (previously solved bugs) are used to train the model, then a prediction step is executed to assign a developer for each new bug report.
Kashiwa et al (Yutaro Kashiwa, 2020) is the first researcher focused on the developer workload. Based on an assumption, equal distribution of bug reports over developers can optimize the fixing time. The authors formulated the triage problem as a multiple knapsack problem for the purpose of finding the best assignment between the developers and the bug reports taking into account the work load of each developer. SVM classifier and Laten Dirichlet allocation are used for classification purposes. The algorithm is evaluated using bug reports from Mozilla Firefox (Mozilla bug tracking system, 2016), Eclipse (Eclipse bug tracking system, 2016) Platform and GNU compiler collection (GCC) (Gcc bug tracking system, 2016). Compared with the manual triage, the bug fixing time is saved by 35%–41%.

Lucas et al (Panjer, 2007) used data mining tools to predict the bug report fixing time. 0-R, 1-R, Decision Tree, Naïve Bayes and Logistic Regression are used. After executing the model over bug reports from Bugzilla (Bugzilla, 2014) and Eclipse (Eclipse bug tracking system, 2016), results show that Logistic Regression is the best algorithm in developer prediction, with accuracy 34.9%.

S. Mani et al (Mani, Sankaran, & Aralikatte, 2018) used deep learning model called deep bidirectional recurrent neural network (DBRNN-A) model to represent bug reports. This approach uses long word sequence to learn syntactic and semantic features. The bug report description and title are included. He used four different classifiers namely multinomial naive Bayes, cosine distance, support vector machines, and Softmax are compared. The bug reports used are from three different repositories Google Chromium (383,104), Mozilla Core (314,388), and Mozilla Firefox (162,307) (Mozilla bug tracking system, 2016). According to the experimental results, DBRNN-A along with Softmax is better than the bag of words model. Additionally, he reported the importance of the bug report description in improving the classification performance is reported.

Bhattacharya et al (Bhattacharya & Neamtiu, Sept, 2010) used the bag of words technique in addition to the term frequency-inverse document frequency (TF-IDF) techniques in order to extract the features. The information extracted from each bug report are title, description, keywords, products, components and last developer activity. TF-IDF is used to represent the importance of a word in each document. Naive Bayes, Bayesian network and tossing graph are used to build the classifier. The dataset is relatively large, they used 306,297 bug reports from Eclipse and 549,962 from Mozilla (Mozilla bug tracking system, 2016). The achieved prediction accuracy is 83.62%. By considering the report ID, the average prediction accuracy became 77.64%. However, the prediction accuracy dropped to 63% when ignoring the bug report ID.

Anvik et al (Anvik, Hiew, & Murphy, 2006) used ML approach to put some constrained to filter the data used from each bug report. In the training stage, the inactive developers who no longer working are ignored. The bug report fields used are the title and description. To apply feature extraction, authors used the normalized tf, a text mining algorithm to apply term frequency on two-fold normalization manner. Three different classifiers are applied, SVM, Naive Bayes, and C4.5. The achieved precision is up to 64%.

Gondaliya et al (Gondaliya, Peters, & Rueckert, 2018) used different methods of text mining; Lemmatization: to reduce the inflectional forms, part-of-speech tagger (POS tagger): a software used to read texts in any language, it assigns part of speech for each term, bigram: consists of a sequence of n adjacent elements (n=2) from string of tokens for the purpose of extracting the probability of token given another token, and finally, stop word removal: to filter the textual data from any unused terms such as “a” and “an”. Bug reports are classified through Linear Support Vector Machines (SVMs), multinomial naive Bayes, and Long Short-Term Memory (LSTM) networks and accordingly triage them. By assessing the proposed approach over small dataset (1215 bug report), the accuracy of the three classification techniques was 47.9 % → 57.2%. For the large dataset the performance ranged from 68.6% → 77.6%.

3.4 Fuzzy Based

Tamrawi et al (Tamrawi, Nguyen, Al-Kofahi, & Nguyen, 2011) proposed a new automated approach called Bugzie for bug triage. It mainly uses fuzzy set and caching the developers. A model is used to cash the developers in a form of queue according to their skills and experience. After that, the assignment phase is executed one by one starting from the first developer in the queue. Each developer is represented by a score explaining his previously fixed bug reports. The author assumed that the software system has many technical aspects, the fuzzy represents each term in them. Bugzie, their proposed approach combines the fuzzy sets and the score of each developer to predict the most suitable developer for each new bug report. The prediction accuracy ranged between 75% to 83%.

3.5 Graph Theory Based

Graph theory is used by some researchers to represent different kinds of relations between the bug report and developers.

Zaidi et a (Syed Farhan Alam Zaidi H. W.-G., 2022) used the graph representation approach to solve the bug triage approach. The description and summary fields from the bug report are used to build a heterogeneous graph. The proposed solution consists of two layers of graph convolution network (GCN); the first one represents word to word co-occurrence and the second one is about word to bug report. point-wise mutual information (PMI) is used to weight the edges between words in the first layer and TF-IDF is used to calculate the occurrence rate for the second layer. In addition, Softmax activation function is used in the prediction stage. Five different datasets are used from Bugzilla (Bugzilla, 2014) and Firefox. Among many distance measurement approaches used, such as PMI, cosine, Jaccard, and Euclidean techniques, PMI recorded the best performance. Comparing with the previous automating triage approaches, predicting top-1 accuracy gets higher by 3% to 6% and up to 5% to 8% in top-10 accuracy.

Zaidi et al (Syed Farhan Alam Zaidi F. M.-G., 2020) again focused on the word-embedding techniques to rank the developers. They proposed a deep-learning based technique that used the convolutional neural network (CNN). Three different word representation techniques are used namely; Word to Vector (Word2Vec), Global Vector (Glove), and Embeddings from Language Models (ELMO). The main solution consists of three layers; word representation, convolutional network and Softmax for prediction actions. In many cases, GloVe outperforms the CNN, only with high numbers of samples. By evaluating their model over three different datasets; 1465 bug reports from JDT, 4825 bug reports from platform and 13667 bug reports from Firefox, and using the 10-fold cross validation approach for splitting the datasets, results show that ELMO-CNN is the best, which reaches 87.23% accuracy.

Aleroud et al (Alazzam, Aleroud, Latifah, & Karabatis, 2020) introduced a graph-based approach representing the relations between the bug report terms. He classified the bug reports according to their priority. Accordingly, he believed that highly-skilled developers should be assigned to the bug report with high priority and for the low priority ones, the new developer, who are not experienced, should be allocated. RFSH technique, RF is the regular feature, S is the summary feature and H is a hybrid approach that depends on a cutoff percentage, is proposed to predict the bug priority. Rather than LDA, which is the main focus (Xia, Lo, Ding, M. Al-Kofahi, & N. Nguyen, 2017) and (Yutaro Kashiwa, 2020), Aleroud (Alazzam, Aleroud, Latifah, & Karabatis, 2020) stated that using LDA, which might use unrelated terms, can mislead the classification action. Additionally, LDA cannot capture all relations between the terms. Therefore, the proposed approach handles this by removing the outlier terms, that may be related to different terms. KNN, SVM, and decision tree (DT) techniques are used for classification purpose. The proposed approach is tested against 135 548 bug reports from Bugzilla (Bugzilla, 2014) bug repository. Results show that SVM worked the best in terms of accuracy and precision.

Jeong et al (Jeong, Zimmermann, & Kim, 2009) proposed the use of Markov chain-based model to reduce the bug tossing actions. It introduces developer network which used to explain the team structure that can be used to find the appropriate developer for each newly submitted bug report. 445,000 bug reports are used from Mozilla (Mozilla bug tracking system, 2016) and Eclipse (Eclipse bug tracking system, 2016). Experiments show that the tossing rate is reduced by 72% and the prediction accuracy is increased by 32%.

Yadav et al (Yadav, KumarSingh, & Suri, 2019) came up with a three-stage novel strategy called developer expertise score DEC. First, an offline process to assign a score for each developer based on versatility, priority and the average fixing time. Finding the capable developers is handled by applying the similarity measures such as namely feature-based, cosine-similarity and Jaccard. Second, ranking the developers according to their DES. Finally, Navies Bayes, Support Vector Machine and C4.5 classifiers are used for classification purpose and he compared their performance with the ML-based bug triage approaches. By evaluating the algorithm over five open-source datasets (Mozilla (Mozilla bug tracking system, 2016) ,Eclipse (Eclipse bug tracking system, 2016), Netbeans (Netbeans bug tracking system, 2016), Firefox) with 41,622 bug reports, results show that DES systems recorded 89.49% in mean accuracy, 89.53% in precision, 89.42% in recall and 89.49% in F-score. Additionally, the bug tossing length is reduced by 88.55%.

3.6 Bug Triage Optimization Based

Park et al (Jin-woo Park, 2011) also used LDA to categorize bug reports for the purpose of enhancing the prediction accuracy in their proposed solution, COSTRIAGE. To formulate a developers' profile Arun's is used to determine the optimal number of K and accordingly average fixing time is used to represent the developer experience with respect to the topic modeling results. To evaluate COSTRIAGE, bug reports from Linux, Apache, Mozilla and Eclipse bug repositories are extracted. Using 80% of the dataset for training the model and 20% for testing, results show that fixing time is reduced by more than 30% in Apache dataset, with 4.5% to 69.7% in prediction accuracy.

Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022) used the Hungarian algorithm and topic modeling technique to assign a developer to each bug report. The proposed approach consists of three main stages; namely, the labeling phase, the scoring phase and the assignment phase. To set a score for each developer, the data extracted and from each bug report and the skills of each developer are represented using the value of the average fixing time of the developer. Topic modeling is used to differentiate between the types of bug reports and also the skills required to fix each bug report; so, each bug report is labeled by a topic and each developer will be represented by a vector of numbers showing his score in each topic. Using these two phases, a bipartite graph is built, which is the input to the Hungarian model. The algorithm is validated using 26,317 bug reports from Bugzilla (Bugzilla, 2014) and 37,154 bug reports from RedHat. Results show that the fixing time is reduced by 17% in Bugzilla and 47% in RedHat. In addition, the developer work load is equally distributed across the developers set.

Madonna et al (Mayez, Nagaty, & Hamdy, Arxiv, 2022) represented the developer experience using bug severity, component and the median fixing time. The goal is to optimize the total fixing time and normalize the work load among developers. Authors used data cleaning, preprocessing and embedding similar to (Mayez, Hamdy, & Nagaty, Arxiv, 2022). However, in this research, authors used a recommendation system using matrix factorization. So, for each bug report, there is a set of candidates (developers), who might be eligible for fixing the bug. Deciding the most appropriate developer is taken over Gale-Shapely algorithm. To evaluate the proposed model, bug reports from Linux, Apache and Eclipse repositories are exported. By comparing the optimization results with COSTRIAGE model (Jin-woo Park, 2011), results show that Madonna et al (Mayez, Nagaty, & Hamdy, Arxiv, 2022) model is far better than COSTRIAGE approach, in terms of fixing time optimization.

Table 1, 2, 3, 4 and 5 summarize a number of research papers that focused of handling the bug triage process in the four above mentioned categorization, starting from 2006 to 2022.

Table 1. Summary for the bug triage automation techniques using information retrieval approach

Author	**Bug repository**	**Techniques**	**Results**
Pahins (C´ıcero A. L. Pahins, 2019)	9.5M industrial mobile bug reports	T-REC, bag-of-words (BOW), FastText approach, Latent Semantic Analysis (LSA)	Acc: 50.9% to 89.7%
Yin (Ying Yin, 2018)	Bugzilla: 5200, Eclips: 4100, Netbeans: 3824 and GCC: 2102	TF-IDF, VSM, ELM	Acc: 71.2%
Sahu (Sahu, Lilhore, & Agarwa, 2018)	From Mozilla dataset	Combination of K-Nearest Neighbor and Naive Base (KNN+NB).	Acc: 81.3%, Recall: 94.9%, Precision: 85.3%, F1- measure: 89.6%.
Xin (Xia, Lo, Ding, M. Al-Kofahi, & N. Nguyen, 2017)	227,278 From GCC, NetBeans, Eclipse, OpenOffice, Mozilla	Topic modeling: Latent Dirichlet Allocation (LDA)	Acc: 48% - 90%.
Nguyen (Nguyen, Nguyen, & Lo, 2012)	OpenOffice: 31,138, Mozilla: 75,653 and Eclipse: 45,234	Markov chain Monte Carlo (MCMC), Textual similarity, topic modeling and Gibbs sampling.	20% triage cost optimization. Acc: 82% in 10-k
Sun (Sun, Lo, Khoo, & Jing , 2011)	209,058 reports from Eclipse	Gradient Descent, linear kernel SVM to optimize REP	Recall:37–71%,, Precision: 47%.
Matter (Matter, Kuhn, & Nierstrasz, May 2009)	Eclipse:130,769	Bags of words, cosine similarity	Recall: 71.0%. Precision: 33.6%

Table 2. Summary for the bug triage automation techniques using machine learning approach.

Author	**Bug repository**	**Techniques**	**Results**
Kashiwa (Yutaro Kashiwa, 2020)	Mozilla Firefox, Eclipse Platform and GNU compiler collection (GCC).	Knapsack approach, SVM classifier and Laten Dirichlet allocation	35%–41% reduction for the fixing time
S.Mani (Mani, Sankaran, & Aralikatte, 2018)	Google Chromium: 383,104, Mozilla Core: 314,388, and Mozilla Firefox: 162,307	Recurrent neural network, naive Bayes, cosine distance, support vector machines, and Softmax and bag of words	Softmax is better than the bag of words model. Acc: 46.6%
Bhattacharya (Bhattacharya & Neamtiu, Sept, 2010)	Eclipse: 306,297 and Mozilla: 549,962	Frequency inverse document frequency (TF-IDF), Naive Bayes and Bayesian network	Acc: 77.64% -83.62% Recall: 50%
Lucas (Panjer, 2007)	Bugzilla and Eclipse	0-R, 1-R, Decision Tree, Naïve Bayes and Logistic Regression	Accuracy 34.9%.
Anvik (Anvik, Hiew, & Murphy, 2006)	Training sets (Eclipse :8655, Firefox: 9752) and testing sets (Eclipse :170, Firefox: 22)	SVM, Naive Bayes and C4.5.	Precision: 51%. Recall: 24% Acc:28%
Gondaliya (Gondaliya, Peters, & Rueckert, 2018)	Large dataset: 7346	Part-of-speech tagger, bigram, stop word removal, Linear Support Vector Machines (SVMs), multinomial naive Bayes, and Long Short-Term Memory (LSTM) networks	68.6%- 77.6%
	small dataset: 1215		47.9 - 57.2%,

Table 3. Summary for the bug triage automation techniques using graph theory approach

Author	**Bug repository**	**Techniques**	**Results**
Zaidi (Syed Farhan Alam Zaidi H. W.-G., 2022)	Bugzilla and Firefox.	Graph convolution network (GCN), point-wise mutual information (PMI), TF-IDF and Softmax	Top-1 accuracy increased by 3 to 6% and 5 to 8% in top-10.
Zaidi (Syed Farhan Alam Zaidi F. M.-G., 2020)	JDT: 1465, platform:4825 and Firefox: 13667	Deep-learning, convolutional neural network (CNN), Word to Vector (Word2Vec), Global Vector (Glove), and Embeddings from Language Models (ELMO), Softmax and 10-fold	Acc: 87.23%.
Aleroud (Alazzam, Aleroud, Latifah, & Karabatis, 2020)	Bugzilla: 135,548	RFSH technique, LDA, KNN, SVM, and decision tree (DT)	SVM is the best classifier Acc: 94.1% Precision: 92.5% Recall: 94.1% F-measure: 93.3%
Jeong (Jeong, Zimmermann, & Kim, 2009)	445, 000 from Mozilla and Eclipse	Markov chain-based model	Tossing actions are reduced by 72% and acc increased by 32%
Yadav (Yadav, KumarSingh, & Suri, 2019)	41,622 from Mozilla, Eclipse, NetBeans, Firefox (Mozilla bug tracking system, 2016)	feature-based, cosine-similarity, Jaccard, Navies Bayes, Support Vector Machine and C4.5	Acc: 89.49%, precision: 89.53%, recall: 89.42%, and F-score: 89.49%

Table 4. Summary for the bug triage automation techniques using fuzzy approach

Author	**Bug repository**	**Techniques**	**Results**
Tamrawi (Tamrawi, Nguyen, Al-Kofahi, & Nguyen, 2011)	Firefox:177,028 Eclipse: 110,231 NetBeans: 42,797 FreeDesktop: 39,771	Bugzie, cashing technique	Acc: 75-83%.

Table 5. Summary for the bug triage automation techniques using Optimization approach.

Author	**Bug repository**	**Techniques**	**Results (Fix time optimization)**
Park et al (Jin-woo Park, 2011)	Linux	Topic modeling (LDA), content-boosted collaborative filtering (CBCF), combining an existing CBR with a collaborative filtering recommender (CF)	28.9 %,
	Eclipse		10.6 %
	Apache		59.66%
Madonna et al (Mayez, Nagaty, & Hamdy, Arxiv, 2022)	Linux	Topic modeling (LDA), developer score, matrix factorization, Gale-Shapely and Time-split approach,	80.67%
	Eclipse		23.61%
	Apache		60.22%
Madonna (Mayez, Hamdy, & Nagaty, Arxiv, 2022)	RedHat: 37,154	Topic modeling (LDA), developer score, Hungarian Algorithm, Time-split approach, Bipartite graph.	47%
	Bugzilla: 26,317		17%

4 ANALYSIS AND DISCUSSION

In previous studies, authors handled the problem as a classification problem considering that each developer is a class. The high accuracy recorded from them such as (Sahu, Lilhore, & Agarwa, 2018), (C´ıcero A. L. Pahins, 2019), (Yadav, KumarSingh, & Suri, 2019) and (Bhattacharya & Neamtiu, Sept, 2010) is because the used classifiers work good in the limited datasets with limited number of classes, but when the number of classes get bigger, the accuracy will negatively affected. Others used information retrieval approaches, they focused on analysis the training dataset and match between the new bugs and the historical ones to determine which developer is more appropriate. Considering the manual triage as the optimal one, Xin et al (Xia, Lo, Ding, M. Al-Kofahi, & N. Nguyen, 2017) recorded the highest accuracy percentage, almost 90%. Few researchers followed the approach of minimizing the total fixing time, Kashiwa et al (Yutaro Kashiwa, 2020) used Knapsack approach, followed by Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022), who handled it as an assignment problem using optimization approach. A special factor in those two papers, not only the optimizing the fixing time but also, the load is normalized among developers, the number of bug reports assigned to each developer are almost normalized.

Representing the developer skills in a numerical form is a perfect way to choose or recommend the appropriate developer easily. Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022), Yadav et al (Yadav, KumarSingh, & Suri, 2019) and Tamrawi et al (Tamrawi, Nguyen, Al-Kofahi, & Nguyen, 2011) used this technique with different factors. Yadav et al (Yadav, KumarSingh, & Suri, 2019) used severity, component and average fixing time to represent the experience of each developer. Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022) inspired by their idea, she focused on the average time spent and added the topic modeling results as a factor. Thus, the experience of each developer will be represented as a vector of numbers. Each number represents the developer average fixing time in one topic. However, the goal of Yadav's et al (Yadav, KumarSingh, & Suri, 2019) work is classification, that ended up with 89.5% accuracy. Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022) focused on optimizing the total fixing time; 47% reduction in fixing time.

Graph theory is used in few papers representing different relations. Zaidi et al (Syed Farhan Alam Zaidi H. W.-G., 2022) used the graph to represent how much each bug report is related to each of the terms, that resulted from the TF-IDF. In addition, Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022) used the complete bipartite graph to represent the relations between the bug reports and developers. The relations represent the skills of developers in the area of the corresponding bug report.

Many researchers used topic modeling approach to differentiate between bug reports. Xin et al (Xia, Lo, Ding, M. Al-Kofahi, & N. Nguyen, 2017) used LDA to match between the topics and the developers experience. Nguyen also used LDA. Both of them recorded high accuracy, 48%-90% for Xin et al (Xia, Lo, Ding, M. Al-Kofahi, & N. Nguyen, 2017) and 20% cost reduction for Nguyen. Additionally, Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022) used LDA as a step to set a scoring phase, each developer has a score per topic, where the fixing time is reduced by 47%. On the other hand, Aleroud et al (Alazzam, Aleroud, Latifah, & Karabatis, 2020) stated that using topic modeling can add some noise in calculating the accuracy, some unrelated terms can be included. So, a layer of filter is added to exclude the outliers (unrelated terms).

Zaidi et al (Syed Farhan Alam Zaidi F. M.-G., 2020) recorded high accuracy; however, the technique overall is not cost-effective. The dataset is very large, which requires large memory and time. They used the graph to represent word-to-word relations, which makes the graph significantly large to be loaded and reused. Therefore, the training phase using graph convolution network (GCN) takes long time to be done, which contradict one of the main goals, automating the bug triage to reduce the total fixing time.

Most of the pre-mentioned papers use the 10-fold technique to split the dataset into training and testing, such as Zaidi et al (Syed Farhan Alam Zaidi F. M.-G., 2020) and Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022). But in this sort of problems, the chorological order is important. Because the bug reports should be assigned according to their date of submission, what comes first should be assigned first. Therefore, Madonna et al (Mayez, Hamdy, & Nagaty, Arxiv, 2022) mentioned that the exported datasets are ordered chronologically as a pre-step before applying the 10-fold approach, which called time-split approach. Another point in cleaning the datasets, many researchers exclude the inactive developers, who fixed few bug reports, for example 10 bugs, in their history. However, Zaidi et al (Syed Farhan Alam Zaidi F. M.-G., 2020) stated that, this step can negatively affect the final results.

5 EMPIRICAL STUDY

Most of the previous work focus on the prediction of the appropriate developers, thus different classifiers are applied. For results validation, this section introduces classification results of five previously applied classifiers, using a single dataset. Bug reports from Eclipse (Eclipse bug tracking system, 2016) are exported, in JSON format.
XSLT parser is used to parse bug reports. As a cleaning step, bug reports with missing information, such as empty assignee cell, are discarded. The features used for classification are bug assignee, cug component and bug description (summary). The corpus, description section in each bug report, is cleaned using stop word removal. For preprocessing purpose, all bug reports are stemmed using Snowball stemming. Considering the bug description, assignee and bug component, five common classifiers are implemented. 160,000 bug report are exported. After cleaning, only 66723 bug report meet the requirements. The dataset is split in 80% for training and 20% for testing. In each experiment, classifier should predict the optimal developer for each bug report in the testing dataset. Prediction accuracy is calculated against the developers assigned using the manual triage.

Table 6. prediction based comparison among five classifiers used in bug triage problem using Eclipse .

	LinearSVM (Lubor Ladicky, 2011)	**Logistic regression** (Zou, Hu, Tian, & Shen, 2019)	**BernoulliNB** (Gurinder Singh, 2019)	**MultinomialNB** (Muhammad Abbas, 2019)	**Decision Tree** (Philip H. Swain, 1977)
Prediction accuracy	**81%**	32%	66%	78%	33%

Table 6 shows the prediction performance of some classifiers using one dataset. Linear SVM, Logistic regression, BernoulliNB , MultinomialNB and Decision Tree are implemented and tested, in terms of their prediction performance. Results shows that Linear SVM has the maximum prediction accuracy compared with other classifiers, followed by MultinomialNB and BernoulliNB. However, Logistic regression and Decision Tree have very low prediction accuracy. What makes SVM classifier perform better than some other classifiers, such as Naive Bayes and Decision Tree, is that is uses lines to separate the between the data points. In the prediction step, it chooses the best line to predict the class of each data points.

6 FUTURE RESEARCH DIRECTION AND CONCLUSION

This section focuses on new research directions that could be used in the future as modifications and extensions for the existing triaging approach.

- The bug description field is the main focus in most of previous research works. Many authors applied natural language processing techniques and text mining to extract the main concepts from each bug report. This manner worked well in most of the previous studies that ended up by high prediction accuracy and fixing time optimization. After deep investigation, we found that comment section in each bug report has detailed information about not only the bug details but also the developer interests. It mainly has much details about a bug report and developer interests. Thus, bug comments will be research area in our future work.

- Setting a score for each developer is an open research area. Previous research papers included limited features from bug report, such as bug severity, component, submission time, closed time and priority. However, more features from bug report could be used to formulate this step. Such as comments and attachments.
- Because developers set is changing over time, some developers leave the project and others join, bug triage depends on the types of the newly submitted bug reports and the skills of existing developers. Thus, it could be handled as a stochastic problem. Stochastic model depends on probability forecasts and random variables to predict outcomes. It is a new area that could have a great impact on the whole software maintenance process.

In conclusion, bug triage is a time-consuming process, when done manually. Different techniques are used to cope with this dilemma. Some of them used classification approach, others used information retrieval approach to automate it. Few researchers focused on the fixing time reduction and developer load normalization. In experiments done by previous work, prediction results are varying but most of them are perfect, however each approach has some limitations. Some papers focus on developer prediction and ignores the developer work load, others consider the total fixing time optimization and developer work load but they totally ignore the classification accuracy. This paper summarizes the previous work, algorithms, datasets, bug repositories and the results, followed by some analysis. In addition, for deeper investigation, an empirical study is conducted to test the impact of different common classifiers. Experimental results show that linear SVM has the best classification performance, compared with, Linear SVM, Logistic regression, BernoulliNB, MultinomialNB and Decision Tree

REFERENCES

Adam Thornton, B. M. (2020). Latent Dirichlet Allocation (LDA) for Anomaly Detection in Ground Vehicle Network Traffic. *in Proceedings of IEEE 39th Digital Avionics Systems Conference (DASC).*

Alazzam, I., Aleroud, A., Latifah, Z. A., & Karabatis, G. (2020). Automatic Bug Triage in Software Systems Using Graph Neighborhood Relations for Feature Augmentation. *IEEE Transactions On Computational Social Systems, 7*(5), 1288 - 1303.

Alenezi, M., Banitaan, S., & Zarour, M. (2018). Using Categorical Features in Mining Bug Tracking Systems to Assign Bug Reports. *International Journal of Software Engineering & Applications (IJSEA), 9*(2), 29-39.

Anjali, S. K. (January 2015). Bug Triaging: Profile Oriented Developer Recommendation. *International Journal of Innovative Research in Advanced Engineering (IJIRAE), 2*(1), 36-42.

Anvik, J., Hiew, L., & Murphy, G. (2006). Who should fix this bug? *in Proceedings of ACM International Conference on Software Engineering*, (pp. 361–370). Shanghai, China.

Bahzad Taha Jijo, A. M. (2020). Classification Based on Decision Tree Algorithm for Machine Learning. *Journal of Applied Science and Technology Trends*, 20-28.

Beyer, K. G. (1999). When Is "Nearest Neighbor" Meaningful? *International Conference of Database Theory* (pp. 217-235). Berlin: Springer.

Bhattacharya, P., & Neamtiu, I. (Sept, 2010). Fine-grained Incremental Learning and Multi-feature Tossing Graphs to Improve Bug Triaging. *in Proceedings of IEEE International Conference on Software Maintenance.* Timisoara, Romania.

Bugzilla. (2014). Retrieved September 15, 2019, from http://bugzilla.org/

C´ıcero A. L. Pahins, F. D. (2019). T-REC: Towards Accurate Bug Triage for Technical Group. *18th IEEE International Conference on Machine Learning and Applications (ICMLA).* Manaus, Brazil.

Cristianini, N. &.-T. (2000). *An introduction to support vector machines and other kernel-based learning methods.* Cambridge university press.

Eclipse bug tracking system. (2016). Retrieved September 15, 2019, from https://bugs.eclipse.org/bugs/

Flach, P. (2012). *The art and science of algorithms that make sense of data.* Cambridge university press.

Fuli Zhang, X. B. (2019). Author Impact: Evaluations, Predictions and Challenges. *2019 IEEE. Translations and content mining are permitted for academic research only*, 38657- 38669.

Gcc bug tracking system. (2016). Retrieved September 15, 2019, from http://gcc.gnu.org/bugzilla/

Glassdoor. (n.d.). Retrieved October 9, 2019, from https://www.glassdoor.com/Salaries/senior-bug-fixer-aka-software-engineer-salary-SRCH_KO0,38.htm

Gondaliya, K., Peters, J., & Rueckert, E. (2018). Learning to Categorize Bug Reports with LSTM Networks. *in Proceedings of The10th International Conference on Advances in System Testing and Validation Lifecycle.* Germany.

Gurinder Singh, B. K. (2019). Comparison between Multinomial and Bernoulli Naïve Bayes for Text Classification. *International Conference on Automation, Computational and Technology Management (ICACTM).*

Hofmann, T. (1999). Probabilistic latent semantic indexing. *in Proceeding of Research and development in information retrieval.* Berkeley.

Hongliang Liang, L. S. (2019). Deep Learning With Customized Abstract Syntax Tree for Bug Localization. *IEEE Access*, 116309-116320.

Huo, X., Thung, F., Li, M., Lo, D., & Shi, S.-T. (2019). DeepTransfer Bug Localization. *IEEE Transactions on Software Engineering*, 1368-1380.

Isabelle Guyon, A. E. (2003). An Introduction of Variable and Feature Selection. *Journal of Machine Learning Research* , 1157 - 1182.

Jeong, G., Zimmermann, T., & Kim, S. (2009). Improving Bug Triage with Bug Tossing Graphs. *in Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering.* Korea.

Jin-woo Park, M.-W. L.-w. (2011). CosTriage: A Cost-Aware Triage Algorithm for Bug Reporting Systems. *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence.* San Francisco, California.

Jira Software. (n.d.). Retrieved September 15, 2019, from https://www.atlassian.com/software/jira/bug-tracking

Lubor Ladicky, P. H. (2011). Linear Support Vector Machines. *Proceedings of the 28th International Conference on Machine Learning.* Bellevue, Washington,USA.

Mani, S., Sankaran, A., & Aralikatte, R. (2018). DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging. *in Proceedings of the ACM India Joint International Conference.* India.

Matter, D., Kuhn, A., & Nierstrasz, O. (May 2009). Assigning Bug Reports using a Vocabulary-based Expertise Model of Developers. *in Proceedings of 6th IEEE Working Con. on Mining Software Repositories (MSR' 09)*, (pp. 131-140). Vancouver, BC, Canada,.

Mayez, M., Hamdy, A., & Nagaty, K. (2022, July 14). *Arxiv*. Retrieved from Arxiv: https://arxiv.org/abs/2202.01713

Mayez, M., Nagaty, K., & Hamdy, A. (2022, July 14). *Arxiv*. Retrieved from arxiv: https://arxiv.org/abs/2207.07149

Mozilla bug tracking system. (2016). Retrieved September 15, 2019, from https://bugzilla.mozilla.org/.

Muhammad Abbas, K. A. (2019). Multinomial Naive Bayes Classification Model for Sentiment Analysis. *IJCSNS International Journal of Computer Science and Network Security,*, 62-67.

Netbeans bug tracking system. (2016). Retrieved September 15, 2019, from https://netbeans.org/bugzilla/

Nguyen, A., Nguyen, T. T., & Lo, D. (2012). Duplicate bug report detection with a combination of information retrieval and topic modeling. *in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering,* (pp. 70-79). Essen, Germany.

NIST. (May 2002). *The economic impacts of inadequate infrastructure for software Testing.* USA: U.S Department of Commerce.

Openoffice bug tracking system. (2016). Retrieved September 15, 2019, from https://bz.apache.org/ooo/

Panjer, L. D. (2007). Predicting Eclipse Bug Lifetimes. *in Proceedings of Mining Software Repositories.* Canada.

Philip H. Swain, H. H. (1977). The Decision Tree Classifier: Design and Potential. *IEEE Transactions on Geoscience Electronics*, 142 - 147.

Puoya Tabaghi, I. D. (2020). Kinetic Euclidean Distance Matrices. *IEEE Transactions on Signal Processing*, 452-465.

Robertson, S., Taylor, M., & Zaragoza, H. (2004). Simple BM25 Extension to Multiple Weighted Fields. *in Proceedings of the thirteenth ACM international conference on Information and knowledge management.*

Sahu, K., Lilhore, U. K., & Agarwa, N. (2018). An Improved Data Reduction Technique Based On KNN & NB with Hybrid Selection Method for Effective Software Bugs Triage. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 3*(5), 2456-3307.

Stephen Robertson, H. Z. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.*, 333-389.

Sun, C., Lo, D., Khoo, S.-C., & Jing . (2011). Towards more accurate retrieval of duplicate bug reports. *in Proceedings of 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, (pp. 253–262). Lawrence, KS, USA.

Syed Farhan Alam Zaidi, F. M.-G. (2020). Applying Convolutional Neural Networks With Different Word Representation Techniques to Recommend Bug Fixers. *ACCESS, 8*, 213729- 213747.

Syed Farhan Alam Zaidi, H. W.-G. (2022). A Graph Convolution Network-Based Bug Triage System to Learn Heterogeneous Graph Representation of Bug Reports. *IEEE access, 10*, 20677-20689.

Tamrawi, A., Nguyen, T. T., Al-Kofahi, J., & Nguyen, T. (2011). Fuzzy Set and Cache-based Approach for Bug Triaging. *in Proceedings of 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13rd European Software Engineering* ,. USA.

Tian, Y., Lo, D., & Sun, C. (2012). Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. *19th Working Conference on Reverse Engineering.*

Wael, G. a. (2013). A Survey of Text Similarity Approaches. *nternational Journal of Computer Applications*, 13-18.

X. Yan, K. J. (2018). Bug Localization withSemantic and Structural Features using Convolutional Neural Networkand Cascade Forest. *22nd International Conference onEvaluation and Assessment in Software Engineering* . ACM.

Xia, X., Lo, D., Ding, Y., M. Al-Kofahi, J., & N. Nguyen, T. (2017). Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering, 43*(3), 272-297.

Yadav, A., KumarSingh, S., & Suri, J. (2019). Ranking of Software Developers based on Expertise Score for Bug Triaging. *Information and Software Technology, 112*, 1-17.

Ying Yin, X. D. (2018). Rapid and Efficient Bug Assignment Using ELM for IOT Software. *IEEE access.* China.

Yutaro Kashiwa, N. M. (2020). A Release-Aware Bug Triaging Method Considering Developers' Bug-Fixing Loads. *IEICE TRANS. INF. & SYST, 103*(2), 348-362.

Z. Zheng, X. W. (2004). Feature selection for text categorization on imbalanced data. *SIGKDD Explor*, 80–89.

Zhou, J., Zhang, H., & Lo, D. (2012). Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. *34th International Conference on Software Engineering (ICSE)*, (pp. 14-24).

Zou, X., Hu, Y., Tian, Z., & Shen, K. (2019). Logistic Regression Model Optimization and Case Analysis. *IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT).* Dalian, China.