Taylor & Francis
Taylor & Francis Group

Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/taut20

Automated program and software defect root cause analysis using machine learning techniques

C. Anjali, Julia Punitha Malar Dhas & J. Amar Pratap Singh

To cite this article: C. Anjali, Julia Punitha Malar Dhas & J. Amar Pratap Singh (2023) Automated program and software defect root cause analysis using machine learning techniques, Automatika, 64:4, 878-885, DOI: 10.1080/00051144.2023.2225344

To link to this article: https://doi.org/10.1080/00051144.2023.2225344

Published online: 27 Jun 2023.

AUTOMATIKA
2023, VOL. 64, NO. 4, 878–885
https://doi.org/10.1080/00051144.2023.2225344

Automated program and software defect root cause analysis using machine learning techniques

C. Anjali[a], Julia Punitha Malar Dhas[b] and J. Amar Pratap Singh[a]

[a]Department of CSE, Noorul Islam Centre For Higher Education, Kumaracoil, Tamilnadu, India; [b]Department of CSE, Karunya Institute of Technology and Sciences, Coimbatore, Tamilnadu, India

ABSTRACT

For the automated root cause analysis (ARCA) method and simplified RCA technique, their empirical assessment is presented in this study. A focus group meeting is a foundation for the target problem identification in the ARCA technique. This is compared to earlier RCA methodologies which rely on problem sampling for target problem discovery and high beginning costs. In this research, we suggest a naïve Bayes based machine learning method for identifying the underlying causes of newly reported software issues, which will facilitate a quicker and more effective resolution of software bugs. The ARCA technique produced a large number of high-quality corrective actions while requiring a reasonable amount of effort. The strategy is an effective way to find new opportunities for process improvement and produce fresh process improvement ideas in contrast to the organization's corporate practices. In addition it is simple to utilize. Ultimately, we compared the methodology with other machine learning classifiers including support vector machine and decision tree.

ARTICLE HISTORY
Received 23 April 2023
Accepted 9 June 2023

KEYWORDS
Software defect prediction (SDP); machine learning; RCA; problem prevention; naïve Bayes

1. Introduction

Many software process improvement approaches, including CMMI, ISO/IEC 12207, and Six Sigma take problem sources into account. Understanding the cause of the problem is essential for successful problem prevention. We think the fundamental reason behind this is that the only way to avoid the issue from recurring is to get rid of its root causes. An organized investigation issue to determine which underlying causes need to be corrected is known as root cause analysis (RCA). It can help with process improvement and address this issue in a variety of situations and across all software organizations, including product design, embedded system, advanced manufacturing, and assembly. The lightweight RCA approach and its actual assessment are presented in this work. We created an RCA technique known as the ARCA method for our research and tested it. The setting of using root analysis techniques software design served the study's environment. The business requirement was to create a simple RCA approach that was used for problem avoidance at medium-sized software firms. A root cause analysis literature research served to develop the knowledge base for the technical design. Four medium-sized software product firms' target challenges served as the test cases for the ARCA approach, which is used to evaluate the design through industrial studies.

Adding new features removing outmoded ones, upgrading the idea or product, streamlining the code base's structure using coding standards, and enhancing scalability are all examples of active software development processes that increase the value of already-existing software. This can lead the institutions to spend a lot of money and time in defect prediction and management based on an appropriate model. The development and use of a process performance model (PPM), which considers the defect prediction model proves itself to be one of the high maturity practices in the software. These models help us in estimating software reliability means including the average initial error content, the average interval of time within failures, the average number of still existing errors at an imaginary testing time point, and the software reliability function.

The number of bugs in large software systems can increase significantly. It is a standard procedure for someone testing software to run into a functional bug and report it to the system. The developer then resolves the bug after receiving it from the assigner. However, these problems typically take time to manifest. It frequently occurs that a developer who has not worked on the root cause module or the functionality behind the bug is mistakenly assigned to fix it. This is due to the fact that recently reported bugs occasionally only include the bug description and stack trace. As a result,

CONTACT C. Anjali anjali.c@mbcet.ac.in Department of CSE, Noorul Islam Centre for Higher Education, Kumaracoil, India

it becomes quite challenging for the assigner to choose the appropriate person to fix the fault. A supervised machine learning technique is utilized to forecast the root cause (RC) of the issues in order to address the aforementioned issue.

Defect modules can be predicted to improve software quality. Designing models for defect prediction is a technique used in the early stages of the process to find flawed systems, such as units or classes. By categorizing the modules as defect-prone or not, this can be accomplished. The classification module is determined using a variety of techniques, the most popular being support vector classifier (SVC), random forest, naïve Bayes, decision trees (DT), and neural networks (NN). During progress testing phases, the identified defect-prone modules are given top priority and the non-defect-prone modules. The main contribution of the work is as follows:

- Design a naïve Bayes based machine learning method for identifying the underlying causes of newly reported software issues, which will facilitate a quicker and more effective resolution of software bugs.
- Categorizing the software dataset as a model for bug prediction into a defective and non-defective dataset.
- Evaluate the performance metrics such as accuracy, precision, recall, and specificity and compared with other machine learning algorithms.

The remainder of the paper is structured as follows: related work of the systems is discussed in Section 2. Section 3 contains information about techniques used to obtain better results. The results of all analysed techniques are included in Section 4, along with a comparison of evaluation measures and analysed methods to earlier approaches. Section 5 concludes the work.

2. Related work

This article compares five machine learning algorithms support vector regression (SVR), using the metrics generated from entropy of changes. Web service and regular expressions are used to automate data extraction for validation purposes [1]. Taking the advantage of the relationship between models explains ability and predictive power. We offer a straightforward model sampling method for identifying predictive results with the smallest number of features. Our main premise is that features that do not contribute to the model's predictive capacity should not be included [2]. The automatic bug identification schemes evaluate the outcomes from a set of controlled experiments. We demonstrate the performance of individual classifiers for defect attribution and is scalable to large-scale industrial applications, with an accuracy of the model ranging from 60% to 85% using large-scale training sets. Finally, while analysing automated bug assignment, we emphasize the necessity of not only relying on cross-validation results [3]. Software defect prediction helps engineers enhance software security by locating the problematic code. Existing research on software defect prediction, on the other hand, is primarily limited to the source code. Previous research has never looked into prediction models for Android binary executables. We offer an exploratory investigation of problem prediction in Android apps [4].

AutoODC was instructed using two cutting-edge text different classifiers, naïve Bayes (NB) it reaches general accuracy rate of 82.9% (NB) and 80.7% (SVM) on the advanced manufacturing fault report, and 77.5% (NB) and 75.2% (SVM) on the greater, more diverse open source defect list [5]. Our suggested model intends to forecast new faults created in the latest version by objectively and formally examining the sorts of changes and taking into account changes in lines of code (LOC) [6]. The research's key findings are that (a) the use of algorithms to speed up time-consuming tasks in developing software like debugging and supporting documents and (b) the structured analysis of large data sets to identify trends and new communications clusters are two AI's major accomplishments and future potentials [7]. A literature evaluation is also conducted in order to ensure needs management and to identify potential gaps in future research directions in order to improve our understanding. Experts share their perspectives on the main reason of the absence of security focus in requirement analysis [8]. This study suggests mining bug repositories using deep-learning models and clustering techniques to determine the risk level of recently reported issues. We explored a variety of models because they all had unique qualities; it can extract features from input and condense the feature space [9]. Based on 33 research topics, we conducted a thorough mapping study on MLS testing procedures. To maximize the consistency and dependability of our findings, we used current recommendations when developing our research process [10].

They introduce a machine learning-based device that effectively provides insight into mistake rates as well as process improvement and recovery procedures into the design phase, removing the requirement for manual computation [11]. In this study, we present a prediction system that takes information from earlier issue submissions and suggests the most aspects for brand-new issues. Our approach uses a deep-learning technique called long short-term memory to automatically extract semantic elements that reflect a problem report and combine them with traditional textual similarity data [12]. This research presents that a rate is usually a deep-learning structure for the classification and extraction of boundary irregularities, which

is demonstrated using surface fracture detection in a particular location. The suggested approach is consistent with various deep-learning approaches, including existing software packages, and has been shown that in the domain of outer edge recognition, the suggested technique has similar methods [13].

Bayesian networks are used to determine the probability impact linkages between evaluation metrics and issue tendency. In addition to the metrics used in the Promise data store, we define two new indicators, NOD for the team of developers and LOCQ for system software quality. The Promised data repository's source code for the data sets is selected. We may accept these metrics [14]. Regarding organizations that do not track defect-related data, we provide a feasible defect prediction approach. We specifically look into the use of cross-company (CC) data for developing localized defect predictors based on static code properties. To begin, we examine the circumstances in which CC data can be used directly. There are only a few of these circumstances [15].

3. Proposed methodology

The methods used for the RCA of the eclipse bugs described above are given. In addition to being grammatically relevant and intuitive, our goal was to arrive at a system that produced good outcomes based on machine learning measures like accuracy and precision. Supervised learning appears to be the solution that best fits the issue previously stated. However, given the limitations of the data that is currently accessible, the semi-supervised technique performs well in the majority of real-world circumstances.

3.1. Machine learning methods for defect root cause analysis

RCA is used to identify and classify the underlying causes of incidents that have effects on production, quality, dependability, safety, and the environment. Investigators will not be able to pinpoint and implement remedial measures to stop similar occurrences in the future until they can establish why an event or failure occurred.

The work presented here is based on earlier studies. The studies show a major difference in terms of the proposed modelling technique, software metrics used, and the verification technique used. The existing models such as SVM, k nearest, and random forest are used in different modelling techniques. These models use function level, and field and file level or package level granularity for prediction. Classification or ranking techniques are used for the purpose of validation. The term "fault" or "defect" or "bug" is used to represent defect in the source code. When the software fails to perform the function specified, the failure occurs

3.1.1. Root cause detection

There are various techniques for gathering and organizing the target problem causes in root cause detection. Utilizing techniques like questionnaires, brainwriting, interviewing, and brainstorming, the causes are typically gathered from various partners. In comparison to the discussion and brain writing processes, which are carried out in front of others, the surveys and interviews are more private methods. The objective problem causes are typically arranged into an occurrence diagram using a diagram, failure mode effect diagram, reason map, matrix diagram, dispersion chart, reasoning tree, or cause element chart depending on their cause-and-effect correlations. Listings, checklists, and charts have been demonstrated to be effective tools for organizing the causes. By examining the gathered implement program causes and concentrating on the sources that can be avoided; the root causes are ultimately found.

- Naïve Bayes
- Maximum entropy
- Decision trees
- Support vector machine
- Safety-based RCA

Safety-based as the name implies, RCA is an analysis carried out on a single premise of safety. These precautions might be categorized as workplace safety, health safety, etc. In this, the RCA method is used to detect or examine problems if any problems or events arise in the fields of safety, health, etc. in the absence of safety barriers, potential risks, and damages. It may be the root cause of this RCA.

Production-based RCA

Production-based RCA is a type of analysis that is based on production. This production might be viewed as quality assurance for businesses involved in product manufacture. In this, the RCA technique is utilized to examine or research any problems that may arise in the manufacturing field. The root causes in this RCA could include a function failure, a manufacturing line error, etc.

3.1.1.1. Process-based RCA. Production-based acronym RCA stands for one basis of process analysis. In this, the RCA technique is utilized to examine or explore any problems that may arise in the field development process, which includes the processes of enhancing design, product management, and project management. The root causes of this RCA could be a failure in an individual's procedure, an error in a process step, etc.

3.1.1.2. Failure-based RCA. Failure-based as its name implies, RCA is an investigation of a single cause of

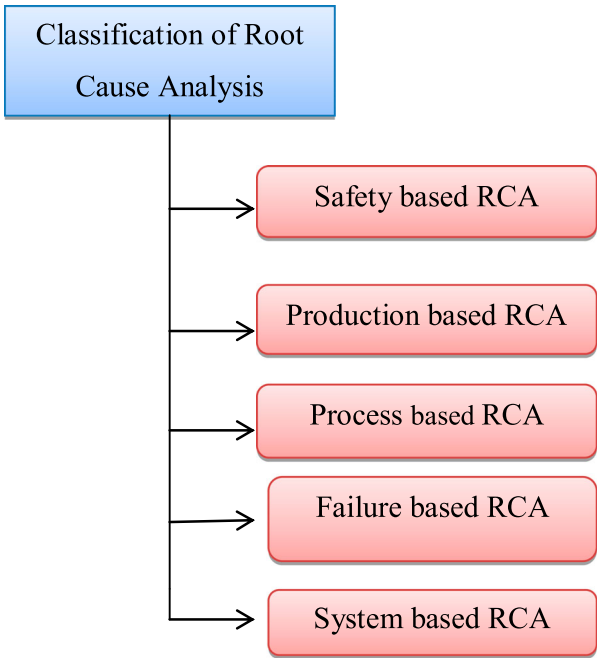

Figure 1. Proposed classification of RCA.

failure. RCA techniques aims to predict the estimated amount of flaws in a particular method or software segment and also it is a technique used to classify the module . For the purpose of defect prediction, practitioners use a number of different techniques and can be broadly categorized into (i) defect prediction and (ii) defect classification. Defect predictions is a technique for estimating the amount of defects that will be detected in a software package. Figure 1 shows how to classify problems using important software root cause defect analysis algorithms.

Comparison metrics: The defect reports (data sets) are divided into specific groups using multi-class classification, and the detection accuracy is evaluated using highest accuracy results.

3.2. Concept of association rule mining and naïves Bayes method:

The goal of structural sequential pattern mining is to discover correlations between attributes that are shared by a large number of entries. If property A is in relation with property B for the majority of positive examples in a binary classification issue, hence a record in which feature A is in interaction with B may belong to a negative occurrence. If only one rule involving B is broken, it does not infer much, but if several such rules are broken, it increases the possibility that the object belongs to the negative class. In association rule mining, frequently occurring attribute value conditions are searched in a dataset and used for prediction purposes thereby neglecting nonfrequent attribute values.

3.3. Software reliability growth models (SRGMs) using the naïves Bayes method:

SRGMs are scientific equivalences used in the development or testing phase used to model the software growth or reliability of the system. By empirically evaluating the performance of models using the testing data a suitable model is selected and defect forecasting is done. When we apply SRGMs to a project, it is used to guesstimate the latent defects and final defect counts with the help of appropriate mathematical models. During the SDLC phase the reliability of the model is measured by SRGMs. Wood applied various SRGMs on defect dataset and found momentous connection amongst the defect count through the period. It shows a comparative study of SRGMs with consumer electronics embedded software.

3.4. Phase wise root causes analysis of software failures

It is clear that software testing, especially upgrades, environment issues, cloud-related issues, and configuration changes are concerns that require attention if software failures are to be avoided. Due to the integration of numerous applications created in multiple networks of various sectors employing web services and other applications, configuration and environment degradation emerge. The following explanations of the root causes from each stage of software failures were compiled. The support vector machine with the Gaussian kernel is employed in this study, and it is indicated by Equation (1).

$$K(x,\ y)\ =\ \exp\left(\frac{x-y}{\sigma^2}\right) \tag{1}$$

3.4.1. Supervised approach

Supervised learning is the term used to describe the machine learning task of trying to insinuate a function from labelled data. The training data consist of a set of training examples. In supervised learning, each sample is a pair that consists of the intended output value and an input item. After examining the training data, a supervised learning algorithm generates an inference value that may be applied to instances. Instances that are not yet visible in a perfect environment will be able to have their class labels determined with accuracy by the algorithm. A classified data set that can be utilized to train and test if the classification algorithms are necessary for supervised learning.

Some of the supervised learning methods that have been examined for the RCA use case include the ones listed below:

- Naïve Bayes
- Maximum entropy

- Decision trees
- Support vector machine

3.4.2. Unsupervised approach

A sort of machine learning algorithm called unsupervised learning is used to derive conclusions from datasets that only contain input data without tagged replies. The most popular unsupervised learning algorithm is cluster analytics, which reveals interesting patterns or grouping in data by retrieving useful information. The algorithm groups the data-based feature similarity. With this method we clustered the training dataset and produced a classifier using an unsupervised learning algorithm. Then, we assigned the correct groupings to newly received bug files using our clustering methodology. When the clustered data were examined for potential RCAs, it was discovered that some of the data fell into the same category.

3.5. Proposed naïve Bayes algorithm

Naïve Bayes is an applied training that is used for the statistical technique knowledge grouping. As the name naïve suggests, this method blithely asserts that the characteristics of a given class are autonomous. Features presuppose strong or gullible isolation. It serves as a template that is used to apply class labels to problematic items and is allocated as a vector that is utilized to extract class descriptors from finite collections. Given their oversimplified character and presumptions, naïve Bayes is categorized under the complex issue of real-world difficulties.

The Bayes theorem is the foundation of NB, a family of simple probabilistic techniques featuring unconventionality suppositions among the predictors. The NB model is accurate, easy to build, and can be used to analyse any dataset with a lot of data. A combination of $P(c)$, $P(x)$, and $P(x|c)$ yields the posterior probability, $P(c|x)$.

The impact of a forecaster's value (x) on the presumptive class (c) is unaffected by the value of other forecasters.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{2}$$

The group of variables with the highest prediction accuracy would involve a thorough evaluation of all potential selected features. We might also take a sample of the simulation area in order to generate a simulation for each classification purpose. To generate the model, they explore the supermodel by identifying a collection of characteristics at random. We begin with counting models with a specific attribute and gradually increase the size of the models until we have a suitable sample size. Each module's characteristics were randomly chosen. As a result, we take a different strategy than the standard XGBoost implementation. The first part of our sampling technique is explained in Algorithm 1.

ALGORITHM 1: Taking a sample of the Naïve Bayes model space

Input: Dataset Train $_{\text{data}} = \{x_t, y_c\}^n{}_{t=1}$, Test $_{\text{data}} = \{x_c, y_t\}_{t=1}$

Proposed Naïve bayes algorithm

Output: Predict the error in terms of SRG and SDP

Initialization: $N = 100$ for number of runs

Procedure: SDP – Naïve bayes

For $i = 1$ to N

Select m top features from train $_{\text{data}}$

$D_{\text{new-train}} = x^2 = \sum_{r=1}{}^{p} \sum{}^{q}{}_{x=1} \frac{(D-Ers)2}{E}$

$D_{\text{new-test}}$ = Select same features from Test $_{\text{data}}$ as $D_{\text{new-test}}$

Model = naïve Bayes ($D_{\text{new-train}}$ Validation size = 0.2)

End for

4. Experimental results

The experiments were carried out by utilizing the eclipse bugs, public data set can be proposed as naïve bayes algorithm when compared with the existing classifiers, namely SVM, random forest, and K nearest method. The findings of the quality measurement assessment of the suggested software defect prediction model are listed and explained below.

4.1. Sensitivity (recall)

Sensitivity measure can also be referred as "Recall" which is computed by taking the ratio of number and it is divided by the number of true positives and false negative instances. If the sensitivity is high then, it suggests that the Experimental Study 72 has higher aggregate of class label predicted as 1 appropriately. It will not permit us to misclassify the positive class labels of the model which is under consideration.

$$\text{Sensitivity} = \frac{\text{Number of true positives}}{\text{Number of true positives} + \text{Number of false negatives}} \tag{3}$$

4.2. Specificity

A parameter that can be measured is the ratio of true negatives divided by the sum of true negative things and positive result occurrence. If the specificity is high enough, it means that we accurately anticipated a higher percentage of class labels as 0. This measure performance measure will not permit to misclassify the negative class labels of the model taken under consideration.

$$\text{Specificity} = \frac{\text{Number of true negatives}}{\text{Number of true negatives} + \text{Number of false positives}} \tag{4}$$

Table 1. Naïve Bayes classifier was used to perform statistical functional testing on the KC1 sample.

Labels	Precision	Sensitivity	Specificity	Recall	*F*-score	False positive	True negative	True positive	Accuracy
Class1	0.5024	0.6544	0.88564	0.6754	0.5545	234	1534	205	85.34%
Class 2	0.6785	0.8796	0.5354	0.7556	0.3498	765	5569	456	

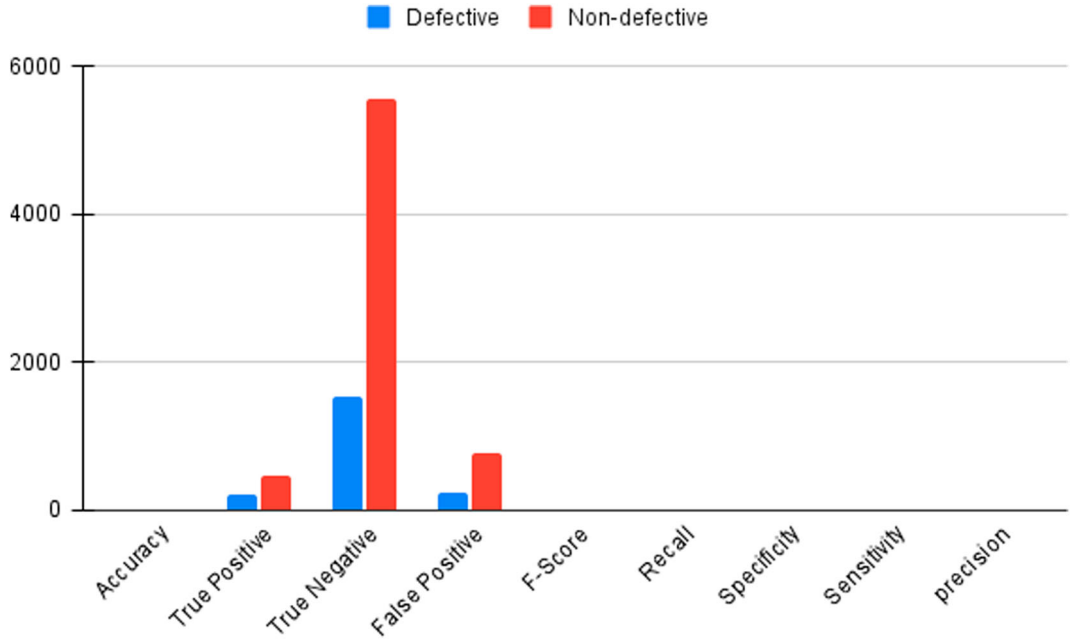

Figure 2. Classification performance of the SVM classifier.

Table 2. Statistical evaluation process and use a decision tree classifier for the eclipse bugs database.

Section	Accurate	Sensitivity	Specificity	Recall	*F*-score	False positive	True negative	True positive	Accuracy
Class 1	0.7653	0.8653	0.5467	0.8743	0.4875	70	1772	145	96.56
Class 2	0.5657	0.8786	0.3765	0.2657	0.5643	205	145	1665	

4.3. Accurate

Accuracy is the ratio of the total number of true positive and false positive cases. When the accuracy of prediction is high, the percentage of positive classifier expected from all positive class labels is indeed high (Table 1).

$$\text{Accuracy} = \frac{\text{Number of true positives}}{\text{Number of true positives} + \text{Number of false positives}} \quad (5)$$

In the next phase, we present statistical performance measurement using the SVM classifier where several performance measurement metrics are considered such as precision, sensitivity, specificity, recall, *F*-measurement false positive, true negative, true positive and, finally, the accuracy of the classifier is given. The present ROC analysis is shown in Figure 2.

It is equal to 1 occurs when there is no overlapping in distribution and a perfect separation of two groups is possible and because of that the graph reaches the upper left corner without any overlap and correctly classifies defective and non-defective modules.

They present statistical performance measurement using the decision tree classifier where the following several performance measurement metrics are considered specificity, susceptibility, selectivity, recalls and *F*-measurement. The reliability of the classifiers is reported in terms of false positive, true negative, true positive, and ultimately, false positive. Now, we present ROC analysis as labelled in Table 2 false positive (Figure 3).

The area is equal to when the curve will coincide with the diagonal. The curve is equal to 1 occurs when there is no over lapping in distribution and there a perfect separation of two groups is possible and because of that the curve reaches the upper left corner without any overlap and correctly classifies defective and non-defective modules (Table 3).

We offer quantitative assessment using the naïve bayes classifier, which considers accuracy, sensitivity, specificity, recalls, and *F*-measurement among other performance parameters. Figure 4 shows the false positive, true negative, true positive, and accuracy of the classification. Performance comparison is shown in Table 4.

5. Conclusion

This work mainly proposed software defect prediction using data-mining techniques with a machine learning

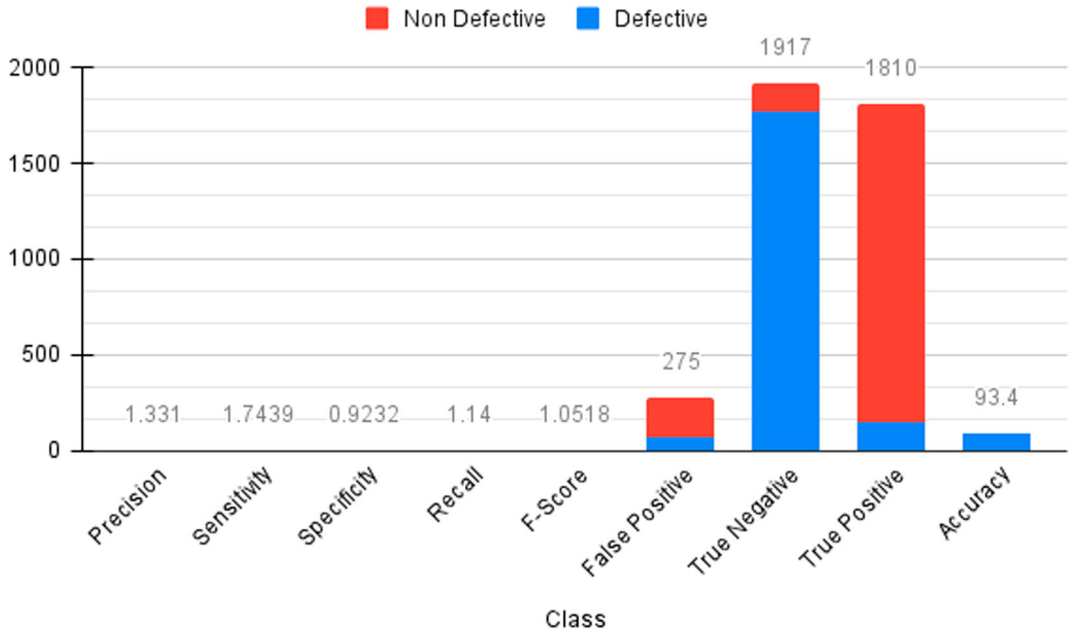

Figure 3. Classification performance of the decision tree classifier.

Table 3. Analytical assessment of the eclipse bugs dataset**.**

Class	Precision	Sensitivity	Specificity	Recall	False positive	True negative	True positive	Accuracy
defective	0.4863	0.8562	0.8664	0.8867	56	432	245	84.56
Non- defective	0.8455	0.8788	0.8772	0.8678	54	342	321	

Table 4. Performance comparison.

Method	Accuracy
RF-FS [16]	85.20
SVM-AdaBoost [18]	79
SVM-PCA [17]	83
RF-PCA [17]	83
Proposed	85.34

approach. However, this area has turned into a fascinating field for scientists where numerous practices are performed for promoting and refining the performance of defect detection or bug prediction. The proposed work concentrates on constructing a new combined approach using feature reduction and classification for the purpose of addressing the classification accuracy for a huge dataset. The eclipse bugs dataset is applied where maximum likelihood is also combined to reduce the error while reconstructing the eclipse bugs dataset to obtain the feature reduction model. The major objective of this article was to use software system analysis to reduce manual interaction in the software development

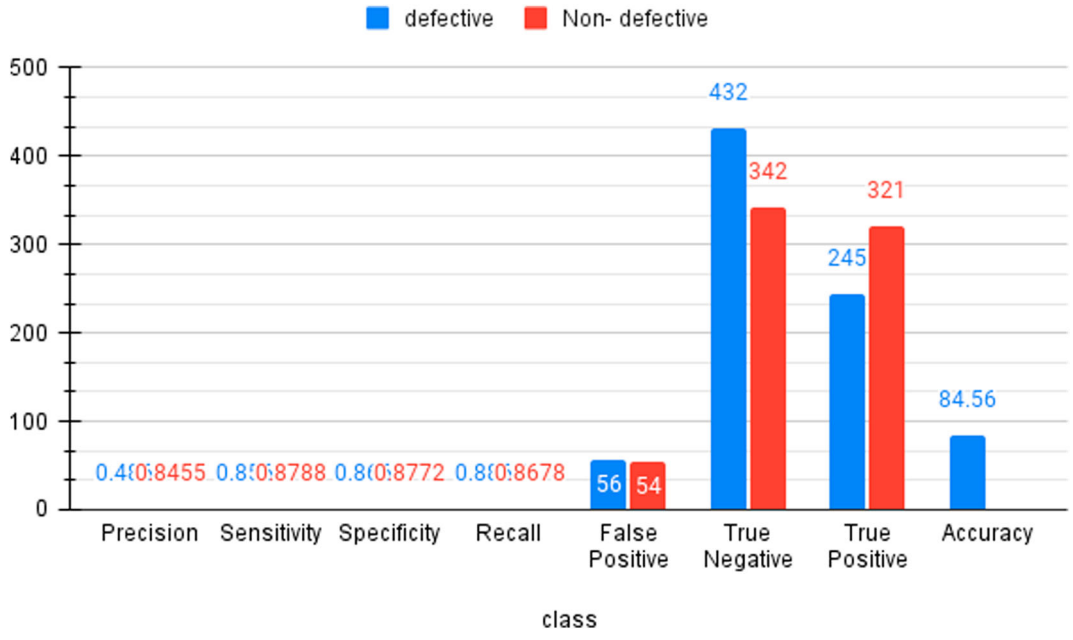

Figure 4. Classification performance of the naïve Bayes classifier.

process. For the most frequent coding errors we built root cause analyses and attributed those to eclipse bugs and issues. In the future, we will try to analyse various ensemble classifier types after applying data balancing approaches because these techniques allow us to enhance error measure as well as obtain the best results. On a large dataset, we will additionally use a different optimization strategy.

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

[1] Kaur A, Kaur K, Chopra D. An empirical study of software entropy based bug prediction using machine learning. Int J Syst Assurance Eng Manage. 2016;8(S2): 599–616. doi:10.1007/s13198-016-0479-2

[2] Jonsson L, Borg M, Broman D, et al. Automated bug assignment: ensemble-based machine learning in large scale industrial contexts. Empir Softw Eng. 2015;21(4):1533–1578. doi:10.1007/s10664-015-9401-9

[3] Esteves G, Figueiredo E, Veloso A, et al. Understanding machine learning software defect predictions. Autom Softw Eng. 2020;27:369–392. doi:10.1007/s10515-020-00277-4

[4] Dong F, Wang J, Li Q, et al. Defect prediction in android binary executables using deep neural network. Wirel Pers Commun. 2017;102:2261–2285. doi:10.1007/s11277-017-5069-3

[5] Huang L, Ng V, Persing I, et al. AutoODC: automated generation of orthogonal defect classifications. Autom Softw Eng. 2014;22(1):3–46. doi:10.1007/s10515-014-0155-1

[6] Kastro Y, Bener AB. A defect prediction method for software versioning. Softw Qual J. 2008;16(4):543–562. doi:10.1007/s11219-008-9053-8

[7] Barenkamp M, Rebstadt J, Thomas O. Applications of AI in classical software engineering. AI Perspectives. 2020;2(1):1–15. doi:10.1186/s42467-020-00005-4.

[8] Althar RR, Samanta D. The realist approach for evaluation of computational intelligence in software engineering. Innov Syst Softw Eng. 2021;17(1):17–27. doi:10.1007/s11334-020-00383-2

[9] Hamdy A, Ezzat G. Deep mining of open source software bug repositories. Int J Comput Appl. 2020;44:1–9. doi:10.1080/1206212x.2020.1855705

[10] Riccio V, Jahangirova G, Stocco A, et al. Testing machine learning based systems: a systematic mapping. Empir Softw Eng. 2020;25:5193–5254.. doi:10.1007/s10664-020-09881-0

[11] Nalbach O, Linn C, Derouet M, et al. Predictive quality: towards a New understanding of quality assurance using machine learning tools. Lecture Notes Business Inform Proc. 2018;320:30–42. doi:10.1007/978-3-319-93931-5_3

[12] Choetkiertikul M, Dam HK, Tran T, et al. Automatically recommending components for issue reports using deep learning. Empir Softw Eng. 2021;26(2):39. doi:10.1007/s10664-020-09898-5.

[13] Tabernik D, Šela S, Skvarč J, et al. Segmentation-based deep-learning approach for surface-defect detection. J Intell Manuf. 2019;31:759–776. doi:10.1007/s10845-019-01476-x

[14] Okutan A, Yıldız OT. Software defect prediction using Bayesian network Empir Softw Eng 2012;19(1):154–181. doi:10.1007/s10664-012-9218-8

[15] Tabernik D, Šela S, Skvarč J, et al. Segmentation based deep-learning approach for surface-defect detection. J Intell Manuf. 2019;31:759–776. doi:10.1007/s10845-019-01476-x

[16] Mumtaz B, Kanwal S, Alamri S, et al. Feature selection using artificial immune network: an approach for software defect prediction. Intell Autom Soft Comput. 2021;29:669–684. doi:10.32604/iasc.2021.018405

[17] Cetiner M, Sahingoz OK. A comparative analysis for machine learning based software defect prediction systems. Proceedings of the 2020 11th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2020 1–3 July.; Kharagpur, India; 2020. p. 1–7.

[18] Alsaeedi A, Khan MZ. Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. J Softw En. Appl. 2019;12:85–100. doi:10.4236/jsea.2019.125007