Int J Syst Assur Eng Manag (June 2023) 14(3):989–1009
https://doi.org/10.1007/s13198-023-01902-7

ORIGINAL ARTICLE

NRPredictor: an ensemble learning and feature selection based approach for predicting the non-reproducible bugs

Kulbhushan Bansal[1] · Gopal Singh[2] · Sunesh Malik[3] · Harish Rohil[4]

Received: 10 February 2022 / Revised: 24 June 2022 / Accepted: 29 March 2023 / Published online: 8 May 2023

Abstract Software maintenance is essential and significant phase of software development life cycle. In software projects, issue tracking systems are used to collect, categorise, and track filed issues. The distinct bug reports are not being able to reproduced by software developers and hence, marked as non-reproducible. Non-reproducible problems are a major performance issue in bug repositories since they take up a lot of time and effort from developers. The goal of this paper is to create a prediction model for detecting non-reproducible bugs. Due to sheer unexpected nature of bug fixation, bug management is frequently a painful undertaking for software engineers. Non reproducible bugs add to the difficulty of this vexing indexing. This paper deals with the development of a early prediction model for identification of non-reproducible bugs. In this work, a novel framework named NRPredictor, has been proposed which uses three ensemble learning and one feature selection algorithm for Non-Reproducible bug prediction. The prediction performance of the proposed framework has been examined using projects of Bugzilla bug tracking system. Three open-source projects viz. Mozilla Firefox, Eclipse and NetBeans have been used for evaluating the prediction performance. While forecasting the fixability of bug reports, the experimental findings reveal that NRPredictor surpasses traditional machine learning techniques. For Mozilla Firefox, Eclipse, and NetBeans projects, NRPredictor, delivers performance (in terms of F1-score) up to 88.3, 87.8, and 87.4% respectively. An improvement in performance up to 6.1, 5 and 2.7% has been obtained for NetBeans, Eclipse, and Mozilla Firefox projects, respectively as compared to the best performing standalone machine learning classifier.

Keywords Non-reproducible bugs · Reproducible bugs · Machine learning · Classification · Ensemble learning · Feature selection · Fixability prediction · Mining software repositories

✉ Kulbhushan Bansal
kul_bansal@yahoo.co.in

Gopal Singh
gsbhoria@gmail.com

Sunesh Malik
suneshmlk@gmail.com

Harish Rohil
harishrohil@gmail.com

[1] Department of CSE, Chaudhary Devi Lal University, Sirsa, Haryana, India

[2] Department of CSA, Maharshi Dayanand University, Rohtak, Haryana, India

[3] Maharaja Surajmal Institute of Technology, Delhi, India

[4] Department of CSE, Chaudhary Devi Lal University, Sirsa, Haryana, India

1 Introduction

The emergence of numerous project management tools and approaches have attributed to the increased project complexity and team-based initiatives. The use of bug tracking tools is an important aspect of open-source project management. Bug reports and their debugging procedures have become an unavoidable part of software development during the previous few decades (Kamkar 1998). Software developers work hard to ensure that the software entity is bug-free (Fagan 2002). However, in actuality, a high number of defect reports are encountered by any software. For collecting, organising, and monitoring of incoming bug reports, large software companies use bug tracking systems (Breu et al. 2010). Bug tracking systems (BTS) are also termed as issue tracking

systems (ITS), hence, they have been used interchangeably in this paper. Past works have produced a multitude of research endeavours to ensure genuine treatment of bugs. Out of these, most works are concerned with bug summary generation (Gupta and Gupta 2021), meta-field prediction (such as severity, priority, etc.) (Kumari et al. 2020; Sharma et al. 2021), duplicate identification (Neysiani et al. 2020; Isotani et al. 2021), developer recommendation (Goyal and Sardana 2016; Ye et al. 2020), reopening (Tagra et al. 2021), fixing time (Lee et al. 2020; Kumari et al. 2020), localization (Li et al. 2021) etc. These methods collect numerous bug report parameters/ meta-fields from bug repositories and use them to create the prediction models for certain tasks.

In basic scenario, various stakeholders of BTS (such as end users, developers, and testers) file the problems found to BTS (Anvik 2006). Bug triager checks the bug's existence and if found as valid, assigns the bug to the developer (Zhang et al. 2014). The developer uses information supplied by reporter in the bug report to reproduce the problem (Shokripour et al. 2015). However, if the developer is unable to replicate the bug, it is designated as Non-Reproducible (NR) (Joorabchi et al. 2014). In bug repositories, NR bug reports are a significant performance issue since they occupy a significant amount of developer's time and effort. NR bugs create delay in bug fixing and they may even lead to the release of software project with critical bugs (Rahman et al. 2020). Hence, the detection of NR bugs in early bug life cycle is an open research problem requiring investigation.

Joorabchi et al. (Joorabchi et al. 2014) published first characterization study on NR bugs. They addressed four research questions related to quantitative and qualitative analysis of NR bugs. They manually mined the cause categories and transition patterns of about 1600 NR bugs. Further, they studied the NR bugs which eventually got fixed. After conducting an exploratory investigation on 6 bug tracking repositories, they discovered that 17% of all bug reports are resolved as NR. The cause categories for 1,643 NR bugs are defined as Interbug Dependencies (45%), Environmental Differences (24%), Insufficient Information (14%), Conflicting Expectations (12%), Non-deterministic Behaviour (3%) and Others (2%). Furthermore, only around 2% of all NR bug reports get fixed with code fixes in the end, while the other half are implicitly repaired. This work puts some light on the factors leading to make bugs NR, however, it does not provide any mitigation strategy. It does not provide any mechanisms to improve the bug fixing process. Further, (Goyal and Sardana 2017) presented a sentiment analysis based study of developers who worked on NR issue fixes. They discovered that developer comments posted in NR bug reports are more negative than standard defects. Machine learning classifiers are then used to forecast fixable issues from NR flagged bugs. Our work is different from this work as we do not study developer sentiments as bug reports are technical documents and they constitute technical keywords which lack any kind of sentiment. Secondly, the prediction model proposed by Goyal and Sardana (2017) deals with the prediction of reopened bugs whereas our work deals with the prediction of new bugs. Hence, the work presented in this paper attempts to fill the research gap present in the literature " to provide a mitigation strategy to early predict the NR bugs".

To the best of our knowledge, there does not exist any work on early prediction of NR bugs. A unique NRPredictor framework is provided in this paper to forecast the fixability of bug reports. For fixability prediction, the proposed model combines feature selection and ensemble learning methods. Ensemble-based approaches use the capabilities of several different basic classifiers to improve classification accuracy (Alzubi 2015). In this method, the training data is first separated into many disjoint groups, and then each subset is trained using a base classifier. Feature selection algorithms try to reduce the complexity of the system.

The following are the current work's key research contributions (RC):

1. The early fixability problem in bug reports has been examined. In this RC, the problem of prediction of bug type (R or NR) when a new bug is filed to BTS has been examined.
2. A novel framework, NRPredictor, based on feature selection and ensemble machine learning algorithms, has been proposed. In this RC, a novel framework has been proposed which predicts whether a new bug report will get fixed or it will be marked as NR.
3. Thirteen machine learning classifiers (Bayes Net, Naive Bayes, Naive Bayes Multinomial Text, Naive Bayes Updateable, IBk, Zero-R, JRip, OneR, PART, Decision Table, J48, Rep Tree and Random Tree) along with three ensemble learning techniques (Bagging, Boosting and Stacking) and one feature selection technique (Classifier Attribute Evaluator) has been utilized in proposed framework, NRPredictor. In this RC, traditional and advanced machine learning algorithms have been utilized for prediction of a newly reported bug as Fixable or NR.
4. The proposed framework, NRPredictor has been tested on three large-scale, well-known, long lived, open-source Bugzilla repository projects, namely NetBeans,[1] Eclipse,[2] and Mo-zilla Firefox.[3] In this RC, bug reports from three long lived software projects have been col-

[1] https://netbeans.org/bugzilla/.

[2] https://bugs.eclipse.org/bugs/.

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

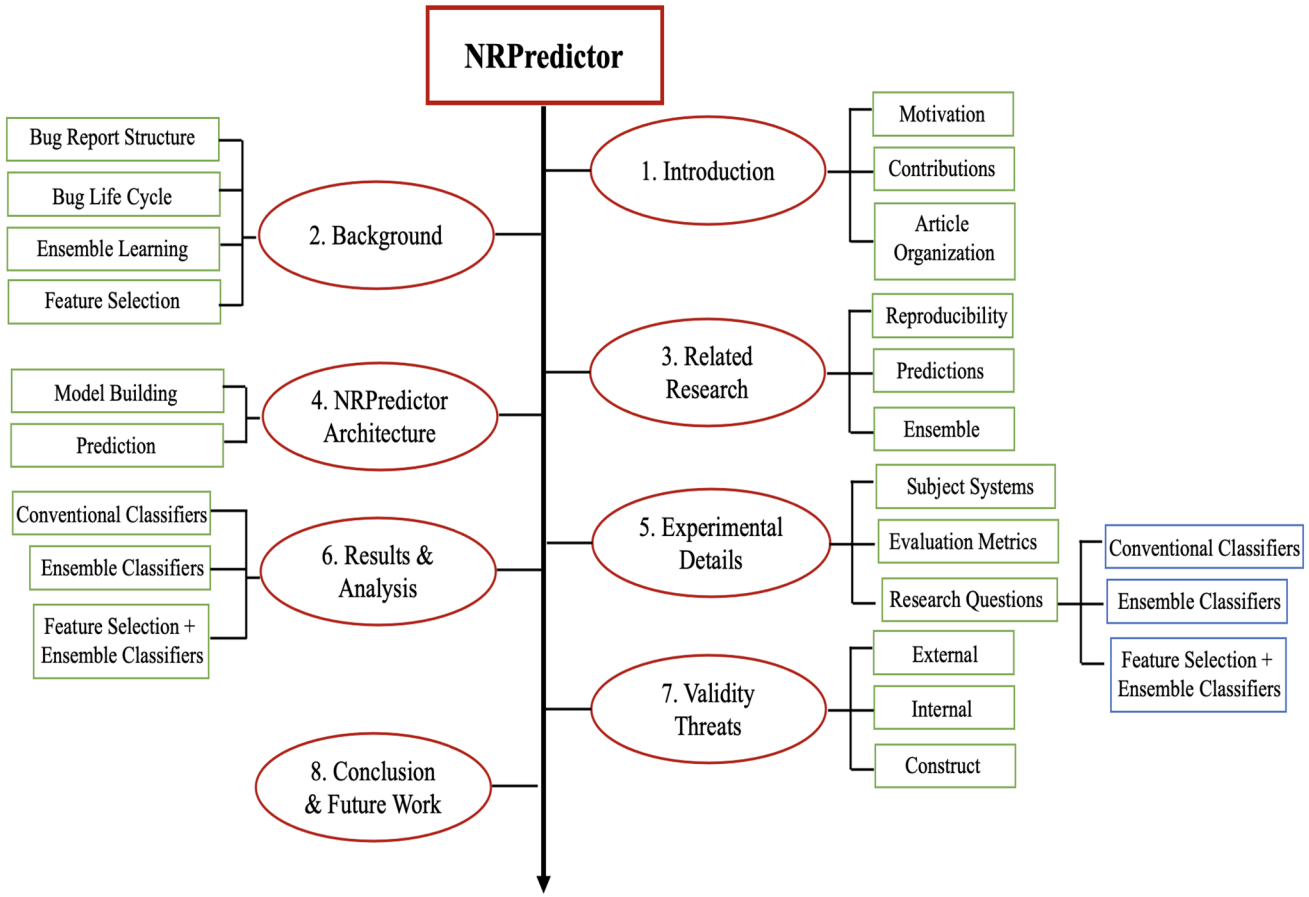

Fig. 1 The roadmap for article

lected and processed to be fed into NRPredictor framework for prediction purposes.

5. Four evaluation metrics (Precision, Recall, F1-Score, Area under Receiver Operating Ch-aracteristic Curve) have been used for comparison. The experimental findings reveal that the proposed framework, NRPredictor outperforms traditional machine learning techniques consistently. F1-scores up to 88.3, 87.8 and 87.4% for Mozilla Firefox, Eclipse and NetBeans projects has been obtained respectively. In this RC, performance evaluation of proposed framework is conducted using various performance evaluation metrics.

The paper is organised as per the roadmap defined in Fig. 1. Section 2 goes through the background information which includes NR bug report structure, the bug report life cycle, and the ensemble and feature selection approaches used in this paper. The relevant past work across three domains (reproducibility, prediction and ensemble techniques) is discussed in Sect. 3. The architecture of proposed NRPredictor framework is detailed in Sect. 4. The experimental details are presented in Sect. 5. The results and analysis of the experimental evaluation are presented in Sect. 6. The risks to validity are discussed in Sect. 7. Finally, Sect. 8 brings the work to a close by providing conclusion. Section 9 discusses future research prospects.

2 Background

This section covers the necessary background information for this research, such as the fundamental layout of a bug report, the normal life-cycle of an issue, and various ensemble learning & feature selection methodologies used.

2.1 Bug report structure

A bug report is a record that contains complete information concerning a problem. It contains a number of bug meta-fields as well as some textual material. Bug id, product, component, platform, hardware, version, operating system, severity and priority, milestone, status, resolution, reporter's name, time-stamp of report submission, assignee, and so on are all included in the meta-fields. A quick summary

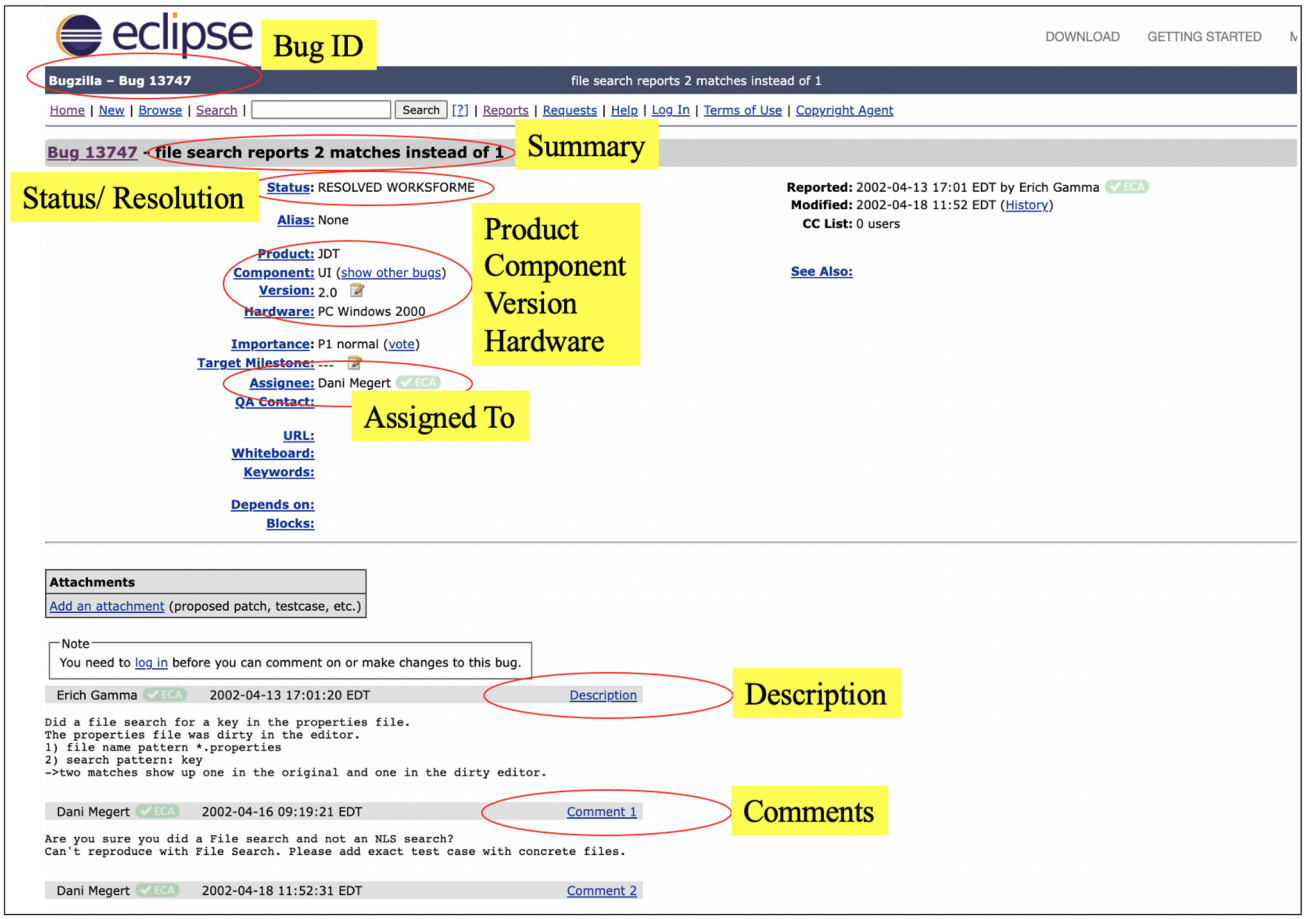

Fig. 2 An example of a NR bug report

or tagline, a detailed explanation of the error, and comments provided by the reporter, developer, or testers are all included in the textual information. Figure 2 displays an example of an Eclipse Project's NR bug report (Bug id: 13747).[4]

In Fig. 2, the unique serial number assigned for every problem is referred to as the "Bug ID". The term "Product" refers to the wide region from which the bug sprang. The term "Component" refers to the product's next level of categorisation. One or more components can be found in a single product. The term "Version" refers to the software product version in which a defect was discovered. The "Status" parameter indicates where the bug is in its life cycle. The name of the developer who has been assigned task for fixing the fault is referred to as "Assigned-to." The term "Summary" refers to a one-sentence explanation of the reported defect. "Description" refers to the bug report's whole comprehensive specification, which is often written by reporter. Description usually consists of 3 main elements: noticed behaviour, reproducible processes, and predicted software behaviour (Chaparro et al. 2017). The term "Comments" refers to an open-ended discussion among developers to find viable remedies for bug solving.

Along with particular meta-fields and textual contents, bug report contains attachments, URLs, and automatically produced notes. Extra information about the problem is commonly included in these columns, like test cases, patch filed, user-supplied screen shots, the URL of website containing issue, similar duplicate bugs, and so on.

2.2 Bug life-cycle

A bug progresses via various phases throughout its existence. Figure 3 shows life-cycle of a bug report in Bugzilla repository.[5] For different projects, life-cycle stages may vary slightly but the mainstream order remains same. Initially, any bug's existence is UNCONFIRMED. A bug reporter has reported the problem thus far, but its existence has yet

[4] https://bugs.eclipse.org/bugs/show_bug.cgi?id=13747.

[5] https://www.bugzilla.org/docs/2.18/html/lifecycle.html.

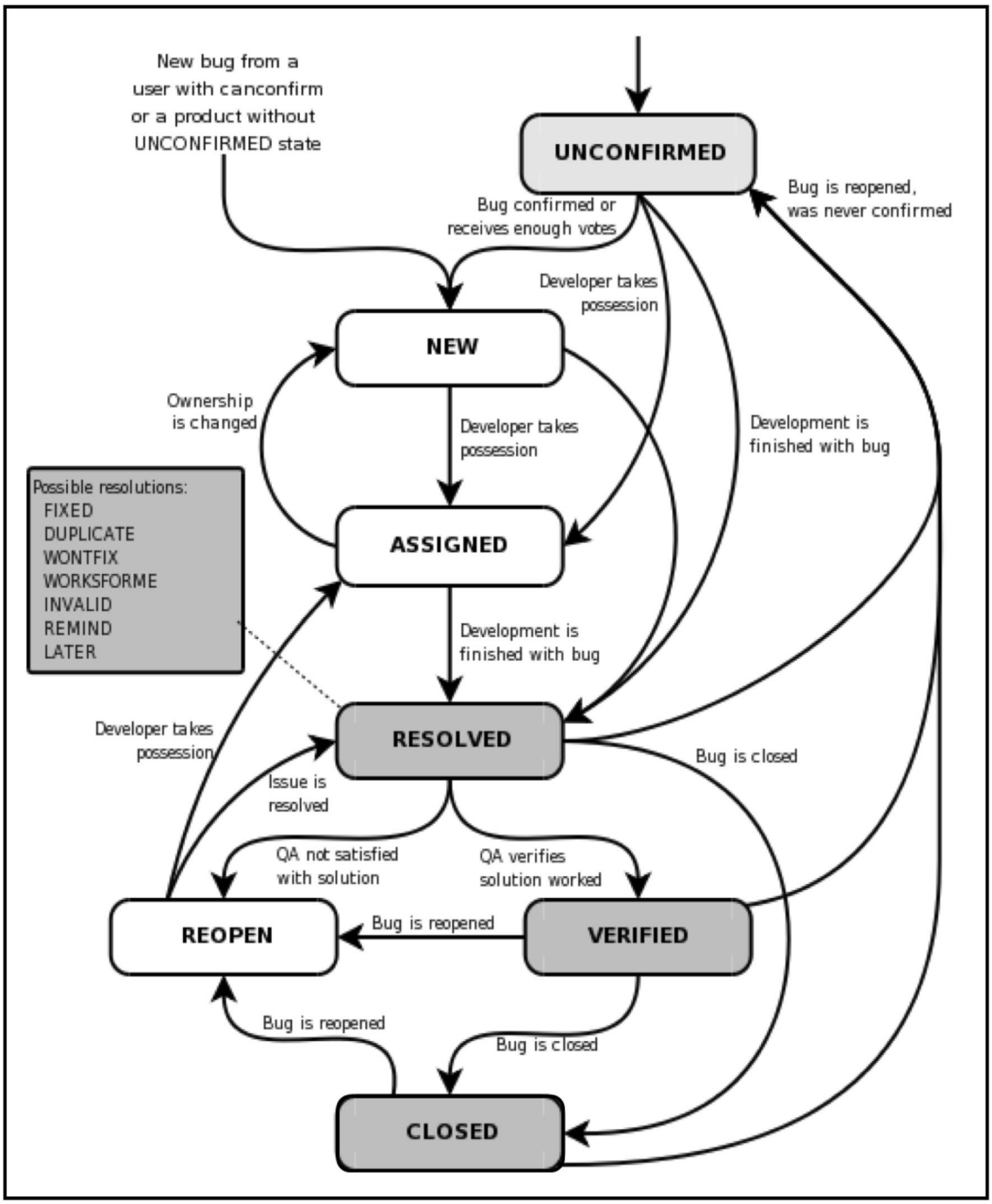

Fig. 3 Life-cycle of a typical bug report in Bugzilla Repository

to be validated. The existence of an unconfirmed issue is confirmed by the bug triager, who then labels the validated bug as NEW. Because it is presumed that a bug submitted by an expert is real and existent, it may reach NEW state immediately. The bug triager assigns a verified bug to the developer and labels the resolution with ASSIGNED. The allocated developer investigates the problem, reproduces it, and performs appropriate modifications for fixing it.

There are numerous bug report resolutions available in the RESOLVED status, including fixed, duplicate, won't fix, worksforme (NR), invalid, remind, and later. The resolution of the problem is indicated as fixed once the assigned developer has successfully made relevant source code adjustments. However, the assigned developer does not have to always discover a valid remedy to the reported issue. A software developer may discover that the claimed

problem is not unique when investigating a bug report. It might be a duplicate of an existing or fixed problem, or it could have the same basic cause as another bug. In this case, the bug's resolution is marked as duplicate (Sureka and Jalote 2010). The resolution of a bug report that outlines a non-rectified issue is set as won't fix. The problem is marked as NR or worksforme if it cannot be recreated using the information given in the bug report. When additional information is added to the NR bug, it may be reopened, which may aid in replicating the problem. A bug is marked as resolving invalid when it is proven to be illegible or spam. Invalid bugs are considered as not real problems (Yuan et al. 2021). Bugs that force third-party software or websites to make changes, for example, constitute a breach of legal and contractual obligations. If bug requires further information and cannot be addressed immediately, then it is marked with resolution remind or later (Abou et al. 2021).

2.3 Ensemble learning/ classification

The classification refers to division or category in a system that organises or categorises objects. Initially, manual classification of items was popular. Manual classification, on the other hand, has the drawbacks of being exceedingly time-consuming and fundamentally subjective in nature (Bauer et al. 1999; Alzubi et al. 2018). As a result, automatic classification algorithms were developed. Automatic classification is more objective, quicker, and scalable. It can be effective in more complicated, nuanced circumstances, such as business-specific material, because it provides companies with a more systematic and consistent classification. Artificial intelligence techniques are excessively used in computing for training, forecasting and evaluation purposes (Movassagh et al. 2021). Automatic document categorization can benefit from machine learning and artificial intelligence techniques to improve speed and efficiency.

Ensemble classification approaches are a type of meta machine learning algorithm that has recently gained popularity. To improve predictive performance, these strategies aggregate predictions from different learning algorithms (Dietterich et al. 2000). Distinct machine learning classifiers have different fundamental principles and training data sensitivity. As a result, various categorization systems make different predictions based on the data. These various outcomes are used by ensemble machine learning algorithms to produce a superior prediction output (Alzubi et al. 2020). These strategies aim to reduce prediction model bias and variance while also attempting to improve prediction accuracy using only one of the constituent learning algorithms. Three alternative ensemble classification approaches were investigated in this paper.

1. *Bagging*: Bagging also referred as "Bootstrap Aggregating" is a meta-estimator that uses several random subsets of the original dataset to fit a base classifier. The original dataset is re-sampled via replacement, and the predictions of several learners are combined for generating final result. Breiman demonstrates bagging approach is helpful for unstable learners (Breiman 1996).
2. *Boosting*: This approach combines various weak classifiers to produce a powerful classifier. If the model has a large error rate, it is deemed weak (0.5 or more for binary classification). The ensemble classifier is constructed to reduce the mistakes obtained in the previous step throughout each iteration. Iterations are repeated till the point of maximum iterations or till whole training dataset is correctly predicted (Freund and Schapire 1995).
3. *Stacking*: Stacked generalisation, is an ensemble strategy which uses a training dataset to train several base classifiers and then uses these base classifiers to build a new dataset. Then, using combiner machine learning approach, this new dataset is incorporated (Wolpert 1992).

2.4 Feature selection techniques

Raw machine learning data is made up of a variety of attributes, some of which are useful for making predictions and others that aren't. Feature selection approaches assist in identifying a set of relevant traits from a large number of options. The Classifier Attribute Evaluator was used to select features in this paper. The attribute evaluator is a tool for evaluating each attribute (also known as a column or feature) in your dataset in relation to the output variable (e.g. the class).

3 Literature review

Since the previous two decades, the study of software flaws has been a hot topic of research. (Perry and Stieg 1993) presented a preliminary research on the investigation of reported problems in major software projects. The authors performed a poll to find out what kinds of difficulties users report, how they are discovered, and at what point of testing they are filed to BTS. Since then, various studies have been done that examine different buggy locations. This section goes into previous research in these buggy domains, which are divided into three categories: reproducing bug reports, prediction models in bug fixing, and ensemble learning in bug fixing.

3.1 Reproducing bug reports

A bug report comprises 3 key elements: procedures for replication of problem, what reporter anticipated to observe, and what reporter actually observed (Chaparro et al. 2017). The above listed 3 elements aid software developers in verifying, finding, and replicating the problematic scenario, as well as understanding the fundamental cause of the fault. After that, the allocated developer fixes the problem by making modifications to the source code. Reproducing a bug report is famously difficult since engineers are only given limited information about the failure, such as a memory dump. ReCrash is an automated approach to construct test-cases for simulating a software failure introduced by Artzi et al. (2008). CRASHDROID was created by White et al. (2015) for Android apps to automate system of replicating problems. Jin and Orso (2012) established BugRedux, an approach to gather extra information from the buggy ground and transmits the collected information to developers for repeating the failure circumstance. Despite the fact that studies exist to assist developers in recreating problem reports, their in-field performance is quite poor. RecrashJ, the Java version of ReCrash, for example, has a performance overhead of 13–64%.

If a developer is unable to replicate an issue, the resolution is marked as NR. It is often perplexing and time-consuming for engineers to manage NR problems. An empirical analysis over 32,000 NR bugs was given by Joorabchi et al. (2014) which discovered that resolution NR is assigned to 17% of all bug reports, and that just 3% of NR-assigned bug reports get repaired. Further, 1,643 NR issue reports are manually sorted into six different cause groups, including inter-bug dependencies, environmental differences, insufficient information, conflicting expectations, non-deterministic bugs, and others. Goyal and Sardana (2017) did a sentiment study of developers who worked on NR issue fixes. They discovered that developer comments posted in NR bug reports are more negative than standard defects. Machine learning classifiers are also used to forecast fixable issues from NR flagged bugs.

Table 1 presents the review of literature in the broad domain of NR bugs. From Table 1 it has been observed that these works put some light on the factors leading to make bugs NR, however, they do not provide any concrete mitigation strategies. Joorabchi et al. (2014) does not provide any mechanisms to improve the bug fixing process. Goyal and Sardana (2017) presented a sentiment analysis based model to forecast fixable issues from NR flagged bugs. However, their work deals with the prediction of re-opened bugs whereas the current manuscript deals with the prediction of new bugs. Hence, the work presented in this paper attempts to fill the research gap present in the literature "to provide a mitigation strategy to early predict the NR bugs".

Table 1 Review of past works on non-reproducible bugs

Reference	Objective of study	Technique Employed	Datasets used	Results
Joorabchi et al. (Joorabchi et al. 2014), 2014	Empirical analysis on NR bugs	Quantitative and qualitative analysis	Mozilla firefox, eclipse, mediawiki, moodle, firefox android, industrial dataset	17% reported bugs are NR. Cause categories: interbug dependencies (45%), Environmental differences (24%), Insufficient information (14%), Conflicting expectations (12%), Non-deterministic behaviour (3%)
Goyal & Sardana (Goyal and Sardana 2017), 2017	Fixability prediction of NR bugs (re-opening prediction)	Empirical analysis & ML	Mozilla firefox, eclipse	developer's are reluctant to work on NR bugs. ML based models can predict fixable NR bugs with performance up to 75%
Rahman et al. (Rahman et al. 2020), 2020	Empirical analysis on NR bugs	User study	Mozilla firefox, eclipse	Key factors: bug duplication, intermittency, missing information, ambiguous specifications, third-party defects, etc
Goyal & Sardana (Goyal and Sardana 2018), 2018	Characterization study of developers involved in fixation of NR bugs	Quantitative analysis	Mozilla firefox	developers who fix NR bugs possess higher expertise in related topics as compared to developers who fix R bugs

3.2 Prediction models in bug fixing

In the arena of software engineering research, debugging is a well-known concept. A lot of research and prediction models have been established for bug summarization (Koh et al. 2021), bug triaging (Mohan et al. 2016; Goyal and Sardana 2017), duplicate detection (Rocha and Carvalho 2021), fix time prediction (Yuan et al. 2021), blocking bug prediction (Cheng et al. 2020), reopened bugs (Shihab et al. 2013), etc. Garcia and Shihab (2014) developed a bug-blocking prediction model. They inspected the performance of decision tree, Naive Bayes, kNN, random forest, and Zero-R classifiers using 14 bug attributes to discriminate between blocking and non-blocking bug reports. Using 10-fold cross validation, they were able to reach an F-measure of $15-42\%$. Xia et al. (2015) expanded on this research to address the problem of class imbalance in the blocking problem prediction. It has been found that ensemble learners successfully operate upon the phenomena of class imbalance and, as a result, may increase minority class prediction accuracy. Our forecasting is based on the same principles. Shihab et al. (2013) took care of the reopened bug reports. For bug report classification, they employed 22 distinct characteristics divided into four categories: developer work habits, bug report, problem fix, and team. While predicting reopened bugs, they reported accuracy values 52.1–78.6% and recall values 0.5–94.1% for reopened bug finding. Comment text and last status were discovered to be the most important elements.

Hewett and Kijsanayothin (2009) built a model that anticipated how long it would take to fix software bugs. On a medical software system dataset, the suggested model has an accuracy of 93.44%. Guo et al. (2010) presented an architecture for predicting fixability of a freshly discovered problem. On Microsoft Windows Vista project, the suggested model achieved accuracy values of up to 68% and recall values of up to 64%. Zimmermann et al. (2012) analysed and evaluated reopened bug complaints to determine likely causes of reopening and to assess their effect. Table 2 presents the review of literature related to prediction models in different phases of bug handling. Meta-heuristic algorithms such as AHP, TOPSIS, etc. are also used in bug handling processes nowadays (Goyal and Sardana 2017). Research is in progress to further optimise the meta-heuristic algorithms (Agushaka et al. 2022; Abualigah et al. 2021, 2022; Oyelade et al. 2022; Abualigah et al. 2021; Sethuraman et al. 2019).

Various studies have showed that machine learning classifiers are successful in predicting different buggy locations(Garcia and Shihab 2014; Ahmed et al. 2021; Malhotra et al. 2021; Rashmi and Kambli 2020). Our research focuses on NR defects, as opposed to many prediction models present in past works relating to software debugging procedures. The tests were carried out on bug reports identified with the classes Reproducible (R) and NR. The goal is to forecast which bug reports can be fixed.

3.3 Ensemble learning in bug fixing

In the literature of machine learning classification, ensemble learning plays an essential role. Numerous ensemble based classifiers have been suggested to increase the performance of a traditional machine learning classifiers (López et al. 2013). The effectiveness of ensembling technique may be attributed to the diversity of their base learners (Guo et al. 2008). As a result, ensemble classifiers use a group of basis classifiers to build a prediction model. There are many different forms of ensemble models, e.g., bagging (Breiman 1996), boosting (Freund and Schapire 1995), stacking (Wolpert 1992), etc. The bagging approach uses the same basic classifier to train several classifiers, which are then combined using an unweighted majority voting mechanism. The majority of votes determines the final forecast. Bagging approaches typically outperform single model algorithms by a wide margin. It is never considerably insufficient since it mitigates the classifiers' volatility by raising the victory proportion (Phua et al. 2004). Boosting is an iterative method which provides a weight value to the training dataset in each iteration. During 1st run, all weights are put equal (Freund and Schapire 1995). The weights of improperly categorised instances are raised with each repetition. This helps weak learners to concentrate on the training set's difficult cases. Using a meta-classifier like Logistic Regression, stacking combines numerous classifiers (Wolpert 1992). Multiple basic classifiers are used to classify a single test case. The output of these several basic classifiers is fed into a meta-classifier, which produces the ultimate prediction.

In the literature on software debugging, ensemble learners have been employed in a number of studies. Stacking ensemble approach for automated bug triaging was assessed by Jonsson et al. (2016). Stacking beats standard machine learning techniques for the multi-class issue of developer selection, according to their findings. Goyal and Sardana (2019) provided an empirical study on bug triaging strategies using ensemble classification algorithms (bagging, boosting, majority voting, average voting, and stacking). Ensemble classifiers outperformed standard machine learning algorithms in the identification of an appropriate developer to handle the bug report, according to the researchers.

Limsettho et al. (2018) presented a SMOTE-based technique to predict cross-project defects. Laradji et al. (2015) found that ensemble learning had a favourable influence on software fault prediction. Xia et al. (2015) proposed ELBlocker, an ensemble learning approach for predicting blocking problems. They demonstrated that orthodox machine learning methods perform poorly for severely unbalanced datasets, but ensemble-based strategies aid in the

Table 2 Review of past works on prediction models in bug fixing

References	Objective of study	Datasets used	ML techniques	Results
Fan et al. Fan et al. (2018), 2020	Valid bug prediction	Eclipse, netbeans, mozilla, firefox, thunderbird	SVM, random forest	Area under receiver operating characteristic curve up to 0.81
Shihab et al. Shihab et al. (2013), 2013	Reopened bug prediction	Eclipse, apache, open office	Decision tree	recall up to 94.1%
Xi *et al.* Xi et al. (2019), 2019	Bug triaging	Eclipse, mozilla, gentoo	SVM, CNN	accuracy up to 0.73
Neysiani and Babamir Neysiani and Babamir (2020), 2020	Duplicate detection	Android	k-nearest neighbour, SVM, Linear regression	accuracy up to 97%
Tamrawi *et al.* Tamrawi et al. (2011), 2011	Bug triaging	Eclipse, mozilla firefox	Naive bayes, C4.5, SVM	Accuracy up to 88.8%
Ekanayake Ekanayake (2021), 2021	Severity prediction	Unix kernel	Naive bayes, logistic regression, decision tree	Area under receiver operating characteristic curve up to 0.65
Shatnawi and Alazzam Shatnawi and Alazzam (2022), 2022	Priority and severity prediction	Eclipse	Multi-nominal naive bayes, random forest, bagging, adaboost, SVM, KNN, linear SVM	Accuracy up to 100%
Dao and Yang Dao and Yang (2021), 2021	Priority Prediction	Eclipse, mozilla, open office	Random forest, support vector machine, and Multinomial Naïve Bayes	Macro precision 26.84%
Zhou et al. Zhou et al. (2019), 2019	Software defect prediction	NASA, PROMISE, AEEEM, relink	Naive bayes, support vector machine, random forest, logistic regression	Area under receiver operating characteristic curve up to 0.80

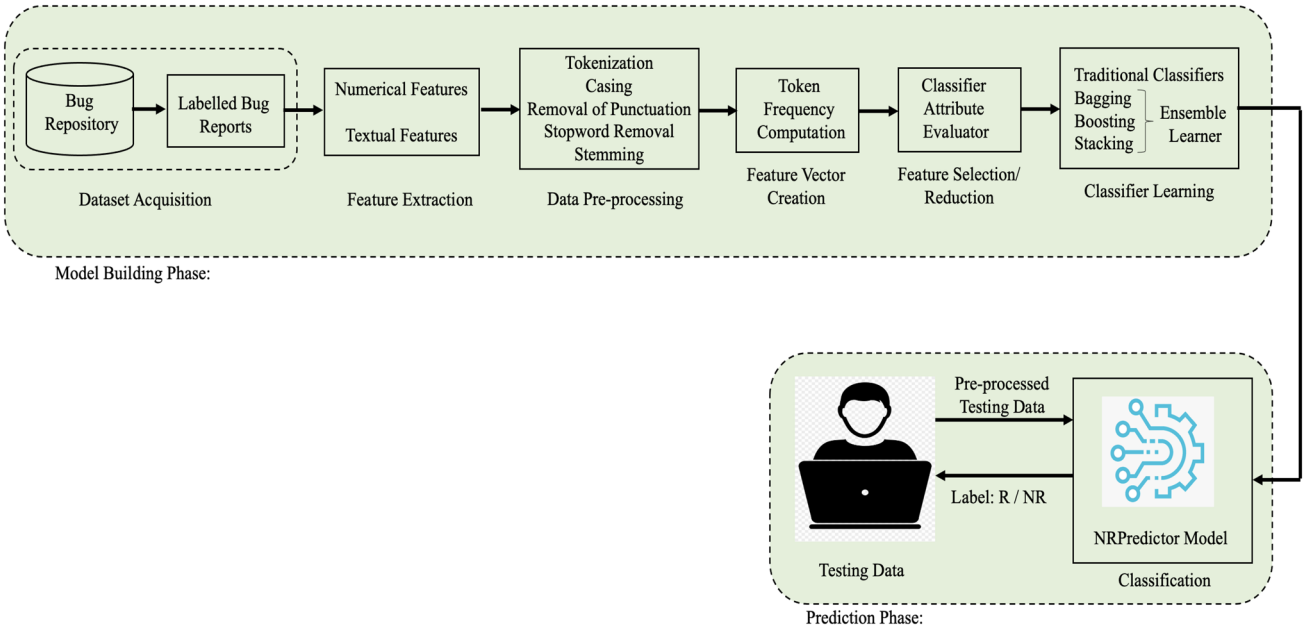

Fig. 4 Architecture of proposed framework: NRPredictor

development of stronger prediction models. They found that employing an ensemble-based method improved F1-Score by 14.69% when compared to traditional machine learning classifiers. Lal, et al. (2017) introduced an ensemble-based model for logging predictions called ECLogger. To deal with the problem of class imbalance, they employed bagging and voting ensemble approaches. Using ensemble-based approaches, they were able to obtain better prediction performance. Motivated by these studies (Jonsson,Borg,Broman,Sandahl,Eldh and Runeson, 2016; Lal et al., 2017; Xia,Lo,Shihab,Wang and Yang, 2015) the issue of fixability prediction has been addressed. Ensemble based techniques along with feature selection techniques has been used to predict the fixable and NR bugs optimally.

4 NRPredictor framework

This section describes the architecture of proposed framework, NRPredictor as shown in Fig. 4. The framework, NRPredictor, is divided into two phases: model building and prediction.

PHASE 1: model building phase

Initially, previous bug reports from the bug repository with known labels (R or NR) are used as input. Following that, different characteristics are retrieved and pre-processing techniques are used. Machine learning is then used to learn multiple models based on the retrieved information. Finally, this step generates hybrid models for forecasting the class for bug reports that haven't been tagged. The following are the phases in model building phase of proposed framework, NRPredictor:

1. *Dataset acquisition:* First, bug reports with class labels (R and NR) are collected from various projects of Bugzilla repositories in this stage. A bug report is considered as NR if it is marked as NR or "worksforme" during the life.
2. *Feature extraction:* Next, 9 different features are drawn out from the collected bug reports. Among the 9 features, 8 are numerical (component, severity, priority, operating system, hardware, version, number of comments and cc count) and 1 is textual in nature. All of the features along with their descriptions which are utilised in this paper are listed in Table 3.
3. *Data pre-processing:* The bug reports' textual contents are evaluated in this stage to build a feature vector containing critical keywords. This stage entails cleaning up the contents of the text. The bug report's textual description is received first. The phrases are then tokenized, with all concatenated terms being broken up and further their case changed to the lower version. Stop words have been eliminated because their frequency is higher but they do not constitute any information. Python Natural Language Processing ToolKit (NLTK) has been used to leverage the stop word list. Porter stemming (supplied by Python NLTK) is then used to transform the remaining tokens to their root phrase. Stemming is the process of combining closely similar phrases (for example,

Table 3 Feature/ parameter list used in NRPredictor framewore

Name of parameter	Description
Component	Component where the problem first appeared
Severity	The intensity of the bug's influence on the software system. The reporter of the bug report assigns this field. Minor, Major, Normal, Trivial, and blocking are all possible values
Priority	Measure of quickness required for addressing the bug. The developers in charge of resolving the problem assign this field. Priority is usually determined by the severity field. Values that might be used: P1 (highest priority: urgent addressing is required), *P*2, *P*3, *P*4, and *P*5 (lowest priority: urgent addressing is required) (lowest priority)
Operating system	The operating system where the problem first appeared
Hardware	Hardware of the system where the problem first appeared
Version	The software's version
Number of comments	Before designating a bug report as NR, the number of comments submitted by developers is counted
CC Count	The number of developers on the bug report's CC list
Keywords	Terms derived from the bug report's textual content

stemming converts the two words *likes* and *likely* to root terms *like*).

4. *Feature vector creation:* Next, the frequency of each token has been determined after data preprocessing, and a textual feature vector has been constructed. In this stage, the top 100 tokens derived from textual characteristics with the highest overall frequency were considered for textual features.
5. *Feature selection/ reduction:* Feature selection techniques help to obtain a set of relevant features from the list of all available features. In this work, Classifier Attribute Evaluator has been used for feature selection. The attribute evaluator is a method for evaluating each attribute in your dataset (also known as a column or feature) in relation to the output variable (e.g. the class).
6. *Classifier learning:* Various NRPredictor models were created in this stage. First, 13 machine learning algorithms (Bayes Net, Naive Bayes, Naive Bayes Multinomial Text, Naive Bayes Updateable, IBk, Zero-R, JRip, OneR, PART, Decision Table, J48, Rep Tree and Random Tree) from four families: Bayes, trees, rules and lazy were used to learn the models. Comparison of the performances of 13 machine learning classifiers was used to conduct an empirical investigation. Then, employing 13 machine learning classifiers, three ensemble learning strategies were used. To create prediction models for fixability prediction, three ensemble-based strategies (bagging, boosting, and stacking) are utilised.

PHASE 2: prediction phase

This phase accepts the bug report as input for which the class label has to be anticipated. Then it draws out the bug report's characteristics, uses pre-processing techniques, and predicts label using the hybrid models created during the model building phase (Phase 1). The following step is involved in the prediction phase of the NRPredictor framework:

Classification In this step, first, 9 features for test bug report are extracted. Then, pre-processing is done by applying tokenization, casing, removal of punctuation, stop word removal and stemming corresponding to the test bug report. The pre-processed feature vector is then supplied to the proposed framework, NRPredictor. On the basis of various learned hybrid models, the test bug report is classified and a label is predicted corresponding to it (label is either R or NR). Next, the predicted label by proposed framework, NRPredictor and the ground truth label is considered and various evaluation metrics are computed to evaluate the prediction performance of learned models and NRPredictor framework.

5 Experimental details

The subject systems, implementation details, assessment measures, and research issues addressed in this paper are detailed in this section. The experimental setup constitutes MacBook Pro with 8 GB of memory and a 2.7 GHz Intel Core i5 processor running Mac OS X 10.13.1. However, few machine learning algorithms were not able to be run in the given platform. The time threshold of 8 hrs has been used. The requirement of high computing power has been considered for all such combinations. Hence, all such algorithm combinations have been run on a GPU equipped with NVIDIA Tesla V100 with 16GB RAM (5120 CUDA Cores). Four such cores in parallel manner have been used for this work. For experimentation, Python programming language has been used with Jupyter Notebook as Integrated Development Environment (IDE). Scikit machine learning library

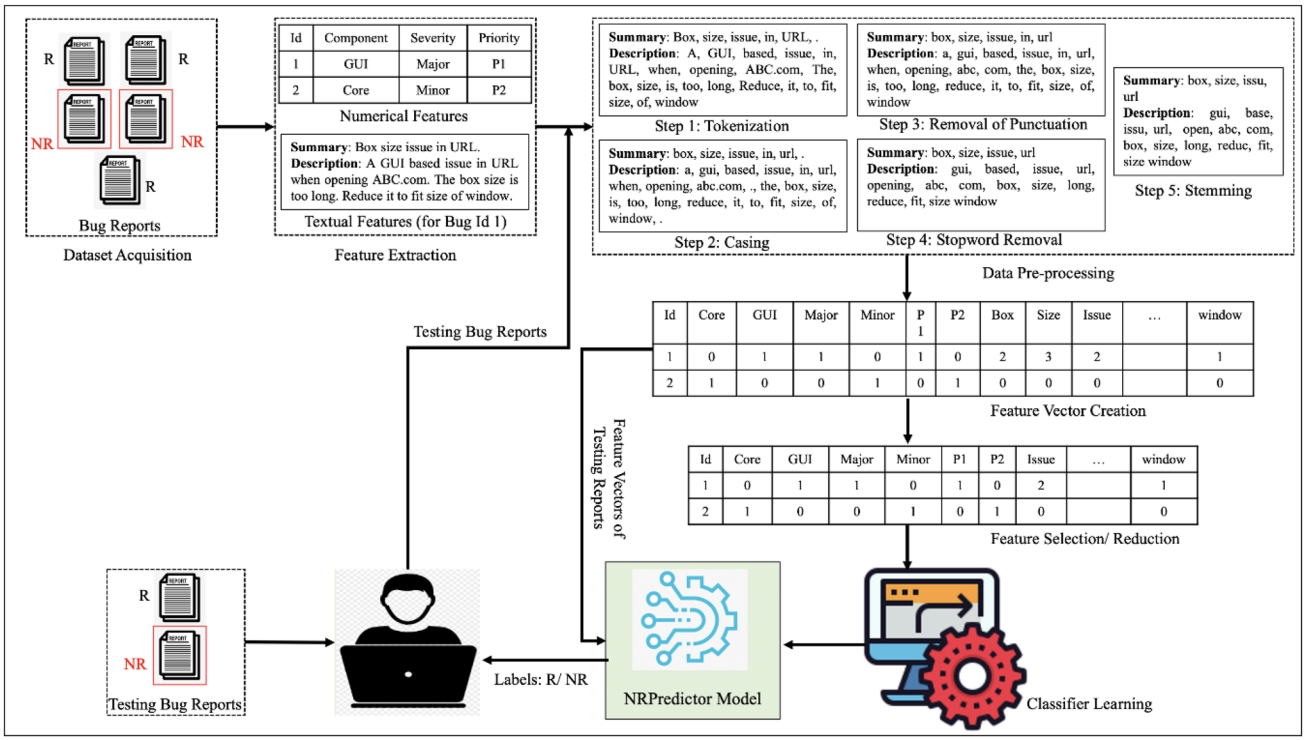

Fig. 5 General procedure of proposed framework: NRPredictor

has been used for building conventional and ensemble classification models.

5.1 Subject systems

This research examined bug reports obtained from three projects of Bugzilla BTS: NetBeans, Eclipse, and Mozilla Firefox. NetBeans is an open source development environment, tooling platform, and application framework for creation of programs using modules.[6] It is a Java application. Eclipse is a framework that contains a number of tools for creating a Java integrated development environment (IDE).[7] It was introduced in 2001 and is the most extensively used Java IDE. It is primarily written in Java. Mozilla Firefox is an open-source web browser launched in 2002.[8] The popular web browser is available for a variety of systems. It is written in C++.

Since a huge number of bug reports are sent on a daily basis, the Bugzilla dataset's different projects are capable of assessing trends and evaluating recommended techniques. While choosing experimental projects, it was attempted to cover a wide variety of disciplines. These projects have been used frequently in the past studies Anvik and Murphy (2011); Bhattacharya and Neamtiu (2010); Tamrawi et al. (2011). These projects' popularity and maturity makes it ideal for researching any novel bug-handling techniques. A total of 261551 bug reports were gathered and divided into two groups (R and NR). The distribution of bug reports into R and NR categories is depicted in Table 4.

5.2 Implementation details

The proposed framework, NRPredictor has been evaluated using bug reports obtained from three long lived projects of Bugzilla BTS: NetBeans, Eclipse, and Mozilla Firefox. The general procedure of implementation is depicted in Fig. 5. First, the bug reports from the bug repository with known labels (R or NR) are collected and are considered as ground truth. Next, 8 numerical (component, severity, priority, operating system, hardware, version, number of comments and cc count) and 1 textual feature is extracted from bug

Table 4 Distribution of bug reports in *R* and NR categories

Project	Total bugs	*R* bugs	NR bugs
NetBeans	114,177	94,173	20,004
Eclipse	90,788	76,006	14,782
Mozilla firefox	56,586	32,113	24,473

[6] https://en.wikipedia.org/wiki/NetBeans.

[7] https://en.wikipedia.org/wiki/Eclipse(software).

[8] https://en.wikipedia.org/wiki/Firefox.

reports. The textual feature is then preprocessed by using five procedures: tokenization, case conversion, punctuation removal, stopword removal and stemming. Next, the frequency of each token has been determined and 100 most frequent tokens are considered for feature vector creation. Further, feature selection technique called Classifier Attribute Evaluator has been used to further reduce the complexity of dataset. Finally, various NRPredictor models have been created using three machine learning algorithms (Bayes Net, Naive Bayes, Naive Bayes Multinomial Text, Naive Bayes Updateable, IBk, Zero- R, JRip, OneR, PART, Decision Table, J48, Rep Tree and Random Tree) and three ensemble learning strategies (bagging, boosting, and stacking) for each considered project. Finally the built models are evaluated by passing new bug reports as input. The proposed framework, NRPredictor, outputs a class label (R or NR) corresponding to the testing bug report.

5.3 Evaluation metrics

Four common performance assessment criteria were utilised to estimate the achieving power of NRPredictor framework: Precision, Recall, F1-Score, and Area under Receiver Operating Characteristic (ROC) curve. All of the above assessment criteria have been extensively utilised in the software debugging area (Hewett and Kijsanayothin 2009; Shihab et al. 2013; Xia et al. 2015). When utilising NRPredictor models to predict class label, there are four possible outcomes: (1) A bug is predicted as NR bug when it is truly NR (True Positive: *tp*), (2) Predicted as NR but is truly R (False Positive: *fp*), (3) Predicted as R when it is truly NR (False Negative: *fn*), or (4) Predicted as R when it is truly R (True Negative: *tn*). Different metrics such as accuracy, recall, F1-Score, and ROC are computed using these four values:

1. *Precision:* It denotes the proportion of relevant occurrences found among the overall number of examples found. Equation 1 shows the formula for precision.

$$\text{Precision} = \frac{tp}{tp + fp} \tag{1}$$

2. *Recall:* It denotes the percentage of relevant occurrences found out of all relevant examples. Equation 2 shows the formula for recall.

$$\text{Recall} = \frac{tp}{tp + fn} \tag{2}$$

3. *F1- Score:* There is a trade-off between precision and recall measurements. An rise in one statistic frequently results in a drop in the other. As a result, evaluating prediction performance using accuracy and recall is problematic. The F1-Score measure combines the advantages of accuracy and recall metrics. The weighted harmonic mean of precision and recall is represented by the F1-Score. It's a frequently used metric for evaluating performance (Lal, et al. 2017; Xia et al. 2015). Equation 3 shows the formula for F1-Score.

$$F1 - \text{Score} = \frac{(\beta^2 + 1) * \text{Precision} * \text{Recall}}{\beta^2 * \text{Precision} + \text{Recall}} \tag{3}$$

when β is equal to 1, F1-Score is calculated as shown in Equation 4

$$F1 - \text{Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

4. *Area under ROC curve*: The ROC (Receiver Operating Characteristic) curve is a graph of the true positive rate (tpr) vs the false positive rate (fpr). The area under the resulting ROC plot is represented by the area under the ROC curve. It assesses the possibility of an NR bug report being assigned a greater likelihood than an R problem report. The ROC value might be anything between 0 and 1. A larger number in ROC value shows better prediction performance of the developed model of NRPredictor framework.

The effectiveness of NRPredictor models was assessed using the cross validation approach. When just a small number of data examples are present, cross validation is employed to provide an impartial estimate of the model's performance. Data is separated into k equal-sized subgroups in k-fold cross-validation. As a result, the model is generated k times, each time utilising $(k - 1)$ sets of data examples for training the learning classifier and one subset for testing predictions.

5.4 Research questions

The following set of research questions (RQs) are examined in this paper:

- *Research Question 1*: What is the performance of traditional machine learning approaches for predicting bug report reproducibility?
- *Research Question 2*: What is the performance of ensemble machine learning approaches for predicting bug report reproducibility?
- *Research Question 3*: What is the performance of ensemble machine learning techniques after applying feature selection for predicting bug report reproducibility?

Table 5 Performance of conventional classifiers on NetBeans project

Project	Classifier	Precision	Recall	F1-score	ROC
Netbeans	Bayes.BayesNet	77.4	82.4	77.2	72.4
	Bayes.NaiveBayes	77.5	82.3	77.7	71.4
	Bayes.NaiveBayesMultinomialText	82.5	100	80	50
	Bayes.NaiveBayesUpdateable	77.5	82.3	77.7	71.4
	Lazy.IBk	79.7	82.6	80.3	69
	Rules.DecisionTable	83.2	84.5	80.1	71.6
	Rules.Zero-R	82.5	100	80	50
	Rules.JRip	85	82.6	74.8	50.3
	Rules.OneR	77.5	82.5	75.3	50.9
	Rules.PART	81.6	84.2	**81.3**	72.9
	Trees.J48	83.7	84.4	79.7	69.4
	Trees.RepTree	82.8	84.5	80.7	71.7
	Trees.RandomForest	80.3	83	80.9	72.8

Table 6 Performance of conventional classifiers on eclipse project

Project	Classifier	Precision	Recall	F1-score	ROC
Eclipse	Bayes.BayesNet	77.3	83	78.1	76.2
	Bayes.NaiveBayes	78	82.1	79.2	75.9
	Bayes.NaiveBayesMultinomialText	83.7	100	80	50
	Bayes.NaiveBayesUpdateable	78	82.1	79.2	75.9
	Lazy.IBk	82.3	85	82.2	76.6
	Rules.DecisionTable	84	85.7	82.1	80.3
	Rules.Zero-R	83.7	100	80	50
	Rules.JRip	83.5	85.2	80.9	57.1
	Rules.OneR	70.1	83.7	76.3	50
	Rules.PART	83.7	85.7	**82.8**	80.7
	Trees.J48	84.4	85.9	82.3	77.8
	Trees.RepTree	84	85.8	82.5	80.3
	Trees.RandomForest	82.4	85	82.4	78.8

Table 7 Performance of conventional classifiers on mozilla firefox project

Project	Classifier	Precision	Recall	F1-score	ROC
Mozilla firefox	Bayes.BayesNet	84.9	84.8	84.8	91.6
	Bayes.NaiveBayes	83.3	82.5	82.6	90.6
	Bayes.NaiveBayesMultinomialText	56.8	100	72.4	50
	Bayes.NaiveBayesUpdateable	83.3	82.5	82.6	90.6
	Lazy.IBk	83.1	83.1	83.1	84.9
	Rules.DecisionTable	84	84	83.8	89.4
	Rules.Zero-R	56.8	100	72.4	50
	Rules.JRip	84	84	84	85.8
	Rules.OneR	79.3	77.6	77.7	78.6
	Rules.PART	85.5	85.6	**85.6**	90.4
	Trees.J48	85.2	85.2	85.2	89.6
	Trees.RepTree	84.5	84.6	84.5	89.6
	Trees.RandomForest	85.4	85.4	85.4	92

Table 8 Performance of bgging ensemble learning technique using conventional classifiers on netbeans project

Project	Classifier	Precision	Recall	F1-score	ROC
Netbeans	Bayes.BayesNet	77.5	82.4	77.2	72.4
	Bayes.NaiveBayes	77.5	82.2	77.6	71.3
	Bayes.NaiveBayesMultinomialText	82.5	100	80	50
	Bayes.NaiveBayesUpdateable	77.5	82.2	77.6	71.3
	Lazy.IBk	79.7	82.6	80.3	69
	Rules.DecisionTable	80.8	83.6	79	65.2
	Rules.Zero-R	82.5	100	80	50
	Rules.JRip	84.7	82.6	74.8	50.9
	Rules.OneR	77	82.5	75.2	52.1
	Rules.PART	82.2	85	82	77.4
	Trees.J48	83	84.7	80.9	71.2
	Trees.RepTree	82.7	84.7	81.3	74.2
	Trees.RandomForest	82.6	84.5	**83.1**	77.4

Table 9 Performance of bgging ensemble learning technique using conventional classifiers on eclipse project

Project	Classifier	Precision	Recall	F1-score	ROC
Eclipse	Bayes.BayesNet	77.3	83	78.2	76.1
	Bayes.NaiveBayes	78	82.1	79.2	75.9
	Bayes.NaiveBayesMultinomialText	83.7	100	80	50
	Bayes.NaiveBayesUpdateable	78	82.1	79.2	75.9
	Lazy.IBk	82.3	85	82.2	76.6
	Rules.DecisionTable	83.8	85.6	81.9	78.6
	Rules.Zero-R	83.7	100	80	50
	Rules.JRip	84	85.1	80.2	58.9
	Rules.OneR	83.7	83.7	83.7	50.1
	Rules.PART	84.1	87	84.2	82.5
	Trees.J48	84.3	86	82.7	79.3
	Trees.RepTree	83.9	85.8	82.8	81.4
	Trees.RandomForest	83.1	87.1	**84.4**	82.1

Table 10 Performance of bgging ensemble learning technique using conventional classifiers on mozilla firefox project

Project	Classifier	Precision	Recall	F1-score	ROC
Mozilla firefox	Bayes.BayesNet	84.9	84.8	84.8	91.6
	Bayes.NaiveBayes	83.3	82.5	82.6	90.6
	Bayes.NaiveBayesMultinomialText	56.8	100	72.4	50
	Bayes.NaiveBayesUpdateable	83.3	82.5	82.6	90.6
	Lazy.IBk	83.1	83.1	83.1	84.9
	Rules.DecisionTable	84.6	84.6	84.5	90.2
	Rules.Zero-R	56.8	100	72.4	50
	Rules.JRip	84.8	84.8	84.7	89.8
	Rules.OneR	79.3	77.6	77.7	78.6
	Rules.PART	86.5	86.5	86.5	92.8
	Trees.J48	86	86	86	91.9
	Trees.RepTree	85.4	85.4	85.4	91.9
	Trees.RandomForest	86.5	86.6	**86.6**	93

Table 11 Performance of boosting ensemble learning technique using conventional classifiers on netbeans project

Project	Classifier	Precision	Recall	F1-score	ROC
Netbeans	Bayes.BayesNet	77.4	82.4	77.2	69.2
	Bayes.NaiveBayes	77.5	82.3	77.7	68.5
	Bayes.NaiveBayesMultinomialText	82.5	100	80	50
	Bayes.NaiveBayesUpdateable	77.5	82.3	77.7	68.5
	Lazy.IBk	79.7	82.6	80.3	69
	Rules.DecisionTable	81.7	84.2	80.9	71.3
	Rules.Zero-R	82.5	100	80	50
	Rules.JRip	85.2	82.7	75.2	53
	Rrules.OneR	77.8	82.6	76.5	71.2
	Rules.PART	82.1	84.7	82.1	74
	Rrees.J48	80.9	83.4	81.4	72.6
	Rrees.RepTree	82.5	84.5	81	73.9
	trees.RandomForest	83.2	86.1	**84.8**	89.2

Table 12 Performance of boosting ensemble learning technique using conventional classifiers on eclipse project

Project	Classifier	Precision	Recall	F1-score	ROC
Eclipse	Bayes.BayesNet	80.4	83.5	81.2	77.4
	Bayes.NaiveBayes	80.6	83.5	81.4	77.4
	Bayes.NaiveBayesMultinomialText	83.7	100	80	50
	Bbayes.NaiveBayesUpdateable	80.6	83.5	81.4	77.4
	Lazy.IBk	82.3	85	82.2	76.6
	Rules.DecisionTable	83.5	85.6	82.6	80.4
	Rules.Zero-R	83.7	100	80	50
	Rules.JRip	82.7	85.2	81.6	79.9
	Rules.OneR	81.5	84	77.5	77.6
	Rules.PART	84	86.5	83.9	81.8
	Trees.J48	83.3	85.5	82.8	80.3
	Trees.RepTree	81.2	83.7	81.4	72.7
	Trees.RandomForest	84.9	87.8	**86.2**	84.1

6 Results and analysis

This section discusses the results obtained corresponding to four research questions addressed in this work.

RQ1: Performance of traditional machine learning approaches

Tables 5, 6 and 7 examines the performance of thirteen base classifiers: Bayes Net, Naive Bayes, Naive Bayes Multinomial Text, Naive Bayes Updateable, IBk, Decision Table, Zero-R, JRip, OneR, PART, J48, Rep Tree and Random Forest from four families: Bayes, lazy, rules and trees to find the best fixation prediction classifier. Various evaluation metrics such as Precision, Recall, F1-Score and ROC has been computed for comparison purposes. For F1-Score, the experimental findings show that the PART classifier surpasses all other classifiers. PART classifier scored the greatest F1-Score of 81.3%, 82.8%, and 85.6% on NetBeans, Eclipse, and Mozilla Firefox projects, respectively.

RQ2: performance of ensemble learning approaches

To address this RQ ensemble learning models have been created using three techniques: Bagging, Boosting and Stacking. Tables 8, 9 and 10 presents the results of bagging ensemble models using thirteen base classifiers on three considered projects. The experimental results reveal that Bagging using Random Forest algorithm performs better than other classifiers when considered F1-Score evaluation metric for comparison. The highest F1-Score of 83.1%, 84.4% and 86.6% was achieved by Bagging ensemble learners on NetBeans, Eclipse and Mozilla Firefox projects respectively. An improvement of 1.8%, 1.6% and 1% was achieved by Bagging models on NetBeans, Eclipse and Mozilla Firefox projects respectively as compared to best performing individual classifier performance (PART).

Tables 11, 12 and 13 presents the results of boosting ensemble models using thirteen base classifiers on three considered projects. The experimental results reveal that Boosting using Random Forest algorithm performs better than other classifiers when considered F1-Score evaluation metric

Table 13 Performance of boosting ensemble learning technique using conventional classifiers on mozilla firefox project

Project	Classifier	Precision	Recall	F1-score	ROC
Mozilla firefox	Bayes.BayesNet	84.8	84.5	84.7	91.3
	Bayes.NaiveBayes	84.5	84.5	84.5	89.5
	Bayes.NaiveBayesMultinomialText	56.8	100	72.4	50
	Bayes.NaiveBayesUpdateable	84.5	84.5	84.5	89.3
	Lazy.IBk	83.1	83.1	83.1	84.9
	Rules.DecisionTable	85.1	85.1	85.1	91.5
	Rules.Zero-R	56.8	100	72.4	50
	Rules.JRip	85.3	85.3	85.3	91.8
	Rules.OneR	79.3	77.6	77.7	78.6
	Rules.PART	85	85	85	90.9
	Trees.J48	84.8	84.9	84.8	91
	Trees.RepTree	85.2	85.2	85.2	89.7
	Trees.RandomForest	87.1	87.3	**87.2**	94.1

Table 14 Performance of ensemble learning techniques using conventional classifiers on three considered projects

Project	Technique	Precision	Recall	F1-score	ROC
Netbeans	BIC (Part)	81.6	84.2	81.3	72.9
	Bagging (random forest)	82.6	84.5	83.1	77.4
	Improvement with BIC	(+ 1%)	(+ 0.3%)	(+ 1.8%)	(+ 4.5%)
	Boosting (random forest)	83.2	86.1	84.8	89.2
	Improvement with BIC	(+ 1.6%)	(+ 1.9%)	(+ 3.5%)	(+ 16.3%)
	Stacking (PART, random forest, reptree, Naive Bayes, J48)	85.5	87.2	86.1	83.2
	Improvement with BIC	(+ 3.9%)	(+ 3%)	(+ 4.8%)	(+ 10.3%)
Eclipse	BIC (Part)	83.7	85.7	82.8	80.7
	Bagging (random forest)	83.1	87.1	84.4	82.1
	Improvement with BIC	(−0.6%)	(+ 1.4%)	(+ 1.6%)	(+ 1.4%)
	Boosting (random forest)	84.9	87.8	86.2	84.1
	Improvement with BIC	(+ 1.2%)	(+ 2.1%)	(+ 3.4%)	(+ 3.4%)
	Stacking (Part, reptree, JRip, Naive Bayes, decision table)	84.4	86.3	85.1	84.2
	Improvement with BIC	(+ 0.7%)	(+ 0.6%)	(+ 2.3%)	(+ 3.5%)
Mozilla firefox	BIC (part)	85.5	85.6	85.6	90.4
	Bagging (random forest)	86.5	86.6	86.6	93
	Improvement with BIC	(+ 1%)	(+ 1%)	(+ 1%)	(+ 2.6%)
	Boosting (random forest)	87.1	87.3	87.2	94.1
	Improvement with BIC	(+ 1.6%)	(+ 1.7%)	(+ 1.6%)	(+ 3.7%)
	Stacking (PART, reptree, Naive Bayes, decision table)	87.6	87.2	87.5	94.8
	Improvement with BIC	(+ 2.1%)	(+ 1.6%)	(+ 1.9%)	(+ 4.4%)

BIC represents best individual classifier

for comparison. The highest F1-Score of 84.8%, 86.2% and 87.2% was achieved by Boosting on NetBeans, Eclipse and Mozilla Firefox proj- ects respectively. An improvement of 3.5%, 3.4% and 1.6% was achieved by Boosting models on NetBeans, Eclipse and Mozilla Firefox projects respectively as compared to best performing individual classifier performance (PART).

For stacking ensemble learning technique, all possible combinations of thirteen base classifiers have been used to learn the models. The experiments have been initialized from size three of base classifiers. In total, 8100 different model combinations corresponding to each considered project have been learned. Logistic Regression has been used as meta classifier. The performance and improvement of best model combination corresponding to each project has been reported in Table 14. The highest F1-Score of 86.1%, 85.1% & 87.5% was achieved by Stacking ensemble learning technique on NetBeans, Eclipse and Mozilla Firefox projects

Table 15 Performance of ensemble learning techniques with feature selection using conventional classifiers on three considered projects

Project	Technique	Precision	Recall	F1-score	ROC
Netbeans	BIC (Part)	81.6	84.2	81.3	72.9
	Bagging (random forest)	82.6	84.5	83.1	77.4
	Improvement with BIC	(+ 1%)	(+ 0.3%)	(+ 1.8%)	(+ 4.5%)
	Boosting (random forest)	83.2	86.1	84.8	89.2
	Improvement with BIC	(+ 1.6%)	(+ 1.9%)	(+ 3.5%)	(+ 16.3%)
	Stacking (part, random forest, RepTree, Naive Bayes, J48)	87.7	87.1	87.4	86.5
	Improvement with BIC	(+ 6.1%)	(+ 2.9%)	(+ 6.1%)	(+ 13.6%)
Eclipse	BIC (Part)	83.7	85.7	82.8	80.7
	Bagging (random forest)	83.1	87.1	84.4	82.1
	Improvement with BIC	(−0.6%)	(+ 1.4%)	(+ 1.6%)	(+ 1.4%)
	Boosting (random forest)	84.9	87.8	86.2	84.1
	Improvement with BIC	(+ 1.2%)	(+ 2.1%)	(+ 3.4%)	(+ 3.4%)
	Stacking (Part, rep tree, JRip, Naive bayes, decision table)	86.2	88.4	87.8	85.7
	Improvement with BIC	(+ 2.5%)	(+ 2.7%)	(+ 5%)	(+ 5%)
Mozilla firefox	BIC (PART)	85.5	85.6	85.6	90.4
	Bagging (random forest)	86.5	86.6	86.6	93
	Improvement with BIC	(+ 1%)	(+ 1%)	(+ 1%)	(+2.6%)
	Boosting (random forest)	87.1	87.3	87.2	94.1
	Improvement with BIC	(+ 1.6%)	(+ 1.7%)	(+ 1.6%)	(+ 3.7%)
	Stacking (Part, rep tree, Naive Bayes, decision table)	87.9	88.5	88.3	95.7
	Improvement with BIC	(+ 2.4%)	(+ 2.9%)	(+ 2.7%)	(+ 5.3%)

BIC represents best individual classifier

respectively. An improvement of 4.8%, 2.3% and 1.9% was achieved by Stacking technique on NetBeans, Eclipse and Mozilla Firefox projects respectively as compared to best performing individual classifier performance (PART). For NetBeans project, stacking with base classifier combination of PART, Random Forest, REPTREE, Naive Bayes and J48 obtained best performance. For Eclipse project, stacking with base classifier having combination of PART, REPTREE, JRip, Naive Bayes and Decision Table obtained best performance. For Mozilla Firefox project, stacking with base classifier combination of PART, REPTREE, Naive Bayes and Decision Table obtained best performance.

RQ3: performance of ensemble learning approaches with feature selection

To address this RQ, Classifier Attribute Evaluator has been used for feature selection before applying ensemble learning techniques for prediction. Table 15 summarizes the performance and improvement of various ensemble learning algorithms along with feature selection technique. The highest F1-Score of 83.1%, 84.4% and 86.6% was achieved by Bagging on NetBeans, Eclipse and Mozilla Firefox projects respectively. The highest F1-Score of 84.8%, 86.2% and 87.2% was achieved by Boosting on NetBeans, Eclipse and Mozilla Firefox projects respectively. The highest F1-Score of 87.4%, 87.8% and 88.3% was achieved by Stacking on NetBeans, Eclipse and Mozilla Firefox projects respectively. An improvement of 6.1%, 5% & 2.7% was achieved by Stacking on NetBeans, Eclipse and Mozilla Firefox projects respectively as compared to best performing individual classifier performance.

7 Threats to validity

Despite the fact that the experiments were structured in such a way that there are few risks to validity, there are still a number of decisions that might impact the findings of this paper. Various dangers to the validity of reported work are examined in this section.

7.1 External validity

The generalizability of generated outcomes is referred to as external validity. The bug reports included in this work's experimental assessments came from three open-source Bugzilla repository projects: NetBeans, Eclipse, and Mozilla Firefox. Bug reports from these projects may differ from those from other open-source and closed-source projects. As a result, the findings of the paper might not apply to other open-source and commercial software projects. Other open-source and closed-source projects, as well as those that employ other development approaches, will require more

research. Despite the fact that big open-source initiatives covering a wide range of topics have been investigated, there may be additional projects adopting diverse software techniques in the future. As a result, the conclusions may not apply to everyone.

7.2 Internal validity

The bias and mistakes in the experimental setting are referred to as internal validity. The data acquired from a bug repository was deemed to be ideal in this paper. However, there is a chance that the extracted data contains mistakes or noise, which might impact the paper's conclusions. To counteract this risk, the dataset included in this work has been drawn from widely used projects if Bugzilla repository. These projects have a lengthy history and are continuously maintained, therefore the retrieved data should be considered acceptable (if not optimal). The input parameters used to train diverse machine learning models pose another danger to internal validity. In this paper, nine bug parameters were employed as input features for model training (eight numerical and one textual parameter). However, there may be an alternative collection of qualities that will produce superior results. However, there may be an other collection of characteristics that perform better for predicting NR fixability. A list of stop words offered by the Python NLTK toolkit was utilised to pre-process textual contents (http://www.nltk.org/). The Porter Stemmer tool from the Python NLTK toolbox was used to stem textual contents. For comparable procedures, this toolbox has been frequently utilised in the literature. Other stop word lists and stemming tools, on the other hand, may have an impact on prediction accuracy. To reduce the risk of code and experimental setup problems, the source code and experimental setup have been double-checked. However, there is still the risk of mistakes. In the experiments, 10 fold cross validation was utilised to eliminate bias.

7.3 Construct validity

The experimental constructs or the adequacy of the assessment measures utilised in the study are referred to as construct validity. The F1-Score was reported in this paper. This measure has been extensively used in the literature to assess the performance of machine learning classifiers, hence there is no concern about construct validity in this paper.

8 Conclusion

The bug management is onerous task for software engineers due to the unpredictable nature of bug fixes. The complexity of this perplexing indexing is exacerbated by non-reproducible faults. To address Non-reproducible bugs, a novel fixability prediction framework named NRPredictor is proposed in this paper. Thirteen traditional machine learning classifiers along with three ensemble learning approaches (Bagging, Boosting, and Stacking) and one feature selection technique have been leveraged in NRPredictor. The experimental evaluation shows that traditional machine learning algorithm, PART scored the greatest F1-Score of 81.3%, 82.8% and 85.6% on NetBeans, Eclipse, and Mozilla Firefox projects, respectively. Ensemble learning techniques outperforms traditional machine learning approaches, achieving F1-Scores of up to 86.1, 85.1, and 87.5% for NetBeans, Eclipse, and Mozilla Firefox applications, respectively. Feature selection with ensemble learning techniques achieves F1-Scores of up to 87.4, 87.8, and 88.3% for NetBeans, Eclipse, and Mozilla Firefox applications, respectively.

9 Future research directions

The performance of proposed framework, NRPredictor, may be investigated in future on closed-source applications. Collaboration with firms that use open-source and closed-source bug repositories to analyse the proposed framework, NRPredictor, in an industrial context is also possible. This will aid in further generalisation of findings. Text mining techniques such as topic modelling may also be used and integrated into the framework. In addition, a fix recommendation tool may be developed, which provide tokens for non-reproducible issues that might be solved. Another area of future study is the creation of a tool for software developers to aid in the prediction of NR bugs. Although the present study aids in the prediction of difficult-to-reproduce bugs that can be labelled as NR. However, NR issues continue to offer a significant barrier to the bug-fixing process; as a result, new ways for resolving NR-marked bug reports can be created.

Funding This study and all authors involved in this study have not received any funding, including after the completion of the study.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article. The authors did not receive support from any organization for the submitted work.

Human or animal rights This tudy did not involve any human participants or animals.

References

Abou Khalil Z, Constantinou E, Mens T, Duchien L (2021) On the impact of release policies on bug handling activity: a case study of eclipse. J Syst Softw 173:110882

Abualigah L, Abd Elaziz M, Sumari P, Geem ZW, Gandomi AH (2022) Reptile search algorithm (RSA): a nature-inspired meta-heuristic optimizer. Exp. Syst. Appl. 191:116158

Abualigah L, Diabat A, Mirjalili S, Abd Elaziz M, Gandomi AH (2021) The arithmetic optimization algorithm. Comput. Methods Appl. Mech. Eng. 376:113609

Abualigah L, Yousri D, Abd Elaziz M, Ewees AA, Al-Qaness MA, Gandomi AH (2021) Aquila optimizer: a novel meta-heuristic optimization algorithm. Comput. Indus. Eng. 157:107250

Agushaka JO, Ezugwu AE, Abualigah L (2022) Dwarf mongoose optimization algorithm. Comput. Methods Appl. Mech. Eng. 391:114570

Ahmed HA, Bawany NZ, Shamsi JA (2021) CaPBug-a framework for automatic bug categorization and prioritization using NLP and machine learning algorithms. IEEE Access 9:50496–50512

Alzubi JA (2015) Optimal classifier ensemble design based on cooperative game theory. Res J Appl Sci Eng Technol 11(12):1336–1343

Alzubi OA, Alzubi JA, Alweshah M, Qiqieh I, Al-Shami S, Ramachandran M (2020) An optimal pruning algorithm of classifier ensembles: dynamic programming approach. Neural Comput Appl 32(20):16091–16107

Alzubi OA, Alzubi JAA, Tedmori S, Rashaideh H, Almomani O (2018) Consensus-based combining method for classifier ensembles. Int Arab J Inf Technol 15(1):76–86

Anvik J (2006) Automating bug report assignment. In: Proceedings of the 28th International conference on software engineering, pp 937–940

Anvik J, Murphy GC (2011) Reducing the effort of bug report triage: recommenders for development oriented decisions. ACM Trans Softw Eng Method (TOSEM) 20(3):10

Artzi S, Kim S, Ernst MD (2008) Recrash: Making software failures reproducible by preserving object states. In: European conference on object-oriented programming. pp 542–565. Springer

Bauer C, Parkinson A, Scharl A (1999) Automated vs. manual classification: a multi-methodological set of web analysis components. In: Australasian conference on information systems, pp 54–64. Citeseer

Bhattacharya P, Neamtiu I (2010) Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In: Software maintenance (ICSM), 2010 IEEE international conference on, pp 1–10. IEEE

Breiman L (1996) Bagging predictors. Machine Learn 24(2):123–140

Breu S, Premraj R, Sillito J, Zimmermann T (2010) Information needs in bug reports: improving cooperation between developers and users. In: Proceedings of the 2010 ACM conference on computer supported cooperative work, pp 301–310

Chaparro O, Lu J, Zampetti F, Moreno L, Di Penta M, Marcus A, Bavota G, and Ng V (2017) Detecting missing information in bug descriptions. In Proceedings of the 2017 11th joint meeting on foundations of software engineering, pp 396–407. ACM

Cheng X, Liu N, Guo L, Xu Z, Zhang T (2020) Blocking bug prediction based on XGBoost with enhanced features. In: 2020 IEEE 44th annual computers, software, and applications conference (COMPSAC), pp 902–911. IEEE

Dao A-H, Yang C-Z (2021) Improving priority prediction for bug reports with comment features. In 2021 IEEE international conference on software engineering and artificial intelligence (SEAI), pp 58–62. IEEE

Dietterich TG (2000) Ensemble methods in machine learning. In: International workshop on multiple classifier systems, pp 1–15. Springer

Ekanayake J (2021) Bug severity prediction using keywords in imbalanced learning environment. Int J Inf Technol Comput Sci (IJITCS) 13:53–60

Erfani Joorabchi M, Mirzaaghaei M, Mesbah A (2014) Works for me! characterizing non-reproducible bug reports. In Proceedings of the 11th working conference on mining software repositories, pp 62–71. ACM

Fagan M (2002) Design and code inspections to reduce errors in program development. In Software pioneers, pp 575–607. Springer

Fan Y, Xia X, Lo D, Hassan AE (2018) Chaff from the wheat: characterizing and determining valid bug reports. IEEE Trans Softw Eng 46(5):495–525

Freund Y, Schapire RE (1995) A decision-theoretic generalization of on-line learning and an application to boosting. In: European conference on computational learning theory, pp 23–37. Springer

Goyal A, Sardana N (2016) Analytical study on bug triaging practices. Int J Open Source Softw Process (IJOSSP) 7(2):20–42

Goyal A, Sardana N (2017) black optimizing bug report assignment using multi criteria decision making technique. Intell Decision Technol 11(3):307–320

Goyal A, Sardana N (2017) Machine learning or information retrieval techniques for bug triaging: Which is better? e-Inform Softw Eng J. https://doi.org/10.5277/e-Inf170106

Goyal A, Sardana N (2017) Nrfixer: sentiment based model for predicting the fixability of non-reproducible bugs. Inform Softw Eng J 11(1):109–122

Goyal A, Sardana N (2018) Characterization study of developers in non-reproducible bugs. In: 2018 eleventh international conference on contemporary computing (IC3), pp 1–6. IEEE

Goyal A Sardana N (2019) Empirical analysis of ensemble machine learning techniques for bug triaging. In: 2019 twelfth international conference on contemporary computing (IC3), pp 1–6. IEEE

Guo PJ, Zimmermann T, Nagappan N, Murphy B (2010) Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In: Software engineering, 2010 ACM/IEEE 32nd international conference on, vol 1, pp 495–504. IEEE

X. Guo, Y. Yin, C. Dong, G. Yang, and G. Zhou (2008) On the class imbalance problem. In: Fourth international conference on natural computation, 2008. ICNC'08., vol 4, pp 192–201. IEEE

Gupta S, Gupta SK (2021) An approach to generate the bug report summaries using two-level feature extraction. Exp Syst Appl 176:114816

Hewett R, Kijsanayothin P (2009) On modeling software defect repair time. Empir Softw Eng 14(2):165

Isotani H, Washizaki H, Fukazawa Y, Nomoto T, Ouji S, Saito S (2021) Duplicate bug report detection by using sentence embedding and fine-tuning. In: 2021 IEEE international conference on software maintenance and evolution (ICSME), pp 535–544. IEEE

Jin W, Orso A (2012) Bugredux: reproducing field failures for in-house debugging. In: 2012 34th international conference on software engineering (ICSE), pp 474–484. IEEE

Jonsson L, Borg M, Broman D, Sandahl K, Eldh S, Runeson P (2016) Automated bug assignment: ensemble-based machine learning in large scale industrial contexts. Empir Softw Eng 21(4):1533–1578

Kamkar M (1998) Application of program slicing in algorithmic debugging. Inform Softw Technol 40(11–12):637–645

Koh Y, Kang S, Lee S (2021) Bug report summarization using believability score and text ranking. In: 2021 international conference on artificial intelligence in information and communication (ICAIIC), pp 117–120. IEEE

Kumari M, Sharma M, Anand S, Singh V (2020) Predicting the fix time of a reported bug using radoop: a big data approach. In: Decision analytics applications in industry, pp 259–269. Springer

Kumari M, Singh UK, Sharma M (2020) Entropy based machine learning models for software bug severity assessment in cross project

context. In: International conference on computational science and its applications, pp 939–953. Springer
Lal S, Sardana N, Sureka A (2017) Eclogger: cross-project catch-block logging prediction using ensemble of classifiers. e-Inform Softw Eng J. https://doi.org/10.5277/e-Inf170101
Laradji IH, Alshayeb M, Ghouti L (2015) Software defect prediction using ensemble learning on selected features. Inform Softw Technol 58:388–402
Lee Y, Lee S, Lee C-G, Yeom I, Woo H (2020) Continual prediction of bug-fix time using deep learning-based activity stream embedding. IEEE Access 8:10503–10515
Li Z, Jiang Z, Chen X, Cao K, Gu Q (2021) Laprob: a label propagation-based software bug localization method. Inform Softw Technol 130:106410
Limsettho N, Bennin KE, Keung JW, Hata H, Matsumoto K (2018) Cross project defect prediction using class distribution estimation and oversampling. Inform Softw Technol 100:87–102
López V, Fernández A, García S, Palade V, Herrera F (2013) An insight into classification with imbalanced data: empirical results and current trends on using data intrinsic characteristics. Inform Sci 250:113–141
Malhotra R, Dabas A, Hariharasudhan A, Pant M (2021) A study on machine learning applied to software bug priority prediction. In: 2021 11th international conference on cloud computing, data science & engineering (Confluence), pp 965–970. IEEE
Mohan D, Goyal A, Sardana N (2016) Visheshagya: Time based expertise model for bug report assignment. In: 2016 ninth international conference on contemporary computing (IC3), pp 1–6. IEEE
Movassagh AA, Alzubi JA, Gheisari M, Rahimi M, Mohan S, Abbasi AA, Nabipour N (2021) Artificial neural networks training algorithm integrating invasive weed optimization with differential evolutionary model. J Ambient Intel Human Comput. https://doi.org/10.1007/s12652-020-02623-6
Neysiani BS, Babamir SM (2020) Automatic duplicate bug report detection using information retrieval-based versus machine learning-based approaches. In: 2020 6th international conference on web research (ICWR), pp 288–293. IEEE
Neysiani BS, Babamir SM, Aritsugi M (2020) Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems. Inform Softw Technol 126:106344
Oyelade ON, Ezugwu AE-S, Mohamed TI, Abualigah L (2022) Ebola optimization search algorithm: a new nature-inspired metaheuristic optimization algorithm. IEEE Access 10:16150–16177
Perry DE, Stieg CS (1993) Software faults in evolving a large, real-time system: a case study. In: European software engineering conference, pp 48–67. Springer
Phua C, Alahakoon D, Lee V (2004) Minority report in fraud detection: classification of skewed data. Acm Sigkdd Explorations Newsletter 6(1):50–59
Rahman MM, Khomh F, Castelluccio M (2020) Why are some bugs non-reproducible?: An empirical investigation using data fusion. In: 2020 IEEE international conference on software maintenance and evolution (ICSME), pp 605–616. IEEE
Rashmi P, Kambli P (2020) Predicting bug in a software using ann based machine learning techniques. In: 2020 IEEE International Conference for Innovation in Technology (INOCON), pp 1–5. IEEE
Rocha TM, Carvalho ALDC (2021) SiameseQAT: a semantic context-based duplicate bug report detection using replicated cluster information. IEEE Access 9:44610–44630
Sethuraman J, Alzubi JA, Manikandan R, Gheisari M, Kumar A (2019) Eccentric methodology with optimization to unearth hidden facts of search engine result pages. Recent Patents Comput Sci 12(2):110–119
Sharma M, Kumari M, Singh V (2021) Bug priority assessment in cross-project context using entropy-based measure. In: Advances in machine learning and computational intelligence, pp 113–128. Springer
Shatnawi MQ, Alazzam B (2022) An assessment of eclipse bugs' priority and severity prediction using machine learning. Int J Commun Netwo Inform Security (IJCNIS) 14(1):62–69
Shihab E, Ihara A, Kamei Y, Ibrahim WM, Ohira M, Adams B, Hassan AE, Matsumoto K-I (2013) Studying re-opened bugs in open source software. Empir Softw Eng 18(5):1005–1042
Shokripour R, Anvik J, Kasirun ZM, Zamani S (2015) A time-based approach to automatic bug report assignment. J Syst Softw 102:109–122
Sureka A, Jalote P (2010) Detecting duplicate bug report using character n-gram-based features. In: 2010 17th Asia pacific software engineering conference (APSEC), pp 366–374. IEEE
Tagra A (2021) Studying reopened bugs in open source software systems. PhD thesis
Tamrawi A, Nguyen TT, Al-Kofahi J, Nguyen TN (2011) Fuzzy set-based automatic bug triaging: nier track. In: Software engineering (ICSE), 2011 33rd international conference on, pp 884–887. IEEE
Tamrawi A, Nguyen TT, Al-Kofahi JM, Nguyen TN (2011) Fuzzy set and cache-based approach for bug triaging. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, pp 365–375
Valdivia Garcia H, Shihab E (2014) Characterizing and predicting blocking bugs in open source projects. In: Proceedings of the 11th working conference on mining software repositories, pp 72–81. ACM
White M, Linares-Vásquez M, Johnson P, Bernal-Cárdenas C, Poshyvanyk D (2015) Generating reproducible and replayable bug reports from android application crashes. In: 2015 IEEE 23rd international conference on program comprehension (ICPC), pp 48–59. IEEE
Wolpert DH (1992) Stacked generalization. Neural Netw 5(2):241–259
Xi S-Q, Yao Y, Xiao X-S, Xu F, Lv J (2019) Bug triaging based on tossing sequence modeling. J Comput Sc Technol 34(5):942–956
Xia X, Lo D, Shihab E, Wang X, Yang X (2015) Elblocker: predicting blocking bugs with ensemble imbalance learning. Inform Softw Technol 61:93–106
Ye L, Jinxiao H, Yutao M (2020) An automatic method using hybrid neural networks and attention mechanism for software bug triaging. J Comput Res Develop 57(3):461
Yuan W, Xiong Y, Sun H, and Liu X (2021) Incorporating multiple features to predict bug fixing time with neural networks. In: 2021 IEEE international conference on software maintenance and evolution (ICSME), pp 93–103. IEEE
Zhang T, Yang G, Lee B, Lua EK (2014) A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In: 2014 21st Asia-pacific software engineering conference, vol 1, pp 223–230. IEEE
Zhou T, Sun X, Xia X, Li B, Chen X (2019) Improving defect prediction with deep forest. Inform Softw Technol 114:204–216
Zimmermann T, Nagappan N, Guo PJ, Murphy B (2012) Characterizing and predicting which bugs get reopened. In: Proceedings of the 34th international conference on software engineering, pp 1074–1083. IEEE Press