Indonesian Journal of Electrical Engineering and Computer Science
Vol. 32, No. 1, October 2023, pp. 478~493
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v32.i1.pp478-493

Improving the accuracy of recurrent neural networks models in predicting software bug based on undersampling methods

Nasraldeen Alnor Adam Khleel, Károly Nehéz
Institute of Information Science, University of Miskolc, Miskolc-Egyetemváros, Hungary

Article Info

Article history:

Received Dec 13, 2022
Revised Mar 27, 2023
Accepted Jul 8, 2023

Keywords:

Class imbalance
Recurrent neural networks
Software bug prediction
Software metrics
Undersampling methods

ABSTRACT

The process of identifying software bugs is of paramount importance as it ensures software reliability and facilitates maintenance activities. The quality improvement process of software relies heavily on software bug prediction (SBP). In SBP, the task of accurately identifying defective source code poses a significant challenge. Numerous of machine learning (ML) models has been developed specifically to address this challenge in SBP. Nonetheless, the class imbalance issue restricts the potential of these models to predict software bugs accurately. This issue poses a significant hindrance to the efficiency of these models, leading to imbalanced false-positive and false-negative outcomes. Previous studies have paid limited attention to addressing the challenge of class imbalance in SBP. This study aims to fill this research gap by employing a combination of two recurrent neural networks (RNNs), namely long-short-term memory (LSTM) and gated recurrent unit (GRU), along with an undersampling method (near miss) to effectively tackle this issue. Experiments have been conducted on publicly available benchmark datasets, considering both class-level and file-level metrics. The experimental results lead to the conclusion that our models outperform others and the combination of RNNs models with undersampling methods leads to improved bug prediction performance, particularly for datasets with imbalanced class distributions.

Corresponding Author:

Nasraldeen Alnor Adam Khleel
Institute of Information Science, University of Miskolc
3515 Miskolc, Miskolc-Egyetemváros, Hungary
Email: nasr.alnor@uni-miskolc.hu

1. INTRODUCTION

Software bug prediction (SBP) is a process for classifying fault-prone software modules based on some underlying properties of the systems, like software metrics that are extracted and collected from real data sets (historical data) during the software development process [1], [2]. Dealing with software bugs during the development process is problematic, as critical software bugs lead to potential risks that can lead to project failure. The final product should have as few bugs as possible to produce high-quality software. Early detection of software bugs can reduce development costs, time, rework efforts [3]. SBP serves as a means to track software modules and assess their reliability by examining specific parameter characteristics obtained from software projects [4]. Several machine learning (ML) techniques have been introduced to deal with the SBP problem [5], [6]. Recurrent neural networks (RNNs) belong to a category of ML models capable of handling a sequence of inputs and retain its state while processing the following series of information and efficiently acquiring the nonlinear features that are in order [7]. RNNs find extensive utility across various domains, encompassing but not restricted to pattern recognition, identification, classification, vision, speech, control systems, and numerous other applications [8]. So RNNs will be a promising technique for SBP. RNNs

encounter the obstacle of long-term dependencies when dealing with excessively lengthy input sequences, leading to an inability to ensure long-term nonlinear relationships [1]. When learning sequences, it is possible to encounter either the problem of gradient vanishing or gradient explosion. To tackle this issue, numerous optimization theories and advanced algorithms have been introduced, including Bidirectional long-short-term memory (LSTM), LSTM networks, gated recurrent unit (GRU) networks, Independent RNNs, echo state networks, and others [9].

The reason for utilizing both LSTM and GRU-optimized algorithms in RNNs for this research is due to their shared goal of efficiently tracing long-term dependencies while addressing the issue of vanishing gradients [10]. GitHub is widely regarded as a popular hosting service for source code, and numerous prominent open-source teams utilize it to manage their projects [9], [11]. So, the datasets selected in the study were obtained from the GitHub repository. But these selected data sets are imbalanced, and class imbalance is one of the problems facing ML techniques, which severely hinders the efficiency of ML techniques. Several data balancing methods have been introduced in previous studies to address this problem and allow the training of robust and well-fit ML models. Therefore, this study seeks to tackle the problem of class imbalance by utilizing data balancing methods (sampling methods) and to analyze the effect of these methods on the performance of RNNs models in SBP. In the first step, we apply sampling methods to balance the data sets. Next, the proposed models are trained and evaluated using these balanced datasets. The evaluation process includes various performance measures derived from the confusion matrix, such as accuracy, precision, recall, and F-measure. Additionally, common metrics like matthews correlation coefficient (MCC), area under the receiver operating characteristic curve (AUC), area under the precision-recall curve (AUCPR), and mean squared error (MSE) are also incorporated in the evaluation. In summary, the aim of this study is to outline its objective and primary contributions as follows: i) our research proposes a new method to enhance SBP accuracy. This method integrates LSTM and GRU models with the near miss method to address class imbalance challenges. By combining advanced recurrent neural network (RNNs) models with targeted undersampling, our method shows promise in improving bug prediction models' performance and reliability; ii) to evaluate the efficiency and effectiveness of the proposed method, various performance metrics will be utilized, and a comparative analysis will be performed against the existing methods in SBP; and iii) we show that by employing data-balancing methods to balance the data set, significant enhancement can be achieved in the performance of RNNs models in SBP.

The paper follows the following structure: in section 2 provides an overview of related works, while sections 3 and 4 delve into the background information on LSTM and GRU, respectively. In section 5 outlines the research questions, and section 6 details the research method. In section 7 presents the experimental results and discussions, and the conclusion is presented in the final section 8.

2. RELATED WORK

Numerous research endeavors have explored SBP, proposing diverse techniques to enhance model accuracy. Prior studies examined ML algorithms (e.g., support vector machines (SVM), decision trees, and random forests) and data augmentation for class imbalance. Certain research explored ensemble strategies like Bagging and Boosting to amplify bug prediction accuracy. Despite potential, the synergy of RNNs and undersampling for SBP remains uncharted. This article addresses this gap, exploring undersampling's potential to enhance RNN-based bug prediction, offering novel insights for more dependable software development. Bani-Salameh *et al.* [1] introduced a model that utilizes LSTM model for the automated allocation of bugs. The model's effectiveness was measured against two ML algorithms, and the findings revealed that LSTM outperformed them, with superior accuracy and efficiency in assigning bug priorities. Hammouri *et al.* [5] developed an SBP model utilizing three distinct ML algorithms, which were then evaluated using historical data. The findings revealed that ML models could be utilized with great accuracy, and a comparison with other ML approaches showcased the superior performance of the proposed method. Fan *et al.* [9] proposed an attention-based RNN framework specifically designed for software defect prediction (SDP). The experimental outcomes showcased that the proposed model exhibits significant enhancements, achieving a 14% improvement in F1 measure and a 7% improvement in AUC in comparison to the currently employed techniques for SBP.

Zhou and Lu [12] introduced a SDP model that utilized an LSTM model incorporating bidirectional and tree structures. Through the evaluation results, it was found that the proposed model surpassed several defect prediction models when applied to eight pairs of open-source Java projects. Tong *et al.* [13] presented a novel approach to address the issue of class imbalance in SDP. The effectiveness of the proposed method was extensively evaluated on diverse datasets and compared with existing techniques. The experimental results unequivocally demonstrated the superiority of the novel method over the alternatives, highlighting its significant potential in effectively handling the class imbalance problem. These findings emphasize the valuable contributions of this innovative approach in enhancing the accuracy and reliability of SDP models.

As a result, this approach holds promise for improving the overall performance of SDP in real-world applications. Ye *et al.* [14] introduced a bug classification model that utilizes an LSTM model. To comprehensively evaluate its capabilities, the model underwent rigorous testing with a significant number of bug reports collected from three distinct software projects. The evaluation results unequivocally showcased the superiority of the proposed model over alternative approaches. This notable finding highlights the model's remarkable performance and its potential to substantially enhance bug classification accuracy, making a valuable contribution to the field of SBP. The success of this model opens up new possibilities for more accurate and efficient bug detection, ultimately benefiting software development practices. Khuat and Le [15] performed an empirical investigation to examine the importance of sampling methods in SDP. Their experimental findings revealed that incorporating sampling methods in conjunction with ensemble learning models yielded beneficial outcomes, effectively mitigating the challenges associated with class imbalance. However, it should be noted that in certain datasets, this approach resulted in lower prediction accuracy.

Munir *et al.* [16] proposed a method for SDP, integrating both GRU and LSTM models. The method was rigorously tested across multiple datasets and compared with various existing approaches. The experimental findings provided compelling evidence, showcasing the superiority of the proposed method over alternative techniques. This significant improvement in performance highlights the potential of the novel method in advancing SDP accuracy, making it a valuable addition to the repertoire of predictive modeling techniques in the software development domain. Kukkar *et al.* [17] introduced a novel deep learning (DL) model known as bug severity classification, tailor-made for multiclass severity classification tasks. The model seamlessly integrates convolutional neural network (CNN) and random forest with boosting. To thoroughly assess its effectiveness, the model was applied to five distinct open-source projects. The evaluation results showcased a significant improvement in bug severity classification performance, providing compelling evidence of the superiority of the proposed model. This noteworthy enhancement underscores the potential of this innovative DL-based approach in accurately and efficiently classifying bug severity levels, offering valuable insights for bug management and software quality improvement across diverse projects. Khleel and Nehéz [18] presented a model for SBP using four ML algorithms.

The historical data obtained from NASA public datasets were used to evaluate the experiments. The proposed approach was subjected to comparison with other ML approaches. Based on the evaluation process and results, it was concluded that ML algorithms could effectively predict bugs. Liang *et al.* [19] proposed a framework named Seml was introduced for SDP, leveraging the power of the LSTM model. Through rigorous experimentation, the research demonstrated that the proposed method outperformed the latest defect prediction techniques, showcasing its remarkable superiority in performance. These results firmly establish Seml as a promising and effective approach in SDP, presenting valuable prospects for advancing the accuracy and reliability of defect detection in software development projects. Ferenc *et al.* [20] introduced a method to adapt deep neural networks (DNNs) for SBP. The method's performance was assessed using various datasets. The results illustrated that incorporating static metrics with DNNs can greatly enhance prediction accuracy. Upon examination of earlier research on SBP, it was observed that the majority of proposed techniques neglect the problem of class imbalance. Research that specifically dealt with the issue of class imbalance and provided solutions as referenced in [13], [15] emphasizes the crucial role of data balancing methods in enhancing SBP accuracy. Hence, the main aim of this study is to mitigate the problem of class imbalance and enhance the efficacy of the proposed models.

3. LONG-SHORT-TERM-MEMORY

The LSTM networks, belonging to the category of RNNs, possess a specialized architecture that enables them to detect patterns within sequential data [9]. The purpose of introducing LSTM networks was to resolve or prevent issues of long-term dependencies, which regular RNNs are susceptible to due to an unstable gradient when connecting prior and current information [14], [19]. The repeating module of LSTM networks is depicted in Figure 1, which illustrates the interaction between its layers. Standard RNNs, which take input sequences where each step refers to a specific moment [10]. In LSTM networks, the output at a specific time moment t, denoted as o_t, is not solely dependent on the current input x_t. However, it is important to note that the output at time t is not solely dependent on the current input; it is also influenced by the output from the previous moment, $t-1$ [19]. This implies that the system's behavior is influenced by its own past states. To express this concept mathematically, we can represent the output at time (t) through the (1).

$$h_t = f(U \times x_t + W \times h_{t-1} + b) \quad (1)$$
$$o_t = g(V \times h_t + c)$$

The weights of the RNNs are labeled as U, V, and W, while the biases are denoted as b and c. The activation functions of the neurons are represented by f and g. Within the LSTM context, the cell state plays a crucial role by preserving information from previous time steps and propagating it through the entire LSTM chain, facilitating the efficient handling of long-term dependencies. A notable component is the forget gate, responsible for filtering out irrelevant information from the previous time step. This forget gate can be mathematically expressed as (2).

$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f) \quad (2)$$

The activation function is denoted by the symbol σ, with W_f and b_f representing the weights and bias of the forget gate. The output of the input gate plays a crucial role in determining which information to retain from the current moment, and its mathematical representation is as (3).

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i) \quad (3)$$

The activation function is denoted by the symbol σ, and the weights and bias of the input gate are represented by W_i and b_i, respectively. Leveraging the information obtained from both the forgetting gate and the input gate, we can advance the LSTM process. The cell state C_{t-1} is updated using the (4).

$$\check{C}_t = \tanh(W_c . [h_{t-1}, x_t] + b_c) \quad (4)$$
$$\check{C}_t = f_t \times C_{t-1} + i \times \check{C}_t)$$

$\check{C}_t$ symbolizes a candidate value aimed at incorporating it into the cell state, while C_t represents the updated cell state at the present moment. Ultimately, the output gate decides which information should be emitted by taking into account both the previous output and the current cell state.

$$o_t = \sigma(W_o . [h_{t-1}, x_t + b_o] \quad (5)$$
$$h_t = o_t \times \tanh(C_t)$$

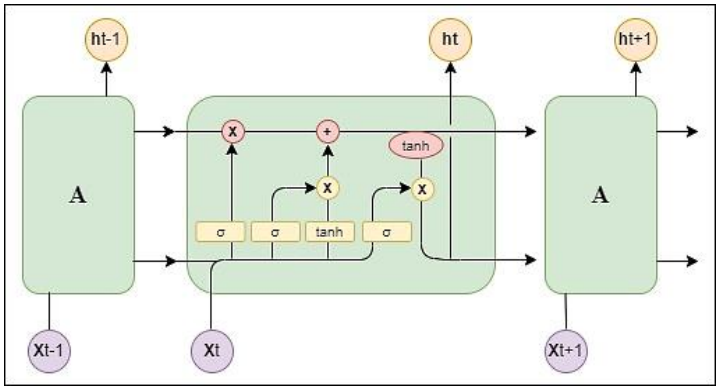

Figure 1. Depicts the interacting layers of the repeating module in an LSTM model [21]

4. GATED RECURRENT UNIT

The GRU network, a type of RNN, is specifically designed to handle long-term information dependencies while addressing the issue of gradient vanishing. Its specialized architecture makes it a powerful solution for processing sequential data and is particularly effective in tasks involving capturing long-range dependencies, like natural language processing and time series analysis [8]. By mitigating gradient vanishing, the GRU network ensures stable and efficient training, making it a valuable tool in the domain of DL and sequential data processing. GRU, in comparison to LSTM, incorporates fewer parameters and includes the update and reset gates in addition to the forget gate, as depicted in Figure 2. The update and reset gates in GRU improve and optimize the learning mechanism [9]. Within the GRU network, the update gate assumes a crucial role in determining the extent to which past information should be retained and carried forward to the future. Conversely, the reset gate assists in determining the degree to which past information should be ignored or forgotten [22], [23]. The calculation of the update gate in the GRU network can be demonstrated by the (6).

$$z(t) = \sigma(W(z).[h(t-1), x(t)]) \quad (6)$$

Where $z(t)$ is the update gate, $h(t-1)$ represents the output of the previous neuron, and $x(t)$ denotes the input of the current neuron, the weight of the update gate is denoted as $W(z)$, and the sigmoid function is represented by σ. The (7) below illustrates the calculation of the reset gate model in GRU networks. The reset gate is a crucial component that enables the network to control the flow of information and retain relevant context from previous time steps, contributing to its effectiveness in capturing long-term dependencies in sequential data.

$$r(t) = \sigma(W(r).[h(t-1), x(t)]) \quad (7)$$

Within the framework of a GRU network, the reset gate is denoted as $r(t)$, $h(t-1)$ signifies the output of the previous neuron, and $x(t)$ represents the input of the current neuron. The weight of the reset gate is symbolized by $W(r)$, and the sigmoid function is denoted by σ. The subsequent (8) elucidates the calculation of the output value of the hidden layer in a GRU network. The reset gate is a critical element that regulates the flow of information and aids in capturing relevant context from prior time steps, thus contributing to the network's ability to effectively model long-term dependencies in sequential data.

$$\check{h}(t) = \tanh\left(W\check{h}.[rt.h(t-1), x(t)]\right) \quad (8)$$

The output value in this neuron is denoted as $\check{h}(t)$, where $h(t-1)$ denotes the output of the previous neuron and $x(t)$ represents the input of the current neuron. $W\check{h}$ symbolizes the weight of the update gate, and the hyperbolic tangent function, tanh(), is applied. The calculated value of rt plays a crucial role in controlling the level of memory retention. The (9) provided below presents comprehensive details concerning the hidden layer and its contribution to the final output. This mechanism enables the GRU network to effectively capture and retain relevant information from previous time steps, facilitating its ability to model long-term dependencies in sequential data.

$$h(t) = \left(1 - z(t)\right).h(t-1) + z(t).\check{h}(t) \quad (9)$$

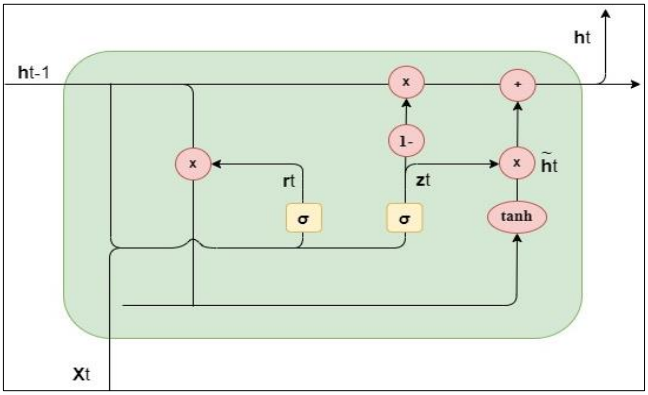

Figure 2. Depicts the interacting layers of the repeating module in a GRU model [23]

5. RESEARCH QUESTIONS

The main focus of this study is to assess the influence of data balancing methods on the predictive capability of RNN models in SBP. The research questions explored in this study specifically investigate the performance and accuracy of RNN models when incorporating data balancing methods for predicting software bugs. By investigating these research questions, valuable insights can be obtained regarding the effectiveness of various data balancing methods and their implications for enhancing the performance of RNN models in SBP.

RQ1: can the accuracy of RNN models in SBP be improved through the use of data balancing methods? The primary aim of this research question is to investigate the impact of data balancing methods on the predictive performance of two RNN models in the specific domain of SBP. By exploring various data

balancing techniques, this study aims to shed light on how these methods influence the accuracy and effectiveness of the RNN models in predicting SBP values. Through this investigation, a comprehensive understanding of the relationship between data balancing strategies and predictive performance can be gained, offering valuable insights for optimizing RNN-based predictions in the domain of SBP.

RQ2: are the proposed models more effective than the current state-of-the-art models in predicting software bugs? The main goal of this research question is to assess the effectiveness of the proposed models in predicting software bugs and compare them with state-of-the-art models. The study aims to thoroughly evaluate their predictive capabilities in this domain, contributing valuable insights to advance SBP and resolution.

6. METHOD

The primary objective of this research is to enhance the accuracy and effectiveness of software bug prediction through the integration of advanced RNN models, specifically LSTM and GRU, with the near miss method. By combining these advanced RNN models with the targeted undersampling technique, the study aims to address the class imbalance issue in a promising manner, leading to improved performance of the SBP models. This innovative fusion holds great potential to revolutionize the field of bug prediction, empowering developers and practitioners to build more reliable and efficient software systems. The method provides a detailed explanation of the various steps that have been implemented. The overall framework of our proposed method for SBP is visually presented in Figure 3, with each step further elaborated in the subsequent sections.

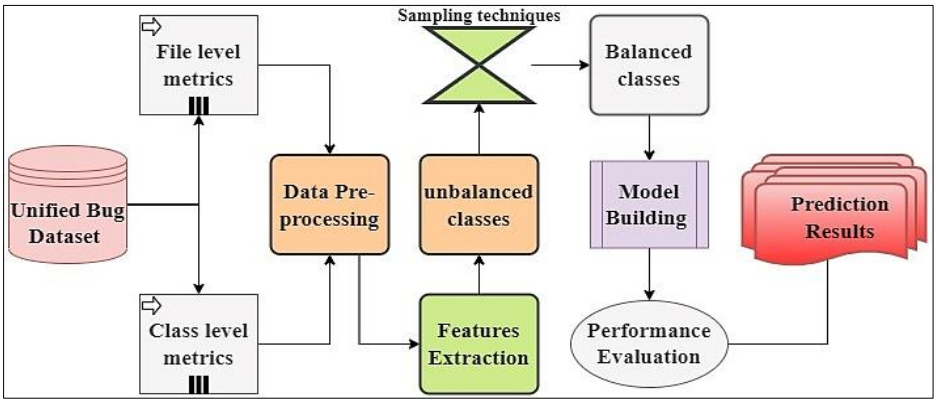

Figure 3. Shows the structure of the proposed method of SBP

6.1. Software metrics and public unified bug dataset

In the realm of software engineering, metrics serve as objective and systematic measurements utilized to evaluate different attributes of a software system. These metrics entail assigning numerical values or symbols to specific properties of the software under examination [23]. By employing software metrics, one can collect data on the structural aspects of a software design, allowing for a comprehensive analysis and interpretation of the software's characteristics using statistical methods [3]. These metrics play a vital role in assessing software quality, identifying potential issues, and guiding the decision-making process during software development and maintenance. By leveraging such objective measurements, software engineers can make informed and data-driven choices to improve the overall quality and reliability of software systems [23]. A public unified bug dataset aggregates bug reports and related data from various projects, domains, and platforms, facilitating bug prediction research and defect analysis. Researchers utilize this dataset to develop and assess bug prediction models, identify common bug patterns, and gain insights into bug characteristics across different projects. Its availability fosters collaboration and facilitates the comparison of bug prediction techniques, advancing bug detection and prevention. In this study, the authors considered five publicly available datasets and extracted shared software metrics from the corresponding source code. In their study, the researchers compiled a unified bug dataset, tailored for the development of novel bug prediction models at both class and file levels. Subsequently, they conducted a thorough comparison of metric definitions and values derived from various bug datasets. This comprehensive analysis enables a better understanding of the similarities and differences among different datasets, and it provides valuable insights into the challenges and opportunities in bug prediction research. The unified bug dataset serves as a valuable resource for the software development community, fostering collaborative efforts and facilitating the development of more robust and accurate bug prediction models [24]. Table 1 shows the public unified bug dataset, and Table 2 displays the utilized metrics.

Table 1. Describe the public bug datasets [24]

Datasets	System name	Lines of code
PROMISE datasets	Camel, Ant, Ckjm, Ivy, JEdit, Forrest, Lucene, PBeans, Synapse, Poi, Velocity, Xalan, Xerces, and Log4J	2,805,253
Eclipse datasets Bug dataset	Eclipse	3,087,826
Bug prediction Datasets	Equinox Framework, Eclipse JDT Core, Eclipse PDE UI, Lucene, and Mylyn	1,171,220
Bug catchers Datasets	Eclipse JDT Core, ArgoUML, and Apache commons	1,833,876
GitHub Bug datasets	Antlr 4, Broadleaf Commerce, Loader, and Android Universal Image, Ceylon IDE Eclipse Plugin, Hazelcast, Elasticsearch, MapDB, JUnit, MCT, Neo4J, Netty, Titan, mcMMO, OrientDB, and Oryx	1,707,446

Table 2. Shows the file and class-level metrics used in the public unified bug dataset [24]

Metrics level	Metric name
File-level metrics	Logical lines of code (LOF)
	Cyclomatic complexity (CC)
Class-level metrics	Weighted methods per class (WMC)
	Coupling between object classes (CBOC)
	Coupling between object classes inverse (CBCI)
	Response for a class (RFC)
	Depth of inheritance tree (DIT)
	Number of children (NC)
	Number of public methods (NPM)
	Lines of code (LC)

6.2. Data pre-processing and features selection

Before building a model, it is imperative to carry out pre-processing of the gathered data. This step is essential for streamlining the model training process and ensuring the creation of reliable and robust models [23], [25]. Data pre-processing involves a set of procedures utilized to improve the quality of data before constructing models. These procedures encompass activities such as eliminating noise and undesirable outliers, handling missing values, converting feature types, and more [17]. In order to enhance the model's efficacy, it is essential to normalize the values by scaling numeric data to a range of 0 to 1. In this regard, Min-Max normalization was employed on the dataset. The equation for calculating the normalized score is presented in (10). Features selection is a critical process that entails selecting a relevant subset of features from a larger pool of available attributes. The ultimate goal is to pinpoint the most informative and discriminative characteristics that exert a substantial influence on the performance of ML models. By carefully choosing these pertinent features, feature selection enhances model efficiency, mitigates overfitting, and improves overall predictive accuracy [23], [25]. Diverse techniques and algorithms, such as filter methods, wrapper methods, and embedded methods, are utilized for feature selection, each with its own specific guidelines and criteria. The ultimate aim of feature selection is to enhance the efficiency, interpretability, and generalization capability of ML models by concentrating on the most crucial and informative features [26]. In this study, embedded methods were employed as the foundation for our models, as they demonstrate better alignment with ML models. The maximum value of attribute x is represented by max (x), while the minimum value is denoted as min (x).

$$x_i = \frac{(x_i - X\,min)}{(X\,max - X\,min)} \tag{10}$$

6.3. Class imbalance and data sampling methods

Class imbalance is a term used in ML to describe the condition where the number of instances in one class is notably lower than in the other classes. This uneven distribution of classes can present difficulties for ML models as they may face challenges in effectively learning and predicting the minority class [23]. Class imbalance is frequently encountered in diverse domains, such as software bug data, where one class occurs significantly less frequently compared to the other classes. As a result, misclassification of cases in the minority class may occur, making it a significant issue to address [27]. The chosen reference dataset for this study is imbalanced, indicating a lack of the true distribution of learning instances. To address this issue, we modified the original dataset to make the data more realistic. Using the undersampling method called "near miss," the dataset's distribution was modified. Data sampling methods aim to address class imbalance by manipulating the dataset, typically by removing the majority of class samples, to achieve a more balanced

distribution [28], [29]. Among these methods, near miss is a technique used to tackle class imbalance by removing instances from the majority class. Its purpose is to equalize the class distribution by selectively eliminating examples that are in close proximity to the majority class instances, based on a predefined distance metric [30]. Figure 4 displays the distribution of learning instances across all datasets.

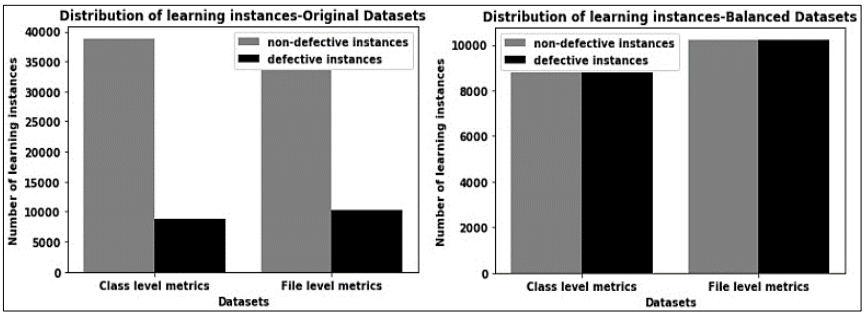

Figure 4. Displays the distribution of learning instances across all datasets

6.4. Models construction and evaluation

In this study, the models were developed using Keras, an advanced API built on the TensorFlow framework [23]. The training datasets consisted of 80% of the complete dataset, where the features were randomly selected, while the test datasets accounted for 20% of the dataset. The development of each model was carried out individually, using distinct parameters that are presented in Table 3. The assessment of the proposed models' performance involves employing various performance measures derived from the confusion matrix, as well as MCC, AUC, AUCPR, and MSE. The MCC is a performance metric for binary classification, considering true positives, true negatives, false positives, and false negatives [13]. Higher MCC values indicate better model performance [23]. AUC is a widely-used evaluation metric in binary classification, assessing ML model performance based on the ROC curve [8]. The ROC curve plots true positive rate against false positive rate across different threshold levels [12]. AUCPR measures the area under the precision-recall curve, ranging from 0 to 1. A perfect classifier achieves AUCPR of 1, indicating perfect precision and recall, while a random classifier has an AUCPR value close to the ratio of positive examples in the dataset, representing random performance [23]. MSE is a widely used evaluation metric in classification and regression tasks, quantifying model error by averaging the squared differences between predicted and actual values. It provides valuable insights into prediction accuracy and guides improvements [23]. A confusion matrix is a tabular representation in binary classification, summarizing model predictions and comparing them to actual labels [18]. It consists of four components: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) [23], [28]. Table 4 displays the layout of a confusion matrix.

Table 3. Displays the parameter configurations of the models

Parameters	Models	
	LSTM	GRU
Cell type	LSTM (64, 32), return sequences=True	-
Layers. GRU	-	100
Activation function	ReLU+sigmoid	Tanh+Sigmoid
Dropouts	0.2	0.2
Dense	64, 1	1
Optimizer	Adam	Adam
Learning rate	0.01	0.01
Loos function	Mean squared error	Mean squared error
Batch size	64	64
Epochs	100	100
Validation split	0.1	0.1
Verbose	1	1

Table 4. Displays the layout of a confusion matrix

Predicted	Actual	
	Positive	Negative
Positive	TP	FP
Negative	FN	TN

$$\text{Accuracy} = \frac{(\text{TP}+\text{TN})}{(\text{TP}+\text{FP}+\text{FN}+\text{TN})} \quad (11)$$

$$\text{Precision} = \frac{\text{TP}}{(\text{TP}+\text{FP})} \quad (12)$$

$$\text{Recall} = \frac{\text{TP}}{(\text{TP}+\text{FN})} \quad (13)$$

$$\text{F}-\text{Measure} = \frac{(2\,.\,\text{Recall}\,.\,\text{Precision})}{(\text{Recall}+\text{Precision})} \quad (14)$$

$$\text{MCC} = \text{TP}.\text{TN} - \text{FP}.\text{FN} \,/\, \sqrt{(\text{TP}+\text{FP}).(\text{TP}+\text{FN}).(\text{TN}+\text{FP}).(\text{TN}+\text{FN})} \quad (15)$$

$$\text{AUC} = \frac{\sum_{ins_i \in Positive\ Class} \text{rank}(ins_i) - \frac{\text{M}(\text{M}+1)}{2}}{\text{M}\,.\,\text{N}} \quad (16)$$

in this context, $\sum_{ins_i \in Positive\ Class} \text{rank}(ins_i)$ signifies the sum of ranks for all positive samples, while M and N respectively represent the numbers of positive and negative samples:

$$\text{AUCPR} = \int_0^1 \text{Precision}(\text{Recall})\ \text{d}(\text{Recall}) \quad (17)$$

$$\text{MSE} = \frac{1}{\text{n}} \sum_{\text{i}=1}^{\text{n}} (\text{x(i)} - \text{y(i)})^2 \quad (18)$$

in this context, n represents the number of observations, x(i) denotes the actual value, and y(i) represents the observed or predicted value for the i^{th} observation.

7. EXPERIMENTAL RESULTS AND DISCUSSION

For the experimental setup, we utilized a Python environment, and the identical project data served for both training and testing purposes. The RNN models proposed in this study were specifically crafted using classification patterns and subsequently assessed using various standard performance measures. This meticulous approach ensures a comprehensive evaluation of the proposed RNN models' effectiveness in predicting software defects, providing valuable insights for practitioners and researchers in the field of software engineering and ML. To answer the RQ1: Tables 5 and 6, as well as Figures 5 to 14, report the performance of the prediction models.

Table 5. Illustrates the performance measures for the proposed models on the class-level metrics dataset

Parameters	Proposed models	Performance measures							
		Accuracy	Precision	Recall	F-measure	MCC	AUC	AUCPR	MSE
Original dataset	LSTM	0.83	0.60	0.25	0.35	0.30	0.78	0.48	0.125
	GRU	0.82	0.58	0.16	0.26	0.23	0.77	0.44	0.130
Balanced dataset	LSTM	0.93	0.95	0.92	0.93	0.86	0.97	0.97	0.051
	GRU	0.93	0.94	0.92	0.93	0.86	0.96	0.97	0.063

Table 6. Illustrates the performance measures for the proposed models on the file-level metrics dataset

Parameters	Proposed models	Performance measures							
		Accuracy	Precision	Recall	F-measure	MCC	AUC	AUCPR	MSE
Original dataset	LSTM	0.78	0.62	0.18	0.28	0.24	0.75	0.49	0.152
	GRU	0.78	0.61	0.22	0.33	0.27	0.75	0.49	0.152
Balanced dataset	LSTM	0.88	0.94	0.81	0.87	0.76	0.93	0.95	0.090
	GRU	0.88	0.94	0.81	0.87	0.76	0.93	0.95	0.093

The results of our LSTM and GRU models are presented in Table 5. The outcomes are reported for both the original and balanced datasets, with a focus on class-level metrics. Notably, we observed that both the LSTM and GRU models attained the highest accuracy of 93% on the balanced dataset, while the GRU model exhibited the lowest accuracy of 82% on the original dataset. In terms of precision, the LSTM model achieved the highest value of 95% on the balanced dataset, while the GRU model demonstrated the lowest precision of

58% on the original dataset. As for recall, both models obtained the highest score of 92% on the balanced dataset, whereas the GRU model exhibited the lowest recall of 16% on the original dataset. Both models achieved the highest F-Measure score of 93% on the balanced dataset. However, on the original dataset, the GRU model had the lowest score of 26%. Both models achieved the highest MCC of 86% on the balanced dataset, whereas the GRU model had the lowest MCC of 23% on the original dataset. The LSTM model attained the highest AUC score of 97% on the balanced dataset, and the GRU model achieved the lowest score of 77% on the original dataset. On the balanced dataset, both models demonstrated the highest AUCPR score of 97%, while the GRU model exhibited the lowest AUCPR score of 44% on the original dataset. Additionally, the GRU model recorded the highest MSE of 0.130 on the original dataset, while the LSTM model achieved the lowest MSE of 0.051 on the balanced dataset.

The results of our LSTM and GRU models are presented in Table 6. The outcomes are reported for both the original and balanced datasets, with a specific focus on file-level metrics. Remarkably, both the LSTM and GRU models achieved the highest accuracy of 88% on the balanced dataset, whereas the lowest accuracy of 78% was observed for both models (LSTM and GRU) on the original dataset. Furthermore, the balanced dataset yielded the highest precision of 94% for both models (LSTM and GRU), while the GRU model had the lowest precision of 61% on the original dataset. In terms of recall, the balanced dataset produced the highest score of 81% for both models. Conversely, the LSTM model achieved the lowest recall of 18% when applied to the original dataset. Similarly, the balanced dataset resulted in the highest f-measure of 87% for both the LSTM and GRU models. Conversely, the LSTM model exhibited the lowest f-measure of 28% when working with the original dataset. Furthermore, both models (LSTM and GRU) attained the highest MCC of 76% on the balanced dataset, while the LSTM model had the lowest MCC of 24% on the original dataset. Similarly, the balanced dataset resulted in the highest AUC of 93% for both models (LSTM and GRU), while the original dataset yielded the lowest AUC of 75% for both models (LSTM and GRU). Both models also achieved the highest AUCPR on the balanced dataset, which is 95%, and the lowest AUCPR on the original dataset, which is 49%. In conclusion, both models (LSTM and GRU) achieved the highest MSE of 0.152 on the original dataset, while the LSTM model obtained the lowest MSE of 0.090 on the balanced dataset.

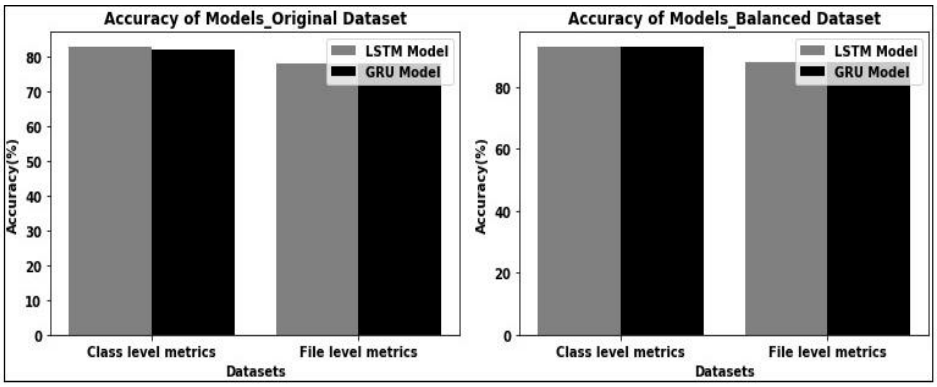

Figure 5. Displays the accuracy of the models across all datasets, including both class-level and file-level metrics

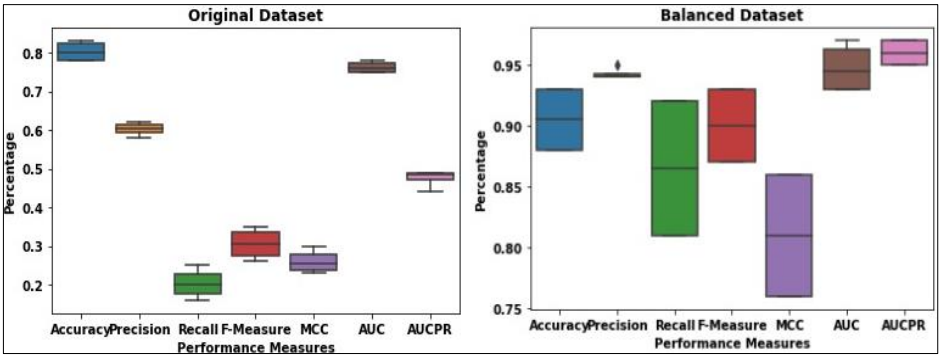

Figure 6. Showcases the boxplots illustrating the performance measures achieved by the proposed models on all datasets, encompassing both class-level, and file-level metrics

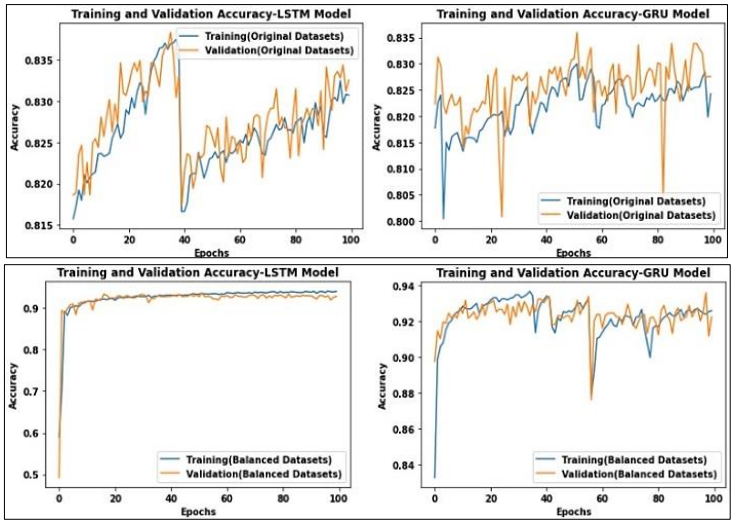

Figure 7. Represents the training and validation accuracy of the models across all datasets-class-level metrics

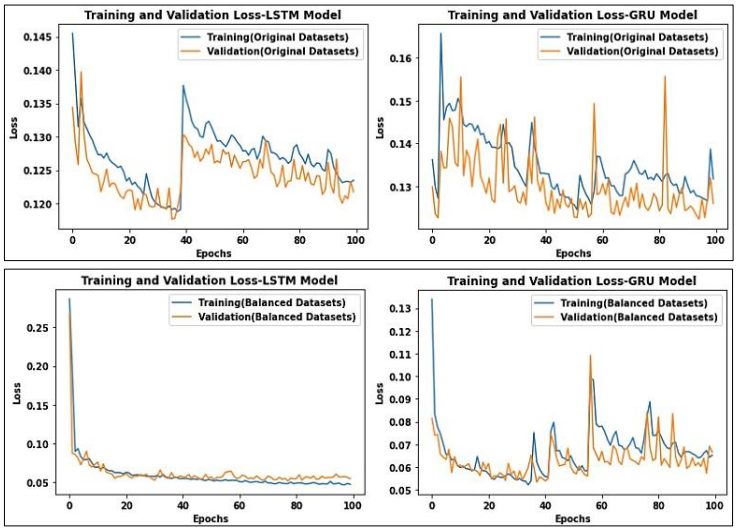

Figure 8. Represents the training and validation loss of the models across all datasets-class-level metrics

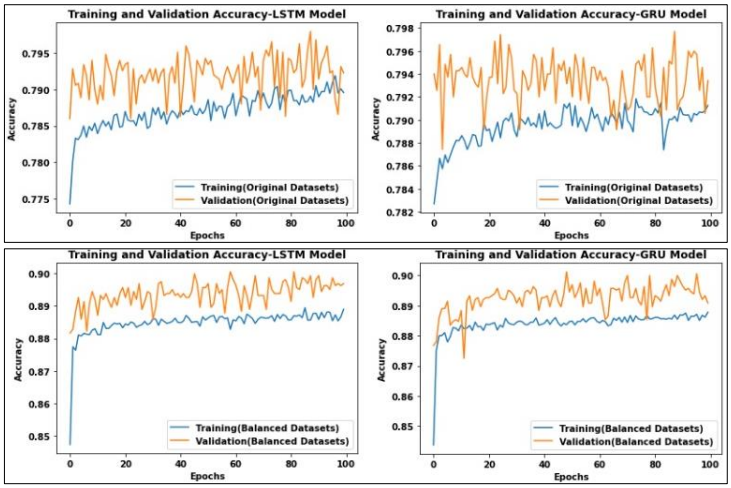

Figure 9. Represents the training and validation loss of the models across all datasets-file-level metrics

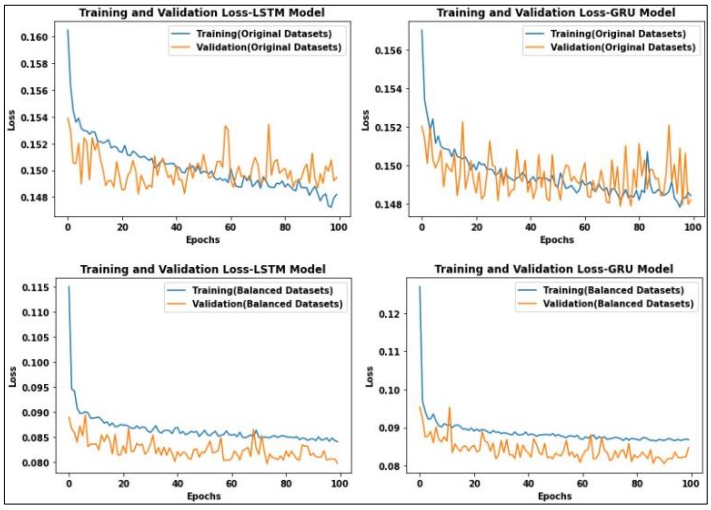

Figure 10. Represents the training and validation loss of the models across all datasets-file-level metrics

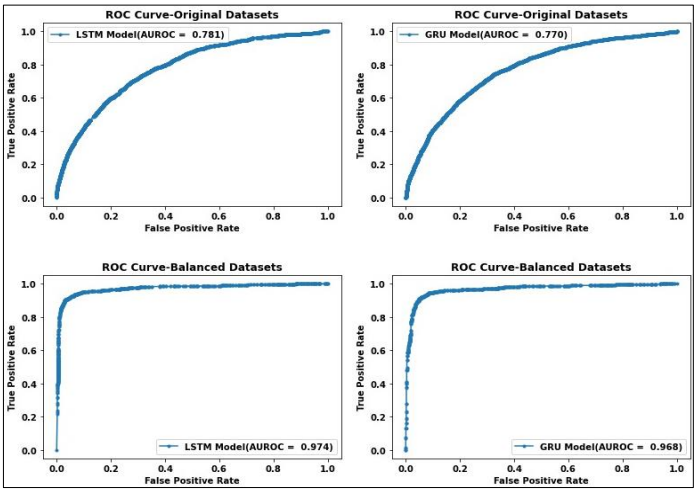

Figure 11. Illustrates the ROC curves of the models across all datasets-class-level metrics

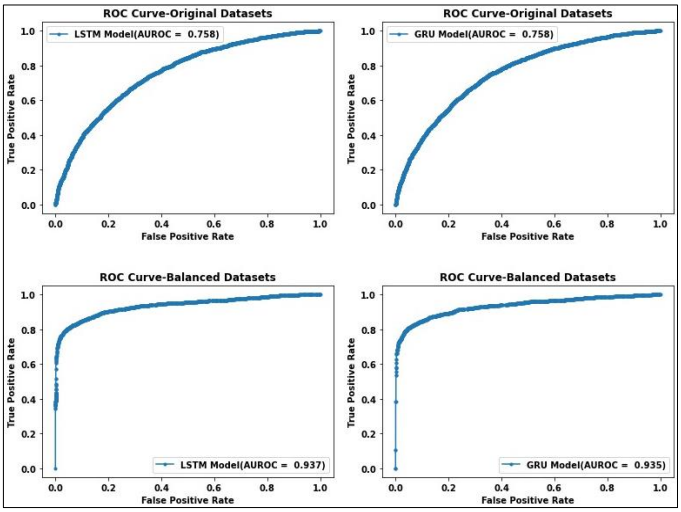

Figure 12. Illustrates the ROC curves of the models across all datasets-file-level metrics

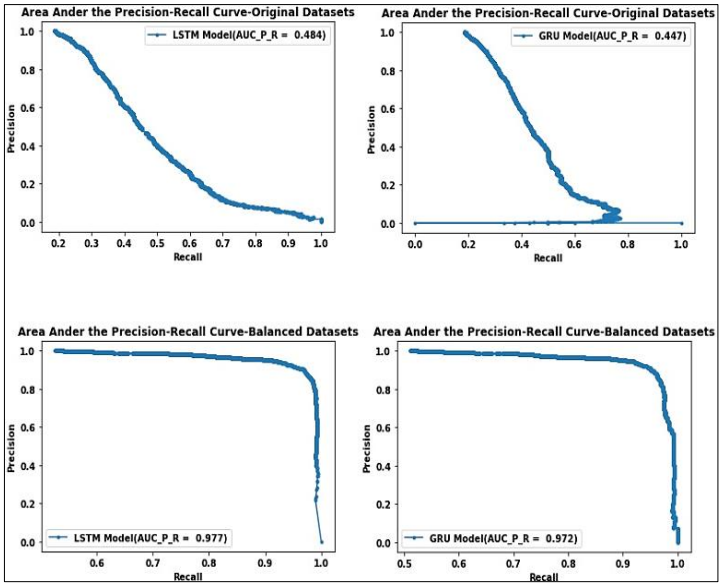

Figure 13. Illustrates the AUCPR of the models across all datasets-class-level metrics

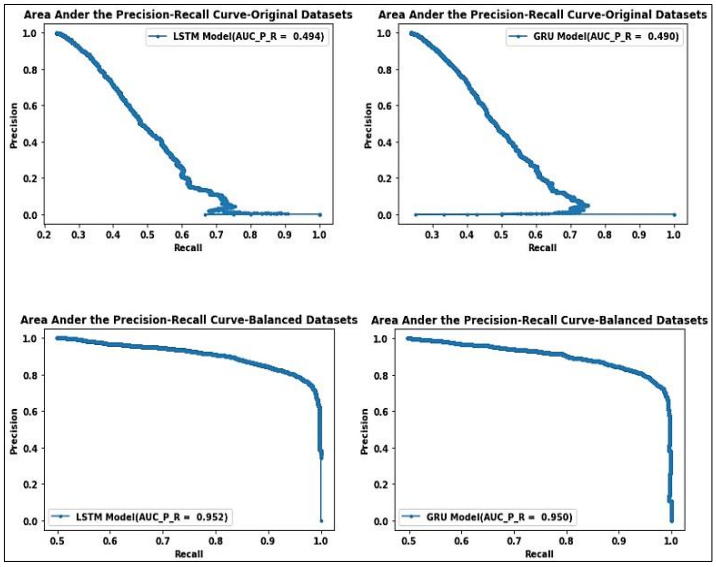

Figure 14. Illustrates the AUCPR of the models across all datasets-file-level metrics

Figure 5 presents a graphical representation of the accuracy performance of the models across all datasets. Additionally, Figure 6 displays Box plots, which effectively depict a range of performance measures for all datasets. Figures 7 to 10 illustrate the accuracy and loss values of the models during the training and validation phases across all datasets. These figures demonstrate a consistent trend of increasing accuracy and decreasing loss as the number of epochs advances. The high accuracy achieved and the low loss obtained serve as evidence of the effective training and validation of the proposed models. The AUC values achieved by the models on both the original and balanced datasets are presented in Figures 11 and 12, respectively. Furthermore, Figures 13 and 14 display the AUCPR scores obtained by the models on the original and balanced datasets, respectively.

After analyzing the outcomes of the proposed models, it was evident that they consistently achieved the highest scores across all datasets, underscoring their effectiveness in handling class imbalance. This notable performance validates the significance of employing undersampling methods to enhance the accuracy of RNN models in SBP. The success of the proposed models in addressing class imbalance further reinforces the

practicality and value of these techniques in mitigating the challenges posed by imbalanced datasets. The results of this study emphasize how undersampling methods have the potential to significantly enhance the performance of RNN models in SBP, presenting promising opportunities for more robust and reliable software defect detection strategies. These findings provide valuable insights into the effectiveness of undersampling techniques, paving the way for advancements in the field of bug prediction and fostering the development of more accurate and dependable bug detection systems.

To answer RQ2, we performed a detailed comparison between our research results and the outcomes of previous studies, with a particular focus on accuracy and AUC metrics. The comprehensive findings, presented in Table 7, showed that while certain earlier studies displayed higher values, our proposed method surpassed other techniques on the majority of datasets. This indicates the superior performance of our approach and its potential to outperform existing methods in the context of SBP. By conducting this rigorous evaluation and providing empirical evidence, our study contributes valuable insights to the field and underscores the effectiveness of our novel approach in improving bug prediction accuracy.

Table 7. Presents a comparison between the proposed method and other existing methods based on accuracy and AUC

Methods	Datasets	Accuracy	AUC
LSTM [1]	JIRA dataset	0.89	-
Naive bayes [5]	software fault datasets (DS1, DS2, and DS3)	0.89, 0.95, 0.95	-
Decision tree [5]	software fault datasets (DS1, DS2, and DS3)	0.95, 0.97, 0.99	-
ANN [5]	software fault datasets (DS1, DS2, and DS3)	0.93, 0.95, 0.96	-
RNNs [9]	PROMISE datasets (Camel, Lucene, Poi, Xerces, Jedit, Xalan, and Synapse)	-	0.79, 0.68, 0.79, 0.76, 0.82, 0.67, 0.64
Credibility-based imbalance boosting [13]	NASA datasets (CM1, KC1, PC1, and JM1)	-	0.72, 0.67, 0.85, 0.67
LSTM [14]	Bug report datasets (Eclipse platform UI and JDT)	0.67, 0.76	-
GRU-LSTM [16]	Code4Bench for C/C++code	0.69	-
CNN and random forest with boosting [17]	Bug report datasets (Mozilla, Eclipse, JBoss, OpenFOAM, and Firefox)	0.94, 0.95, 0.94, 0.98, 0.97	-
DNN [20]	Unified bug dataset (bug drediction dataset, PROMISE dataset, and GitHub bug dataset)	-	0.81
Our models (LSTM and GRU)	Unified bug dataset_balanced dataset (class-level)	0.93, 0.93	0.97, 0.96
Our models (LSTM and GRU)	Unified bug dataset_balanced dataset (file-level)	0.88, 0.88	0.93, 0.93

8. CONCLUSION

The process of software bug identification is one of the most common causes of wasted time and increased costs during the software lifecycle. Improving the quality and reliability of software systems can be achieved by detecting software bugs in the early stages of development. In the ever-changing software development landscape, ensuring precise bug prediction remains an indispensable factor in delivering dependable and top-notch software products. Embracing undersampling methods represents a pivotal stride towards realizing this objective, as they foster a balanced and unbiased training environment for RNNs. In this study, a novel method is introduced that combines LSTM and GRU with the undersampling technique to tackle the challenge of class imbalance and improve the accuracy of classifying defective or non-defective software modules. To validate the effectiveness of our models in predicting software bugs, we conducted a comparison using several performance measures. Python programming language with rich data science packages was chosen to implement the experiments. Using a public dataset is advised for ensuring the replicability, falsifiability, and verifiability of bug prediction models. Typically, public datasets that offer a wide range of metrics to explore are commonly utilized in bug prediction studies. Consequently, this study utilized combined datasets, comprising bugcatchers bug dataset, PROMISE, eclipse, bug prediction dataset, and GitHub bug dataset. The experimental findings indicated that our models exhibit promise and competitiveness, demonstrating the feasibility of utilizing RNNs in conjunction with sampling methods for SBP. We evaluated our proposed SBP method by comparing it with existing methods using various standard performance measures. The outcomes of the comparison indicate that the proposed method exhibits superior performance compared to the majority of existing state-of-the-art SBP methods across diverse datasets. Moving forward, our future work aims to evaluate the robustness of the proposed method on a wide range of datasets. Furthermore, we aspire to enhance the accuracy of SBP by incorporating additional ML techniques with various data-balancing methods.

ACKNOWLEDGEMENTS

The authors express their sincere gratitude for the financial support extended to this study by the Institute of Information Science, Faculty of Mechanical Engineering and Informatics, University of Miskolc.

REFERENCES

[1] H. Bani-Salameh, M. Sallam, and B. A. shboul, "A deep-learning-based bug priority prediction using RNN-LSTM neural networks," *E-Informatica Software Engineering Journal*, vol. 15, no. 1, pp. 29–45, 2021, doi: 10.37190/E-INF210102.

[2] A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, "SLDeep: statement-level software defect prediction using deep-learning model on static code features," *Expert Systems with Applications*, vol. 147, p. 113156, Jun. 2020, doi: 10.1016/j.eswa.2019.113156.

[3] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Information and Software Technology*, vol. 96, pp. 94–111, Apr. 2018, doi: 10.1016/j.infsof.2017.11.008.

[4] J. A. Fadhil, K. T. Wei, and K. S. Na, "Artificial intelligence for software engineering: an initial review on software bug detection and prediction," *Journal of Computer Science*, vol. 16, no. 12, pp. 1709–1717, Dec. 2020, doi: 10.3844/jcssp.2020.1709.1717.

[5] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 2, pp. 78–83, 2018, doi: 10.14569/IJACSA.2018.090212.

[6] E. Öztürk, K. U. Birant, and D. Birant, "An ordinal classification approach for software bug prediction," *Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi*, vol. 21, no. 62, pp. 533–544, May 2019, doi: 10.21205/deufmd.2019216218.

[7] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques," *Expert Systems with Applications*, vol. 144, p. 113085, Apr. 2020, doi: 10.1016/j.eswa.2019.113085.

[8] M. Samir, M. El-Ramly, and A. Kamel, "Investigating the use of deep neural networks for software defect prediction," in *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, Nov. 2019, vol. 2019-November, pp. 1–6, doi: 10.1109/AICCSA47632.2019.9035240.

[9] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," *Scientific Programming*, vol. 2019, pp. 1–14, Apr. 2019, doi: 10.1155/2019/6230953.

[10] Z. Yang and H. Qian, "Automated parameter tuning of artificial neural networks for software defect prediction," in *ACM International Conference Proceeding Series*, Jun. 2018, pp. 203–209, doi: 10.1145/3239576.3239622.

[11] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth, and T. Gyimóthy, "An automatically created novel bug dataset and its validation in bug prediction," *Journal of Systems and Software*, vol. 169, p. 110691, Nov. 2020, doi: 10.1016/j.jss.2020.110691.

[12] X. Zhou and L. Lu, "Defect prediction via LSTM based on sequence and tree structure," in *Proceedings - 2020 IEEE 20th International Conference on Software Quality, Reliability, and Security, QRS 2020*, Dec. 2020, pp. 366–373, doi: 10.1109/QRS51102.2020.00055.

[13] H. Tong, S. Wang, and G. Li, "Credibility based imbalance boosting method for software defect proneness prediction," *Applied Sciences (Switzerland)*, vol. 10, no. 22, pp. 1–29, Nov. 2020, doi: 10.3390/app10228059.

[14] X. Ye, F. Fang, J. Wu, R. Bunescu, and C. Liu, "Bug report classification using LSTM architecture for more accurate software defect locating," in *Proceedings - 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018*, Dec. 2019, pp. 1438–1445, doi: 10.1109/ICMLA.2018.00234.

[15] T. T. Khuat and M. H. Le, "Evaluation of sampling-based ensembles of classifiers on imbalanced data for software defect prediction problems," *SN Computer Science*, vol. 1, no. 2, p. 108, Mar. 2020, doi: 10.1007/s42979-020-0119-4.

[16] H. S. Munir, S. Ren, M. Mustafa, C. N. Siddique, and S. Qayyum, "Attention based GRU-LSTM for software defect prediction," *PLoS ONE*, vol. 16, no. 3 March, p. e0247444, Mar. 2021, doi: 10.1371/journal.pone.0247444.

[17] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B. G. Kang, and N. Chilamkurti, "A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting," *Sensors (Switzerland)*, vol. 19, no. 13, p. 2964, Jul. 2019, doi: 10.3390/s19132964.

[18] N. A. A. Khleel and K. Nehez, "Comprehensive study on machine learning techniques for software bug prediction," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 8. pp. 726–735, 2021, doi: 10.14569/IJACSA.2021.0120884.

[19] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: a semantic LSTM model for software defect prediction," *IEEE Access*, vol. 7, pp. 83812–83824, 2019, doi: 10.1109/ACCESS.2019.2925313.

[20] R. Ferenc, D. Bán, T. Grósz, and T. Gyimóthy, "Deep learning in static, metric-based bug prediction," *Array*, vol. 6, p. 100021, Jul. 2020, doi: 10.1016/j.array.2020.100021.

[21] Y. Verma, "Complete guide to bidirectional LSTM (with python codes)," Analytics India Magazine Pvt Ltd., 2021, [Online]. Available: https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-codes/.

[22] M. Z. Ansari, T. Ahmad, M. M. S. Beg, and F. Ahmad, "Hindi to English transliteration using multilayer gated recurrent units," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 27, no. 2, pp. 1083–1090, Aug. 2022, doi: 10.11591/ijeecs.v27.i2.pp1083-1090.

[23] N. A. A. Khleel and K. Nehéz, "A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method," *Journal of Intelligent Information Systems*, vol. 60, no. 3, pp. 673–707, Jun. 2023, doi: 10.1007/s10844-023-00793-1.

[24] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, "A public unified bug dataset for Java," in *ACM International Conference Proceeding Series*, Oct. 2018, pp. 12–21, doi: 10.1145/3273934.3273936.

[25] M. W. Thant and N. T. T. Aung, "Software defect prediction using hybrid approach," *2019 International Conference on Advanced Information Technologies, ICAIT 2019*, pp. 262–267, 2019, doi: 10.1109/AITC.2019.8921374.

[26] T. A. Assegie, R. L. Tulasi, V. Elanangai, and N. K. Kumar, "Exploring the performance of feature selection method using breast cancer dataset," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 25, no. 1, pp. 232–237, Jan. 2022, doi: 10.11591/ijeecs.v25.i1.pp232-237.

[27] S. Sharma and S. Kumar, "Analysis of ensemble models for aging related bug prediction in software systems," in *ICSOFT 2018 - Proceedings of the 13th International Conference on Software Technologies*, 2019, pp. 256–263, doi: 10.5220/0006847702560263.

[28] N. A. A. Khleel and K. Nehéz, "Deep convolutional neural network model for bad code smells detection based on oversampling method," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 26, no. 3, pp. 1725–1735, Jun. 2022, doi: 10.11591/ijeecs.v26.i3.pp1725-1735.

[29] T. T. Khuat and M. H. Le, "Ensemble learning for software fault prediction problem with imbalanced data," *International Journal of Electrical and Computer Engineering (IJEECS)*, vol. 9, no. 4, pp. 3241–3246, Aug. 2019, doi: 10.11591/ijece.v9i4.pp3241-3246.
[30] N. M. Mqadi, N. Naicker, and T. Adeliyi, "Solving misclassification of the credit card imbalance problem using near miss," *Mathematical Problems in Engineering*, vol. 2021, pp. 1–16, Jul. 2021, doi: 10.1155/2021/7194728.

BIOGRAPHIES OF AUTHORS

Nasraldeen Alnor Adam Khleel received a B.Sc. degree in Information Systems from the University of Kassala, Kassala-Sudan, in 2011. He got an M.Sc. degree in Software Engineering at Khartoum University, Khartoum-Sudan, in 2015. He is currently pursuing a Ph.D. at the University of Miskolc under the Faculty of Mechanical Engineering and Informatics, Miskolc-Hungary, since 2019. His primary research interests include Artificial Intelligence and Software Engineering. He can be contacted by email: nasr.alnor@uni-miskolc.hu.

Károly Nehéz received an M.Sc. degree in mechanical engineering from the University of Miskolc, Hungary, in 1997 and a Ph.D. degree in software engineering in 2003. He currently works as an associate professor at the Institute of Computer Science, head of the institute since 2019. His primary research interest is software engineering, although he has concurrent research in machine learning and artificial intelligence. He can be contacted by email: aitnehez@uni-miskolc.hu.