A Novel Approach for Software Defect Prediction using CNN and GRU Based on SMOTE Tomek Method

Nasraldeen Khleel (nasr.alnor@uni-miskolc.hu)
University of Miskolc

Research Article

Keywords: software defect prediction (SDP), software metrics, deep learning (DL), CNN, GRU, class imbalance, sampling techniques, SMOTE Tomek

Posted Date: November 30th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2305042/v2

A Novel Approach for Software Defect Prediction using CNN and GRU Based on SMOTE Tomek Method

Abstract

Software defect prediction (SDP) plays an important role in enhancing the quality of software projects and reducing maintenance-based risks through the ability to detect defective software components. SDP refers to the methods that use historical defect data to build the relationship between software metrics and software defects. Several prediction models such as machine learning (ML), deep learning (DL) have been developed and adopted to recognize defect in software modules and many methodologies and frameworks have been presented. One of the most difficult problems that these models face in binary classification is the classes imbalance. When the distribution of classes is unbalanced, the accuracy may be high, but the model cannot recognize data instances in the minority class, this will lead to weak classifications. So far, few research have been done in the previous studies that address the problem of class imbalance in SDP. To address the class imbalance problem, we propose a novel SDP approach based on convolutional neural network (CNN) and gated recurrent unit (GRU) combined with synthetic minority oversampling technique plus Tomek link (SMOTE Tomek). To establish the efficiency of the proposed models, the experiments have been conducted on benchmark datasets which obtained from the PROMISE repository and the experimental results have been compared and evaluated in terms of accuracy, precision, recall, f-measure, the area under the ROC curve (AUC), the area under the precision-recall curve (AUCPR), mean square error (MSE). The average accuracy of the proposed models on the original datasets were 89% for CNN and 87% for GRU, while the average accuracy of the proposed models on the balanced datasets were 94% for CNN and 92% for GRU. The results showed that the proposed models on the balanced datasets improves the average accuracy by 5% for both models compared to original datasets. This indicates the positive effects of combining ML techniques with data balancing methods on the performance of defect prediction regarding datasets with imbalanced class distributions.

Keywords: software defect prediction (SDP), software metrics, deep learning (DL), CNN, GRU, class imbalance, sampling techniques, SMOTE Tomek.

1. Introduction

Determining defects in source codes usually is a difficult task due to the huge code base of software projects. The importance and challenges of defect prediction have made it an active research area in software engineering [1]. Defects in software are often difficult to detect or identify and developers spend a significant amount of time locating and fixing them. The software life cycle includes many activities for identifying source code defects, such as design reviews, code inspections, integration tests, functions testing, and unit tests, etc. [2]. Early detection of a defect in software projects during the development phase helps allocate testing resources reasonably, determine the testing priority of different software modules and improves the effectiveness of the software development process [3]. SDP is a process for predicting source code defects using tools or techniques based on historical data. SDP can be divided into with-in project defect prediction (WPDP), cross project defect prediction (CPDP) for similar dataset, and cross project defect prediction (CPDP) for heterogeneous dataset [4, 5]. In this study, we develop our models based on with-in project defect prediction (WPDP) approach. In the WPDP approach, a prediction model can be built based on collecting historical data from a software project and predicting defects in the same project. WPDP performs best if there is enough quantity of historical data available to train model [6]. There are two ways in which previous studies have attempted to build accurate SDP models: the first approach is to manually design new features or new sets of features to represent defects more effectively, while the second approach involves applying new and improved machine learning based classifiers. Several models have been proposed for SDP based on the second approach (machine learning based classifiers), but there is still a need to develop accurate defect detection models or detectors, as well as robust software metrics to distinguish between defective and non-defective software modules. Latest studies leverage manually designed software metrics such as Halstead features, McCabe features, CK features, MOOD features, etc. to build classifiers. Recently, DL algorithms have been adopted to improve research tasks in software engineering, especially in the field of SDP [6, 7]. DL algorithms differ from classical artificial neural networks in one key aspect, namely that

they contain many hidden layers [8, 9]. DL is a type of ML that allows computational models consisting of multiple processing layers to learn data representations with multiple levels of abstraction. DL architecture has been widely applied in many fields and used to solve many detections, classification, and prediction problems [10]. DL transforms initial "low-level" feature representation into "high-level" through multi-layer processing; it can complete complex classification tasks with simple models. Also, it has the capabilities of feature learning and representational learning. Therefore, it can achieve better performance while being flexible [11]. DL has drawn more and more attention, because of its powerful feature learning capability, and has been successfully used in many domains, such as speech recognition, image classification, etc. CNN and GRU models are two of the most famous DL architectures [2]. When there is an uneven distribution of classes in the training data set, this indicates that this data is imbalanced. Imbalanced classes classification biases performance towards majority numbered class in case of a binary application. Most ML techniques can predict better when the number of instances of each classis are roughly equal. so, data imbalance is the biggest problem faced by ML techniques. This problem severely hinders the efficiency of these models and produces unbalanced false-positive and false-negative results. Usually, defective software databases consist of imbalanced data which. This study selects imbalanced datasets from the public PROMISE repository for experimental purposes [12, 13, 14], so this motivates a solution such as applying the sampling methods and there is great interest in building unbiased classifiers that start from imbalanced software defect data. Although several experiments in the previous studies [12, 15, 16, 17] are conducted based on these datasets using many ML models, very few of them are based on CNN and GRU. Even there is no experiment using CNN and GRU combined with oversampling techniques in the literature. To bridge these gaps, the novelty and main contributions of our work are summarized as follows:

1. In this study, we propose a novel approach that combines CNN and GRU with SMOTE Tomek technique to predict software defects.
2. We evaluate the performance of the proposed approach and compares it with the traditional ML model (RF) as baseline model and also compares it with the existing approaches used in SDP.

The structure of this paper is organized as follows. Section 2 presents a discussion on related work. Section 3 presents background on the topics of CNN, and GRU. Section 4 presents the hypothesis and research questions. After that, our research methodology is presented in Section 5. Section 6 presents the experimental results. Section 7 presents the discussion. Section 8 presents implication of the findings. Section 9 presents threats to validity followed by conclusions in the last section.

2. Related work

The prediction of defects in software systems is very important and there is great interest in the development of novel high-performance software defect predictors. The purpose of SDP models is to improve the quality of software application systems [15]. Many models have been constructed to recognize the defects in software modules using artificial intelligence and statistical methods [1, 18, 19, 20, 21, 22].

JIEHAN. Kun Zhu et al. [10] proposed a novel just-in-time defect prediction model named DAECNN-JDP based on denoising autoencoder and CNN. The model was evaluated based on six large open-source projects and compared with 11 baseline models. Experimental results showed that the proposed model outperforms these baseline models. Jiehan Deng et al. [12] proposed a novel LSTM method to perform SDP, their method can automatically learn semantic and contextual information from the program's ASTs. Experiment was performed on several open-source projects, the results showed that the proposed LSTM method is superior to the state-of-the-art methods. Thanh Tung Khuat and My Hanh Le. [15] conducted an empirical study to evaluate the importance of sampling techniques in SDP. Experimental results indicated the positive effects of combining sampling techniques with ensemble learning models. This method solved the class-imbalance problem and achieve high prediction accuracy. Diana-Lucia Miholca et al. [16] presented a supervised classification approach named (HyGRAR). It is a non-linear hybrid model that combines gradual relational association rule mining and artificial neural networks to predict software defects. Experiments were conducted using 10 open-source datasets. Experimental results showed the excellent performance of the proposed classifier and better performance than most of the previously proposed classifiers. This method achieved high prediction accuracy. Racharla Suresh Kumar and Bachala Satyanarayana [17] developed a Hybrid Neural Network model with object oriented and CK metrics for software fault prediction. Adaptive Genetic Algorithm has been used for ANN optimization. The proposed model has been tested with PROMISE data sets. Experimental results showed the better performance compared to major existing schemes. Hao Xu et al. [18] proposed a novel approach based on the transfer CNN model to mine the transferable semantic features for CPDP tasks. Experiments were conducted based on 10 benchmark projects with 90 pairs of CPDP tasks. Their results showed that the proposed model is superior to the reference methods. Cong Pan et al. [20] proposed an improved CNN model for WPDP and compare the results of

the experiment with those of existing CNN studies. Experiment was performed based on a 30-repetition holdout validation and a 10 * 10 cross-validation. Their results showed that CNN model outperformed the state-of-the-art ML models significantly for WPDP. Hafiz Shahbaz Munir et al. [22] proposed a new framework based on GRU and LSTM for software defect prediction. Experiments were evaluated based on 119,989 C/C++ programs in Code4Bench. The proposed method was compared with different approaches. Experimental results demonstrated that the proposed method has a better performance in terms of recall, precision, accuracy and F1 metrics. Jian Li et al. [23] proposed a framework based on the programs' Abstract Syntax Trees called Defect Prediction via CNN. The model was evaluated based on seven open-source projects in terms of f-measure. Experimental results showed that on average, the model improves the state-of-the-art method by 12%. Ashima Kukkar et al. [24] proposed a novel DL model for multiclass severity classification called bug severity classification using a CNN and Random Forest with Boosting based on five open-source projects. Their results prove that the proposed model enhances the performance of bug severity classification over state-of-the-art techniques. Sushant Kumar Pandey et al. [25] proposed a new method using deep representation and ensemble learning with sampling techniques for software bug prediction and dealing with the class imbalance problem. Experiment was performed based on 12 data sets from the PROMISE repository. Evaluation results showed that the proposed method outperformed other state-of-the-art techniques. This method solved the class-imbalance problem and achieve high prediction accuracy. Zhao Yang and Hongbing Qian [26] proposed ANNs model, which automated parameter tuning technique to optimize the defect prediction models. The model was evaluated based on 30 datasets are downloaded from the Tera-PROMISE Repository. Their results showed that the performance of the proposed model have been indeed improved after tuning parameter settings. The authors suggested that researchers should pay attention to tuning parameter settings by Caret for ANNs instead of using suboptimal default settings if they select ANNs for training models in the future defect prediction studies. Linchang Zhao et al. [27] proposed a novel SDP model, called Siamese parallel fully-connected networks, which combines the advantages of Siamese networks and DL. The authors compared the proposed model with the state-of-the-art SDP models using six datasets from the NASA repository. Experimental results showed that the proposed model contributes to significantly higher performance compared with benchmarked SDP approaches. Ahmed Bahaa Farid et al. [28] proposed a hybrid model using bidirectional long short-term memory and CNN to predict software defects. The proposed model was evaluated using seven open-source Java projects from the PROMISE dataset. Their results showed that the proposed model is accurate for predicting software defects. This method used different performance measures and achieve high prediction accuracy. Guisheng Fan et al. [29] presented a framework named SDP via an attention-based RNN. The models were evaluated based on an open-source Apache Java project, using F1-measure and AUC. Experimental results demonstrated that the proposed model improves the F1 measure by 14% and AUC by 7% compared with the state-of-the-art methods. HONGLIANG LIANG al. [30] proposed Seml, a novel framework that combines word embedding and LSTM for software defect prediction. The model was evaluated based on eight open-source projects. The experimental results showed that the Seml outperforms three state-of-the-art defect prediction approaches on most of the datasets for both WPDP and CPDP.

After reviewing some previous studies in SDP, we noticed that most of the proposed methods ignore the class imbalance problem. According to studies that dealt with the problem of class imbalance and handled it [15, 25], the authors point out that the data balancing methods have an important role in improving SDP accuracy. So, the major point from the recent studies is that ML combined with data balancing methods can improve and increase prediction accuracy. Therefore, this work focuses on addressing the class imbalance problem for improving the efficiency of our proposed models.

3. Background

In this section, we present the background of the topics of convolutional neural network and gated recurrent unit.

3.1.Convolutional neural network

Convolutional neural network (CNN) is a special type of deep neural network or a class of convolutional feedforward neural networks used to process data that has a known, grid-like topology. It is constructed to mimic the visual perception of biological processes and can be used for both supervised learning and unsupervised learning. CNN has been tremendously successful in practical applications, including speech recognition, image classification, and natural language processing [31]. CNN model is inspired by the typical CNN architecture used in image classification and consists of a feature extraction part and a classification part as shown in figure 1. These parts consist of multiple layers are convolution, batch normalization, and maximum merge layers. These layers constitute the hidden layer of the architecture. The convolutional layer performs convolution operations based on the specified filter and kernel parameters and calculates the network weights to the next layer, while the maximum pooling layer achieves a reduction in the dimension of the feature space. Batch normalization is used to mitigate

the effect of different input distributions for each training mini-batch to improve training. Activation functions enable the training of the CNN model in a fast and accurate manner. There are many activation functions used in CNN such as Sigmoid, Rectified Linear Unit (ReLU), and hyperbolic tangent (Tanh) [11, 20,]. In this model, we used two activation functions, ReLU function for the input and hidden layers and the Sigmoid function for the output layer as shown in the equations below.

$$h_i^m = ReLU(W_i^{m-1} \times V_i^{m-1} + b^{m-1}) \quad (1)$$

Where h_i^m represents the convolutional layer, W_i^{m-1} represents the weights of neurons, V_i^{m-1} represents the nodes, and b^{m-1} represents the bias layer.

$$S(x) = \frac{1}{1+e^{-\sum_k W_i + X_i + b}} \quad (2)$$

Where X_i represents the input, W_i is the weight of the input and b is the bias.

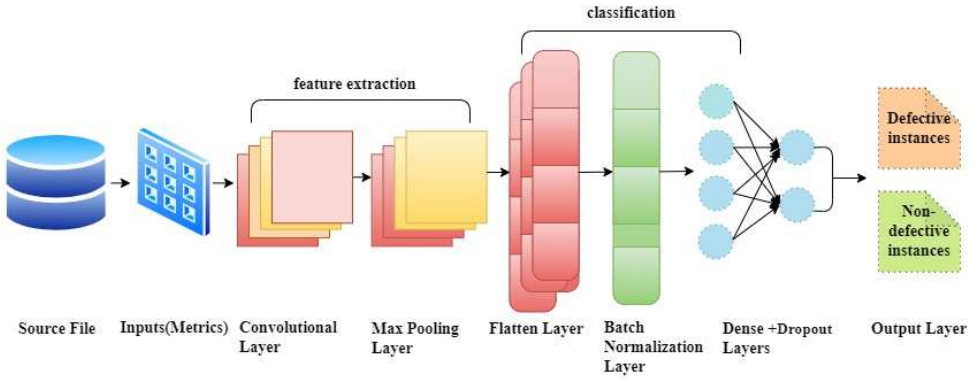

Figure 1. CNN Model for SDP [20]

3.2.Gated recurrent unit

Gated recurrent unit (GRU) network is one of the optimized structures of the recurrent neural network (RNN). Due to the problem of long-term dependencies that arise when the input sequence is too long, RNN cannot guarantee the long-term nonlinear relationship. This means that there is a gradient vanishing and gradient explosion phenomena at the learning sequence. Many optimization theories and improved algorithms have been introduced to solve this problem such as GRU networks, long short-term memory networks, Bidirectional long short-term memory, echo state networks, and independent RNN [11]. The goal of the GRU network is to solve the long-term dependence and gradient disappearance problem of RNN. A GRU networks is similar to long short-term memory networks with a forget gate, but it has fewer parameters than long short-term memory (LSTM) and uses an update gate and a reset gate as shown in figure 2. The update gate helps the model to determine how much of the past information (from previous time steps) needs to be passed along to the future and the reset gate helps the model to decide how much of the past information to forget [32]. The update gate model in the GRU network is calculated as shown in the equation below.

$$z(t) = \sigma(W(z).[h(t-1), x(t)]) \quad (3)$$

the $z(t)$ is the update gate function, $h(t-1)$ is the output of the previous neuron, $x(t)$ is the input of the current neuron, $W(z)$ represents the weight of the update gate, and σ represents the sigmoid function. The reset gate model in the GRU neural networks is calculated as shown in the equation below.

$$r(t) = \sigma(W(r).[h(t-1), x(t)]) \quad (4)$$

$r(t)$ is the reset gate function, $h(t-1)$ represents the output of the previous neuron, $x(t)$ is the input of the current neuron, $W(r)$ represents the weight of the reset gate, and σ is the sigmoid function. The output value of the GRU hidden layer is shown in the equation below.

$$\check{h}(t) = \tanh\left(W\check{h}.[rt * h(t-1), x(t)]\right) \quad (5)$$

$\check{h}(t)$ is the output value to be determined in this neuron, $h(t-1)$ is the output of the previous neuron, $x(t)$ represents the input of the current neuron, $W\check{h}$ represents the weight of the update gate, and tanh () is the hyperbolic tangent function. rt is used to control how much memory needs to be retained. The hidden layer information of the last output as shown in the equation below.

$$h(t) = \big(1 - z(t)\big) * h(t-1) + z(t) * \check{h}(t) \quad (6)$$

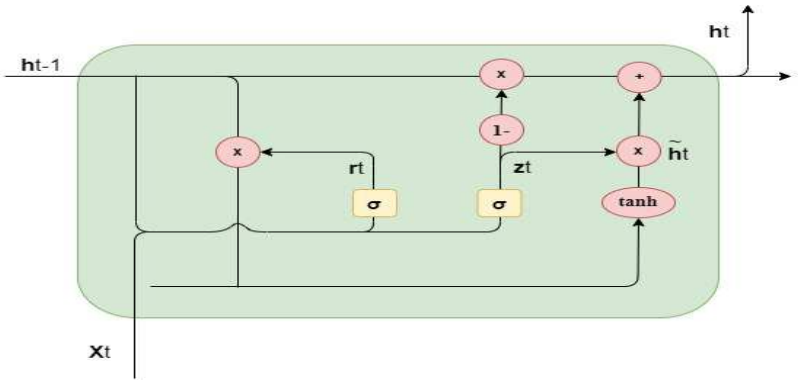

Figure 2. GRU Model for SDP [33]

4. Hypothesis and research questions

Our hypothesis in this study is if data balancing methods are applied to balance the original data sets, the classification performance of the proposed models will be better in SDP. To investigate our hypothesis, we used a paired t-test to find out whether there was a statistically significant difference between our models on the original and balanced datasets. To statistically prove the validity of the impact of data balancing methods on the performance of ML algorithms, the hypothesis is formed as follows:

H1: There is no difference in the accuracy of models when there are no data balancing methods and when the data balancing methods are used.

H2: There exists a difference in the accuracy of models when there are no data balancing methods and when the data balancing methods are used.

Based on our hypothesis, the purpose of this study is to understand the impact of data balancing methods on the performance of ML algorithms in SDP. In particular, we aim to address the following research questions.

RQ1: Do data balancing methods improve the accuracy of ML models in SDP?

This RQ aims at investigate data balancing methods in improving the accuracy of ML models in SDP.

RQ2: Does the proposed approach outperform the state-of-the-art approaches in SDP?

This RQ aims at investigate the performance of the proposed approach in SDP compared against the state-of-the-art approaches.

5. Methodology

This study proposed a novel approach based on CNN and GRU with SMOTE Tomek technique for predicting defective software. The experiments were performed on public benchmark datasets. A series of steps have been taken and described such as benchmark datasets used, software metrics used, data pre-processing, features selection, dataset balancing, performance measures used. Figure 3 illustrates the whole workflow of the proposed SDP approach where each step is described in the following sections.

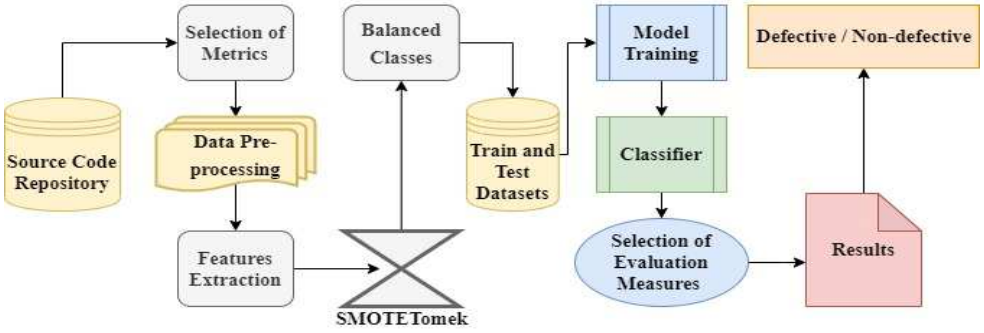

Figure 3. Overview of the proposed approach for SDP

5.1.Benchmark datasets

To verify the validity of the proposed approach, we selected six open-source Java projects from PROMISE dataset [34]. The source code and corresponding PROMISE data for all six projects are public [12, 28, 35]. These projects cover several applications such as XML parser, text search engine library, and data transport adapters, and these projects have traditional static metrics for each Java file. The selection of projects was based on the percentage of data imbalance in them. To guarantee the generality of the evaluation results, experimental datasets consist of projects with different sizes and defect rates (in the six projects, the maximum number of instances is 965, and the minimum number of instances is 205. In addition, the minimum defect rate is 2.23% and the maximum defect rate is 92.19%). Table 1 shows the essential information of selected projects, including project name, project version, number of instances, and defect rate or the percentage of defective instances.

Table 1 Description of the PROMISE datasets that we have chosen

Project Name	Project Version	# Of Instances	Defect Rate %
ant	1.7	745	22.28%
camel	1.6	965	19.48%
ivy	2.0	352	11.36%
jedit	4.3	492	2.23%
log4j	1.2	205	92.19%
xerces	1.4	588	74.31%

5.2.Defect prediction metrics

Metrics play the most important role to build a prediction model that aims to improve software quality by predicting as many software defects as possible. Metrics in the context of SDP are considered as independent variables. Many previous researchers have pointed out that there is a relationship between software metrics and defect predictions [3]. In general, the type of software metrics used in SDP can be divided into static code metrics and process metrics. Static code metrics represent how the source code is complex and include information about the software codes depending on the type of coding while process metrics represent how the development process is complex and constituted from some values such as developer count, time, effort, and cost [5]. In 1976, McCabe released the first standard of static code metrics and in 1977, Halstead developed a new metric standard. Some practitioners use this metric as an indicator of the level of defect-proneness. With the evolution of programming languages, object-oriented metrics standards were developed such as Lorenz, Kidd, Chi- damber, and Kemerer [36, 37]. The primary studies use software metrics as independent variables for measuring the quality of software modules. Several researchers used McCabe and Halstead metrics as independent variables in SDP. This study relies on the McCabe and Halstead metrics as independent variables. Table 2 shows the traditional static code metrics contained in the PROMISE repository, and for the descriptions, the readers are referred to [35].

Table 2 List of 20 traditional static metrics of PROMISE. Descriptions were given in [35].

Attribute	Description
dit	The maximum distance from a given class to the root of an inheritance tree
noc	Number of children of a given class in an inheritance tree
cbo	Number of classes that are coupled to a given class
rfc	Number of distinct methods invoked by code in a given class

lcom	Number of method pairs in a class that do not share access to any class attributes
lcom3	Another type of the lcom metric proposed by Henderson–Sellers
npm	Number of public methods in a given class
loc	Number of lines of code in a given class
dam	The ratio of the number of private/protected attributes to the total number of attributes in a given class
moa	Number of attributes in a given class that are of user-defined types
mfa	Number of methods inherited by a given class divided by the total number of methods that can be accessed by the member methods of the given class
cam	The ratio of the sum of the number of different parameter types of every method in a given class to the product of the number of methods in the given class and the number of different method parameter types in the whole class
ic	Number of parent classes that a given class is coupled to
cbm	Total number of new or overwritten methods that all inherited methods in a given class are coupled to
amc	The average size of methods in a given class
ca	Afferent coupling, which measures the number of classes that depend on a given class
ce	Efferent coupling, which measures the number of classes that a given class depends on
max_cc	The maximum McCabe's cyclomatic complexity (CC) score of methods in a given class
avg_cc	The arithmetic mean of McCabe's cyclomatic complexity (CC) scores of methods in a given class

5.3.Data pre-processing and features selection

Pre-processing the collected data is one of the important stages before constructing the model. To generate a good model, the quality of data needs to be considered. Not all data collected is suitable for training and model building. Anyhow the inputs will greatly impact the performance of the model and later moreover affect the output. Data pre-processing is known as a group of techniques that are applied to the data to improve the quality of the data before model building for the purpose of removing noise and unwanted outliers from the data set, dealing with missing values, feature type conversion, etc. [16, 27, 28] In addition, normalization is necessary to convert the values into scaled values (scaling of the data in numeric variables in the range of 0 to 1) to increase the efficiency of the model. Therefore, the data set was normalized using Min–Max normalization. The formula for calculating normalized score can be described by (7). Feature selection is a crucial step to select the most discriminative features from the list of features using appropriate feature selection methods. The goal of feature selection is to select the features which are more relevant to the target class from high-dimensional features and remove the features which are redundant and uncorrelated [5, 38]. Feature extraction facilitates the conversion of pre-processed data into a form that the classification engine can use [39, 40]. Feature selection methods are categorized into three categories (i) Filter methods: These methods are model agnostic, i.e., variables are selected independently of ML algorithms. These methods are faster and less computationally expensive, (ii) Wrapper methods: These methods are greedy and choose best feature subsets in each iteration according to ML algorithms. It is a continuous process of finding a feature subset. These methods are very computationally expensive and often unrealistic if the feature space is vast, (iii) Embedded methods: in these methods, feature selection is a part of building ML algorithms. These methods select the best possible feature subset as per the ML model to be implemented [41]. In this study, we applied embedded methods because it is faster and less computationally expensive than other methods and it fits ML models and feature scaling technique was applied to make the output the same standard.

$$x_i = (x_i - X\,min)/(X\,max - X\,min) \quad (7)$$

Where *max(x)* and *min(x)* represent the maximum and minimum value of the attribute *x* respectively.

5.4.Class imbalance and sampling techniques

Class imbalance in classification models represents those situations, where the number of examples of one class is much smaller than other classes. The class with the higher size of data is the majority class, while the class with a smaller size is considered as minority class [42]. Class imbalance is an important special of the software defects

data, which consists of only a few defective instances and there are large number of non-defective instances. Hence, the class imbalance problem often could cause misclassification of the instances in the minority class. The datasets used in our study suffer from common problem in SDP studies, which is class imbalance [12, 13, 14]. The reference datasets are not properly distributed which shows a lack in the real distribution of learning instances as shown in Table 1. We manage this problem by modifying the original datasets to increase the realism of the data [36]. The most common methods used to deal with distributions of unbalanced classes are sampling techniques, might be divided into two categories: oversampling techniques and under-sampling techniques [43, 44]. Oversampling techniques supplements instances of the minority class to the dataset, while the under-sampling techniques eliminate samples of the majority class for the goal of obtaining a balanced dataset [15]. The SMOTE is a classic oversampling technique that increases the samples while Tomek Link is an under-sampling technique that decreases the samples. SMOTE Tomek is a new technique that was applied using the library from imbalanced learn, which combines the synthetic minority oversampling technique (SMOTE) function for oversampling as well as the Tomek Link function for under-sampling [45, 46, 47]. In this study, we used SMOTE Tomek technique to deal with the class imbalance problem. Figure 4 shows the distribution of learning instances over the original and balanced data sets.

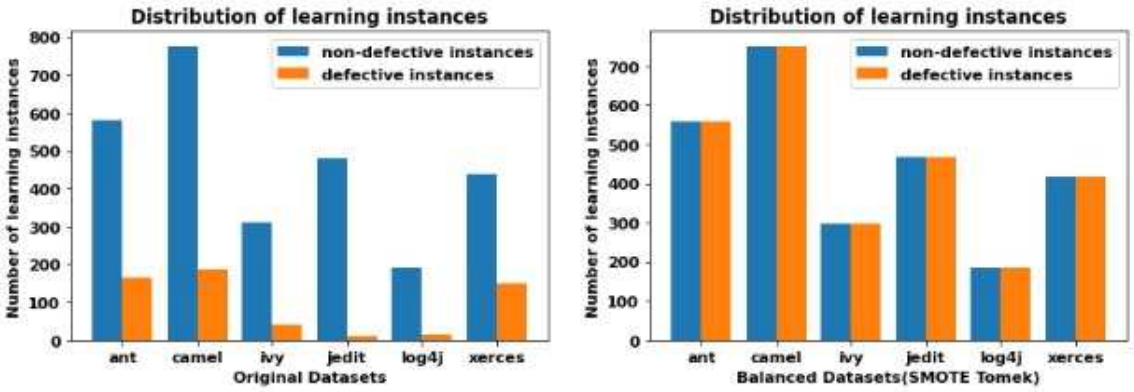

Figure 4. Distribution of learning instances over the original and balanced data sets

5.5.Models building and evaluation

Different ML and DL algorithms have been used to build defect prediction models and each algorithm has its own benefits. Most studies of SDP divide the data into two sets: a training set and a test set. The training set is used to train the model, whereas the testing set is used to evaluate the performance of the defect's prediction model. Once a defects prediction model is built, its performance needs to be evaluated [48]. Implementation framework of our models: we use Keras as a high-level API based on TensorFlow to build our models for simplicity and correctness, training is performed with 80% of the dataset (random selection of features), while the remaining 20% is used for validation, each model was developed separately with different parameters as shown in Table 3. We evaluate the performance of our proposed models based on a set of common performance measures such as confusion matrices, AUC, AUCPR and MSE as a Loss function. AUC, which plots the false positive rate on the x-axis and true positive rate on the y-axis over all possible classification thresholds. AUCPR is a curve plots the precision versus the recall or a single number summary of the information in the precision-recall curve. MSE is a metric which measures the amount of error the model. It assesses the average squared difference between the actual and predicted values. A confusion matrix is a specific table used to measure the performance of a model. A confusion matrix summarizes the results of the testing algorithm and presents a report of True Positive (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) as shown in Table 4.

Table 3. Parameter settings of the models

Parameters	**Models**	
	CNN	**GRU**
Layers. GRU	-	100
Activation function	ReLU + Sigmoid	Tanh + Sigmoid
Dropouts	0.2	0.2
Dense	10, 1	1
Optimizer	Adam	Adam
Learning Rate	0.01	0.01

Loos Function	Mean squared error (MSE)	Mean squared error (MSE)
Batch Size	25	64
Epochs	100	100
Validation Split	0.1	0.1
Verbose	-	1

Table 4. Confusion matrix

	Actual	
Predicted	Defective	Non-defective
Defective	TN	FP
Not defective	FN	TP

Accuracy = (TP+TN) / (TP+FP+FN+TN) (8)

Precision = TP / (TP+FP) (9)

Recall = TP / (TP + FN) (10)

F-Measure = (2 * Recall *Precision) / (Recall + Precision) (11)

$$\text{AUC} = \frac{\sum_{ins_i \in Positive\ Class} \text{rank}(ins_i) - \frac{M(M+1)}{2}}{M \cdot N} \quad (12)$$

Where $\sum_{ins_i \in Positive\ Class} \text{rank}(ins_i)$ is the sum of ranks of all positive samples, M and N are the number of positive samples and negative samples, respectively.

$$\text{AUCPR} = \int_0^1 \text{Precision}(\text{Recall}) \, d(\text{Recall}) \quad (13)$$

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (x(i) - y(i))^2 \quad (14)$$

Where n is the number of the observations, x(i) is the actual value, y(i) is the observed or predicted value for the i^{th} observation.

6. Experimental results

In this section, we evaluate the efficiency of our proposed models. The experimental environment was based on a Python environment and used data from the same project for both training and testing. The study has considered six open-source datasets for experimental analysis using CNN and GRU. We also did experiment by using traditional ML model (RF) as baseline model and compared it with our proposed models.

To answer the research questions - RQ1, the performance of the prediction models is reported in Tables 5 to 9, and Figures 5 to 21 mentioned below.

Table 5 presents the results of CNN model on the original datasets in terms of accuracy, precision, recall, f-measure, AUC, AUCPR and MSE. We notice that the accuracy values of the model range from 0.80 to 0.97, the precision values of the model range from 0.00 to 0.95, the recall values of the model range from 0.00 to 1.00, the f-measure values of the model range from 0.00 to 0.97, the AUC values of the model range from 0.53 to 0.97, the AUCPR values of the model range from 0.03 to 0.98, and the MSE values of the model range from 0.030 to 0.150.

Table 6 presents the results of CNN model on the balanced datasets in terms of accuracy, precision, recall, f-measure, AUC, AUCPR and MSE. We notice that the accuracy values of the model range from 0.86 to 0.99, the precision values of the model range from 0.84 to 1.00, the recall values of the model range from 0.88 to 1.00, the f-measure values of the model range from 0.87 to 0.99, the AUC values of the model range from 0.91 to 0.99, the AUCPR values of the model range from 0.91 to 0.99, and the MSE values of the model range from 0.018 to 0.114.

Table 7 presents the results of GRU model on the original datasets in terms of accuracy, precision, recall, f-measure, AUC, AUCPR and MSE. We notice that the accuracy values of the model range from 0.76 to 0.96, the precision values of the model range from 0.00 to 0.95, the recall values of the model range from 0.00 to 0.97, the f-measure values of the model range from 0.00 to 0.96, the AUC values of the model range from 0.27 to 0.88, the AUCPR values of the model range from 0.02 to 0.96, and the MSE values of the model range from 0.037 to 0.190.

Table 8 presents the results of GRU model on the balanced datasets in terms of accuracy, precision, recall, f-measure, AUC, AUCPR and MSE. We notice that the accuracy values of the model range from 0.84 to 0.99, the precision values of the model range from 0.83 to 1.00, the recall values of the model range from 0.85 to 1.00, the

f-measure values of the model range from 0.84 to 0.99, the AUC values of the model range from 0.90 to 0.99, the AUCPR values of the model range from 0.88 to 0.99, and the MSE values of the model range from 0.013 to 0.123.
Table 9 presents the statistical analysis results (paired t-test) of proposed models on the original and balanced datasets in terms of mean, Standard Deviation (STD), min, max and P value. We notice that the mean values of CNN model are 0.89 on the original datasets and 0.94 on the balanced datasets, while the mean values of GRU model are 0.87 on the original datasets and 0.92 on the balanced datasets. The STD values of CNN model are 0.07 on the original datasets and 0.05 on the balanced datasets, while the STD values of GRU model are 0.07 on the original datasets and 0.05 on the balanced datasets. The Min values of CNN model are 0.80 on the original datasets and 0.86 on the balanced datasets, while the Min values of GRU model are 0.76 on the original datasets and 0.84 on the balanced datasets. The Max values of CNN model are 0.97 on the original datasets and 0.99 on the balanced datasets, while the Max values of GRU model are 0.96 on the original datasets and 0.99 on the balanced datasets. The P value of CNN model is 0.004 based on the original and balanced datasets, while the P value of GRU model is 0.004 based on the original and balanced datasets. Based on the P value of both models on the original and balanced data sets, we note that the P value is less than 0.05, and this indicates that there is a difference between the results of the models on the original and balanced data sets.
Figure 5 bellow shows the Box plots for the performance measures (Accuracy, Precision, Recall, F-measure, AUC, AUCPR and MSE) on the original and balanced datasets. The averages of (Accuracy, Precision, Recall, F-measure, AUC, AUCPR and MSE) of CNN model on the original datasets are 0.89, 0.57, 0.51, 0.53, 0.72, 0.54 and 0.086, respectively. The averages of (Accuracy, Precision, Recall, F-measure, AUC, AUCPR and MSE) of CNN model on the balanced data sets are 0.94, 0.93, 0.95, 0.94, 0.96, 0.95 and 0.051, respectively. The averages of (Accuracy, Precision, Recall, F-measure, AUC, AUCPR and MSE) of GRU model on the original datasets are 0.87, 0.52, 0.48, 0.50, 0.63, 0.49 and 0.113, respectively. The averages of (Accuracy, Precision, Recall, F-measure, AUC, AUCPR and MSE) of GRU model on the balanced data sets are 0.92, 0.92, 0.92, 0.92, 0.95, 0.95 and 0.067, respectively.
Figures 6 to 13 below show the training and validation accuracy and training and validation loss of the models on the original and balanced datasets. As shown in the figures, the accuracy of training and validation increases and the loss decreases with increasing epochs. Regarding the high accuracy and low loss obtained by the proposed models, we note that the models are well trained and validated.
Figures 14 to 17 below show the ROC curves of the models on the original and balanced datasets. The best AUC obtained by CNN model in the original data sets is 97% on the xerces data set, while the worst AUC is 53% on the log4j data set. The best AUC obtained by CNN model in the balanced data sets is 99% on the log4j and xerces data sets, while the worst AUC is 91% on the camel data set. The best AUC obtained by GRU model in the original data sets is 88% on the xerces data set, while the worst AUC is 27% on the jedit data set. The best AUC obtained by GRU model in the balanced data sets is 99% on the jedit data set, while the worst AUC is 90% on the camel data set.
Figures 18 to 21 below show the AUCPR of the models on the original and balanced datasets. The best AUCPR obtained by CNN model in the original data sets is 98% on the xerces data set, while the worst AUCPR is 0.03% on the jedit data set. The best AUCPR obtained by CNN model in the balanced data sets is 99% on the log4j and xerces data sets, while the worst AUCPR is 91% on the camel data set. The best AUCPR obtained by GRU model in the original data sets is 96% on the log4j data set, while the worst AUCPR is 0.02% on the jedit data set. The best AUCPR obtained by GRU model in the balanced data sets is 99% on the ivy and jedit data sets, while the worst AUCPR is 88% on the camel data set.
After comparing the results obtained by the proposed models on the original datasets with results obtained by the proposed models on the balanced datasets, as shown in the tables and figures, we note that the models got good scores on the balanced datasets and the results improved further due to balancing, which indicated that the proposed models performed well and data balancing techniques play an important role in improving the accuracy of the models.

Table 5. Performance analysis for proposed CNN Model-Original Data sets

Datasets	**Performance Measures**						
	Accuracy	**Precision**	**Recall**	**F-Measure**	**AUC**	**AUCPR**	**MSE**
ant	0.81	0.54	0.47	0.50	0.78	0.49	0.150
camel	0.80	0.44	0.19	0.27	0.71	0.37	0.149
ivy	0.89	0.57	0.44	0.50	0.80	0.47	0.114
jedit	0.97	0.00	0.00	0.00	0.58	0.03	0.030
log4j	0.95	0.95	1.00	0.97	0.53	0.94	0.031
xerces	0.94	0.95	0.98	0.96	0.97	0.98	0.044
Averages	**0.89**	**0.57**	**0.51**	**0.53**	**0.72**	**0.54**	**0.086**

Table 6. Performance analysis for proposed CNN Model-Balanced Datasets

Datasets	Performance Measures						
	Accuracy	**Precision**	**Recall**	**F-Measure**	**AUC**	**AUCPR**	**MSE**
ant	0.89	0.90	0.88	0.89	0.94	0.93	0.091
camel	0.86	0.84	0.91	0.87	0.91	0.91	0.114
ivy	0.95	0.92	0.98	0.95	0.97	0.95	0.042
jedit	0.98	0.96	1.00	0.98	0.98	0.95	0.019
log4j	0.99	1.00	0.98	0.99	0.99	0.99	0.018
xerces	0.98	0.99	0.97	0.98	0.99	0.99	0.025
Averages	**0.94**	**0.93**	**0.95**	**0.94**	**0.96**	**0.95**	**0.051**

Table 7. Performance analysis for proposed GRU Model-Original Data sets

Datasets	Performance Measures						
	Accuracy	**Precision**	**Recall**	**F-Measure**	**AUC**	**AUCPR**	**MSE**
ant	0.80	0.50	0.47	0.48	0.72	0.47	0.176
camel	0.76	0.22	0.11	0.15	0.57	0.22	0.190
ivy	0.89	0.57	0.44	0.50	0.79	0.39	0.128
jedit	0.96	0.00	0.00	0.00	0.27	0.02	0.037
log4j	0.93	0.95	0.97	0.96	0.57	0.96	0.062
xerces	0.90	0.93	0.93	0.93	0.88	0.91	0.090
Averages	**0.87**	**0.52**	**0.48**	**0.50**	**0.63**	**0.49**	**0.113**

Table 8. Performance analysis for proposed GRU Model-Balanced Datasets

Datasets	Performance Measures						
	Accuracy	**Precision**	**Recall**	**F-Measure**	**AUC**	**AUCPR**	**MSE**
ant	0.86	0.85	0.89	0.87	0.92	0.91	0.108
camel	0.84	0.83	0.85	0.84	0.90	0.88	0.123
ivy	0.95	0.95	0.95	0.95	0.98	0.99	0.042
jedit	0.99	0.98	1.00	0.99	0.99	0.99	0.013
log4j	0.96	1.00	0.93	0.96	0.95	0.98	0.060
xerces	0.94	0.94	0.93	0.94	0.96	0.97	0.061
Averages	**0.92**	**0.92**	**0.92**	**0.92**	**0.95**	**0.95**	**0.067**

Table 9. Comparison of the proposed models in terms of accuracy using paired t-test

Paired t-test	CNN Model		GRU Model	
	Original Datasets	**Balanced Datasets**	**Original Datasets**	**Balanced Datasets**
Mean	0.89	0.94	0.87	0.92
STD	0.07	0.05	0.07	0.05
Min	0.80	0.86	0.76	0.84
Max	0.97	0.99	0.96	0.99
P value	**0.004**		**0.004**	

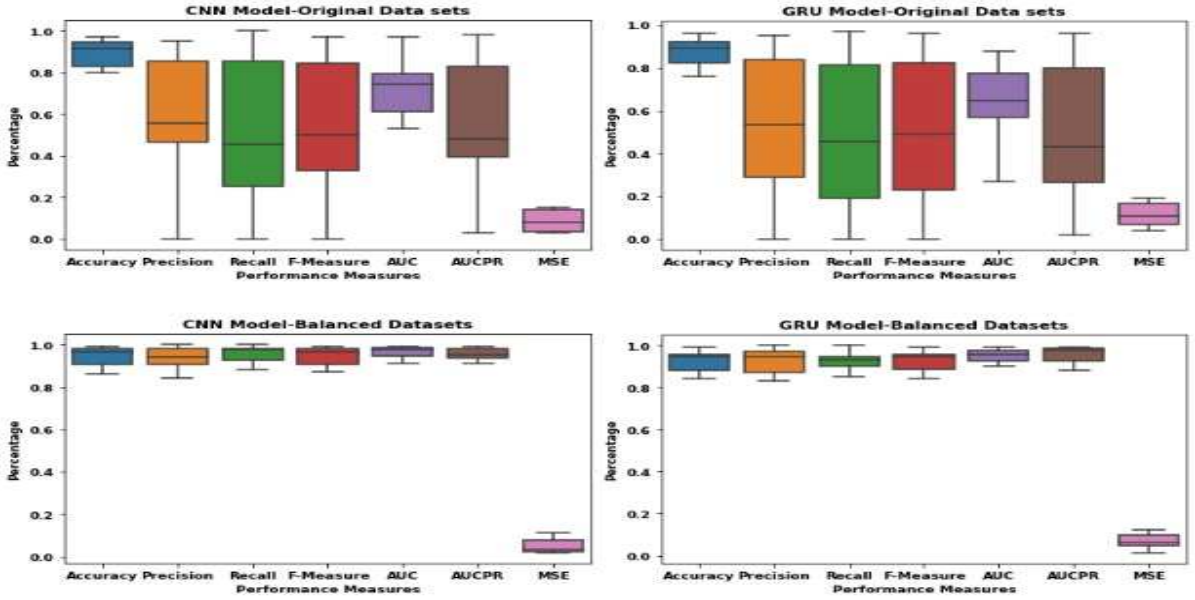

Figure 5. Boxplots represent performance measures obtained by proposed models on all datasets

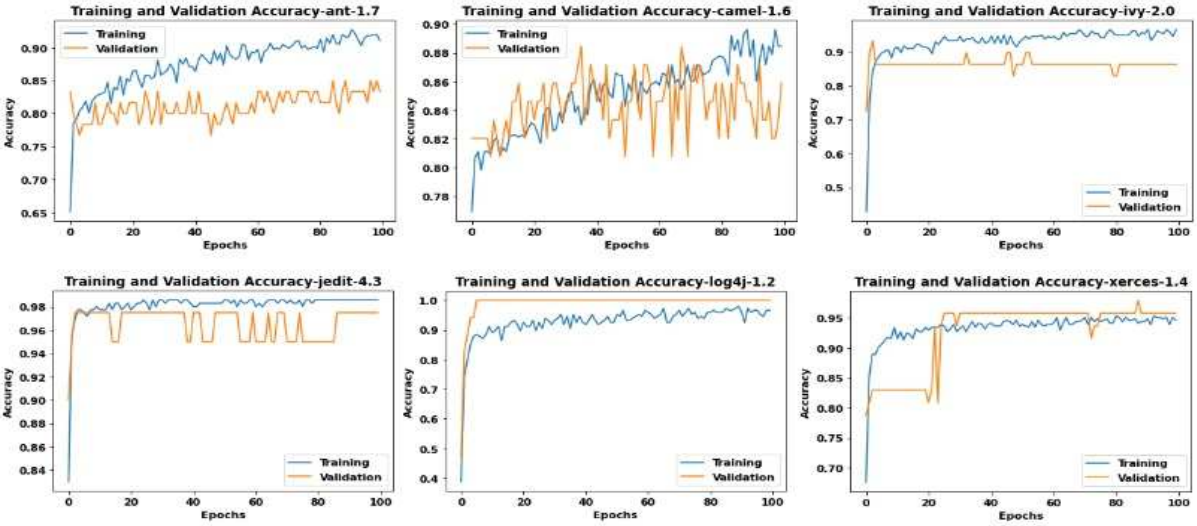

Figure 6. Training and Validation Accuracy for the original data sets - CNN model

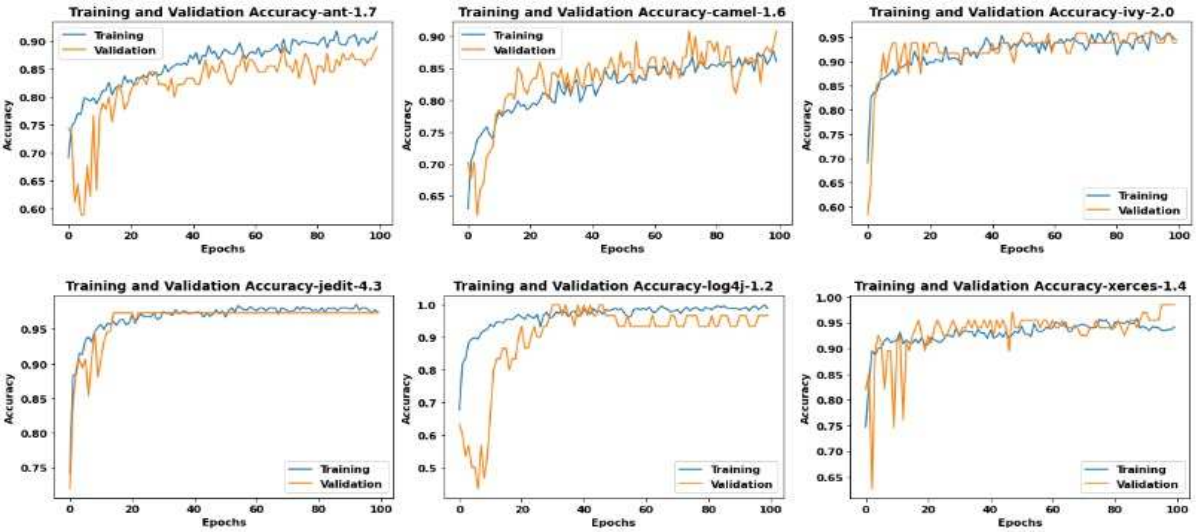

Figure 7. Training and Validation Accuracy for the balanced data sets - CNN model

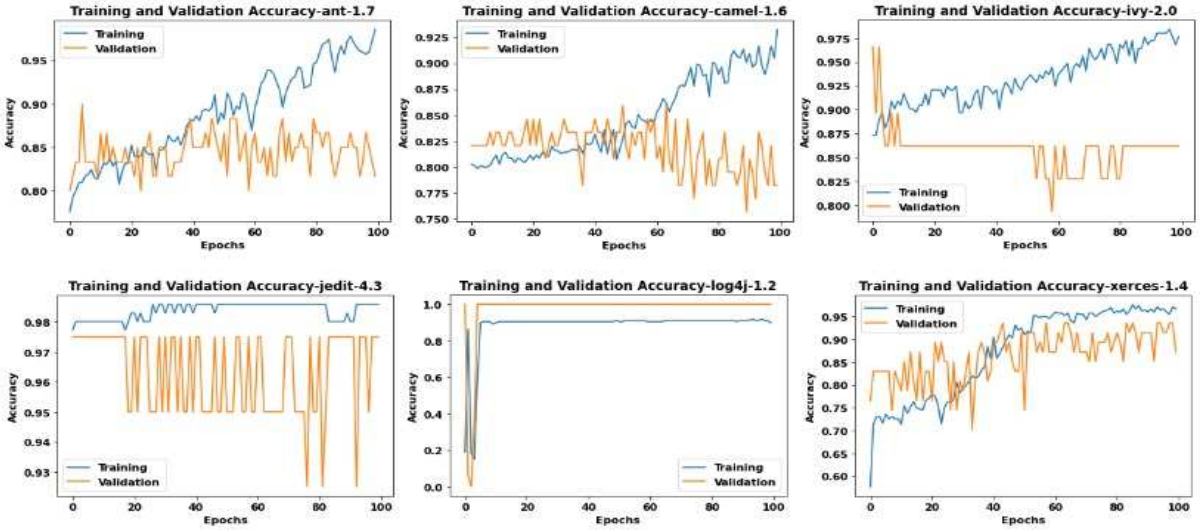

Figure 8. Training and Validation Accuracy for the original data sets - GRU model

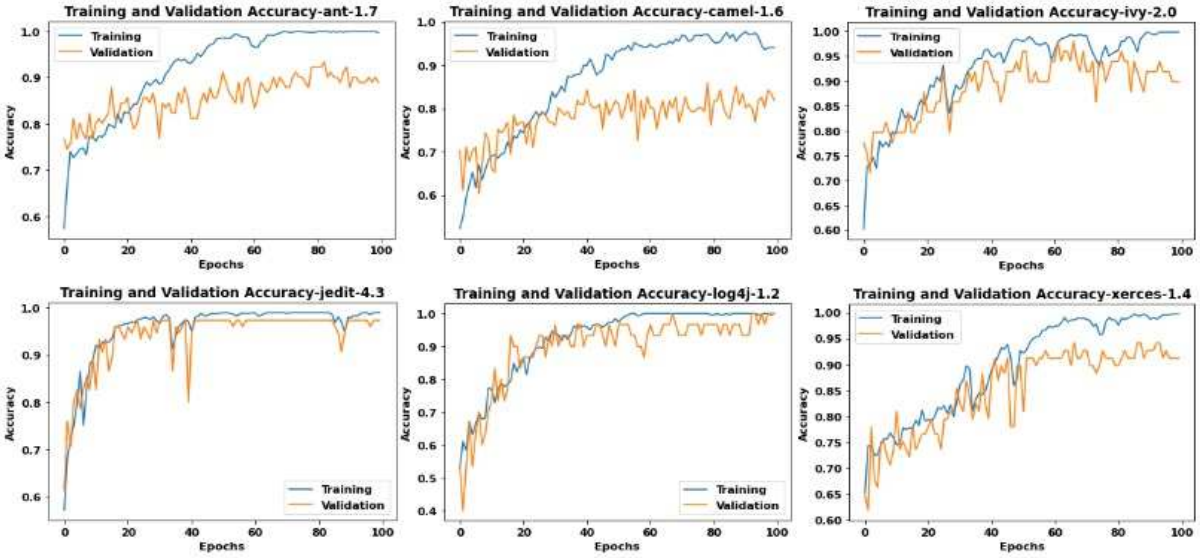

Figure 9. Training and Validation Accuracy for the balanced data sets - GRU model

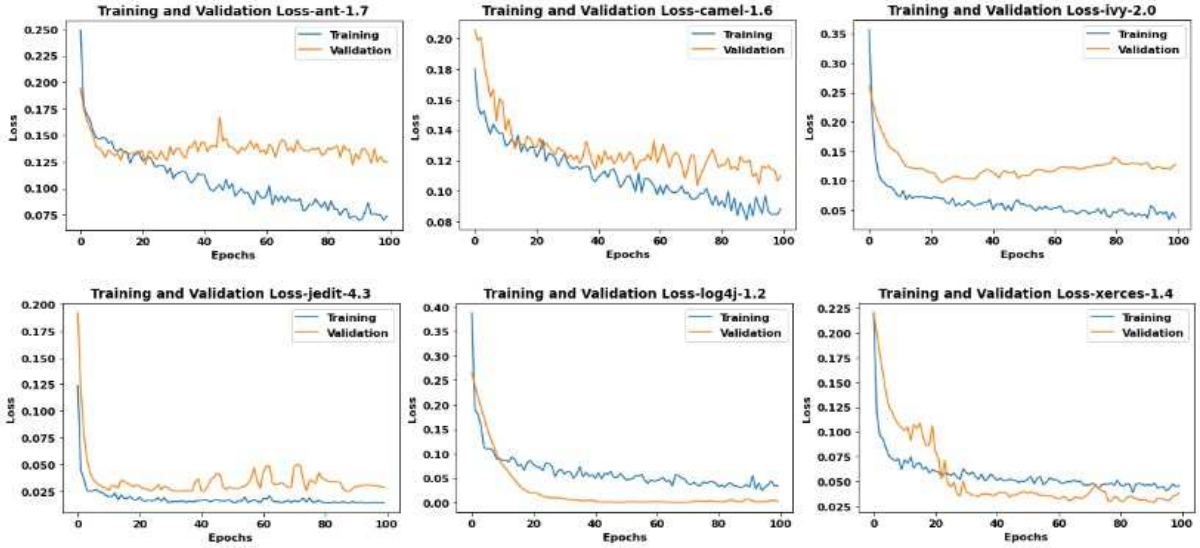

Figure 10. Training and Validation Loss for the original data sets - CNN model

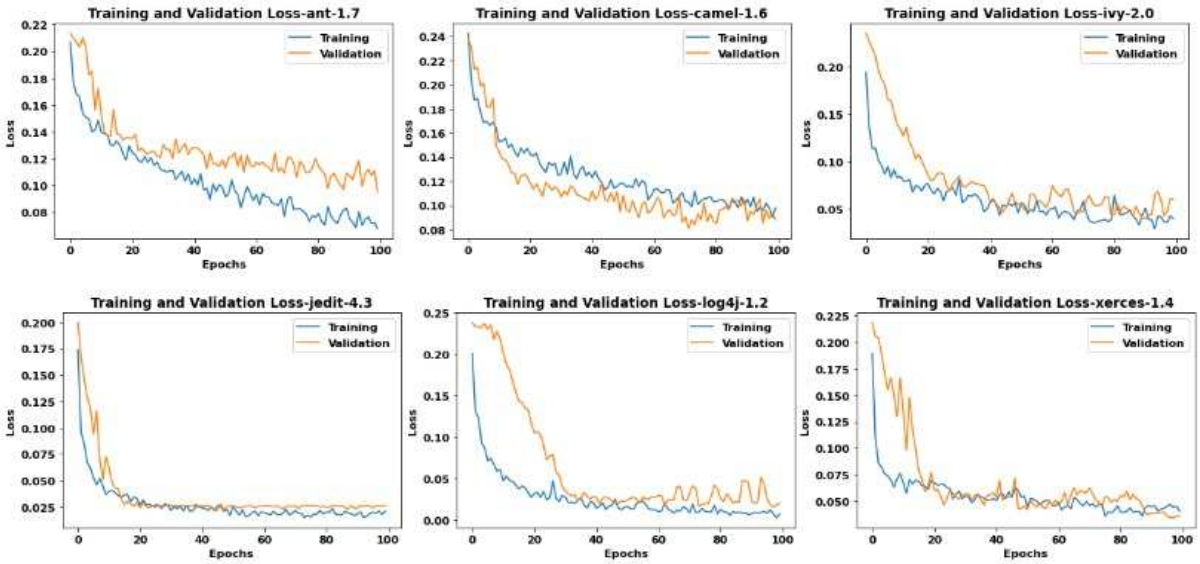

Figure 11. Training and Validation Loss for the balanced data sets - CNN model

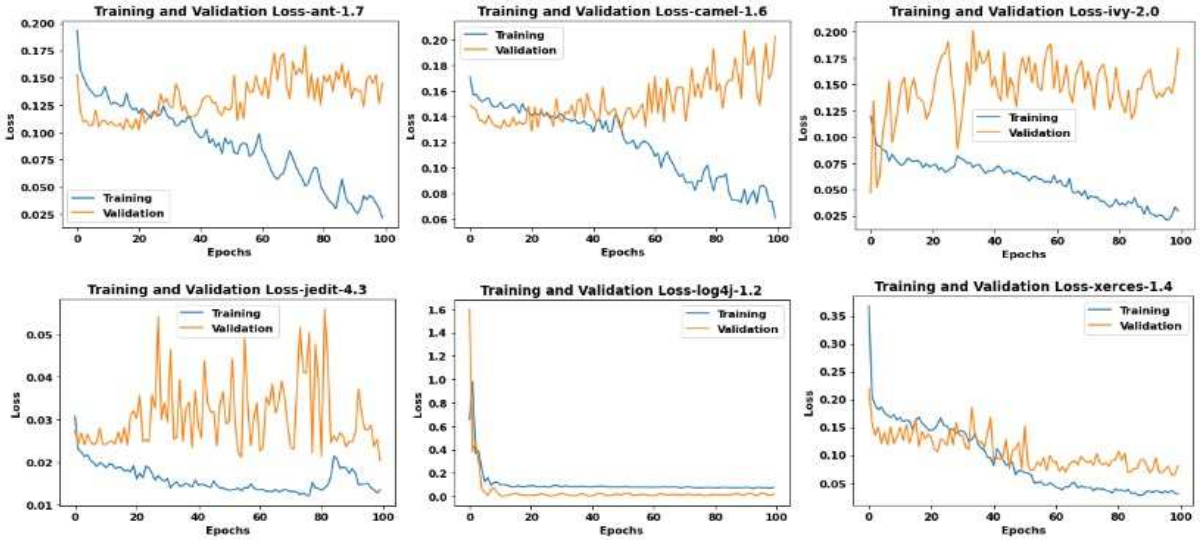

Figure 12. Training and Validation Loss for the original data sets - GRU model

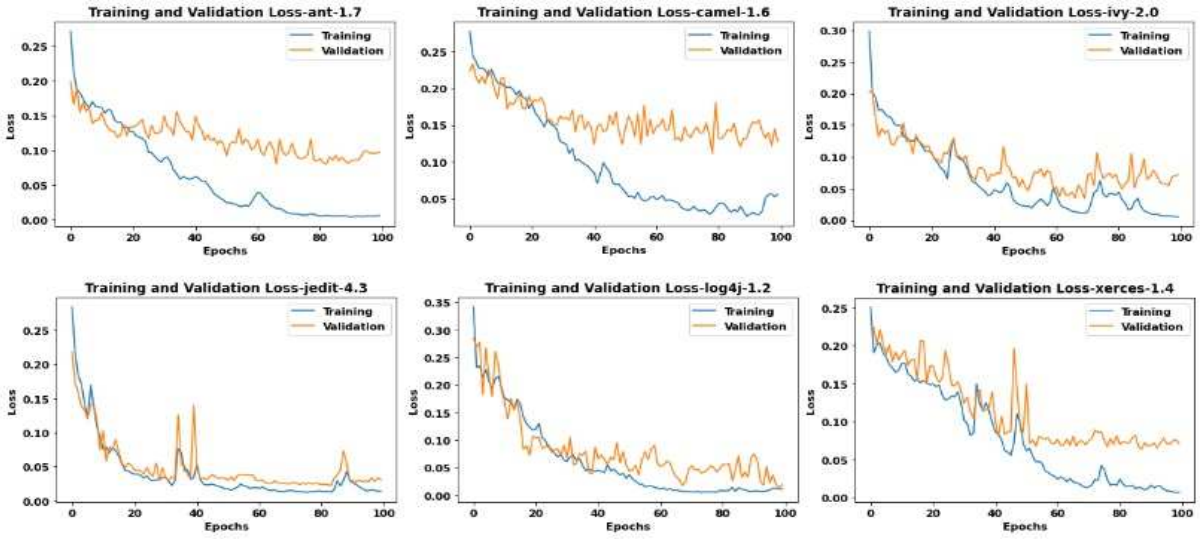

Figure 13. Training and Validation Loss for the balanced data sets - GRU model

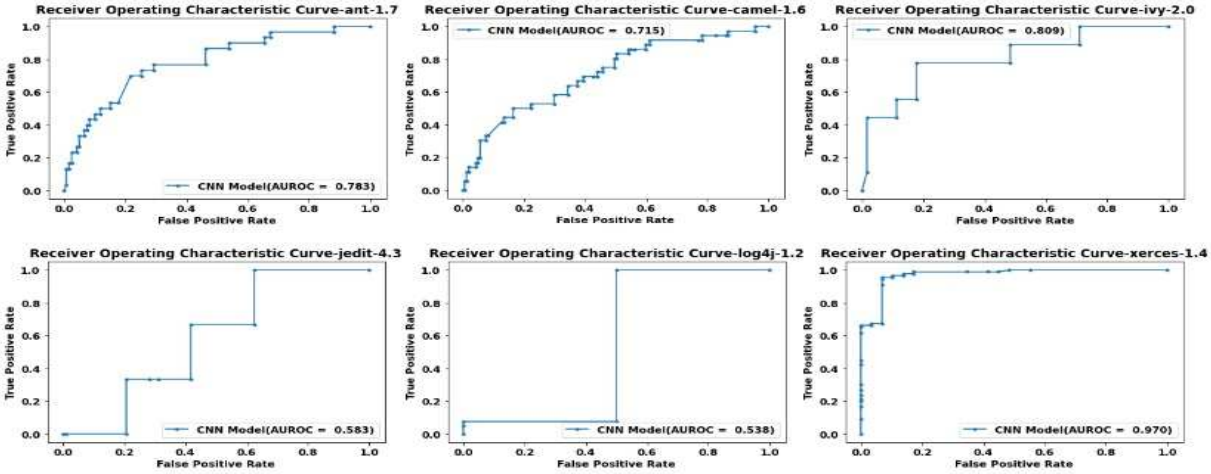

Figure 14. ROC curves for the original data sets - CNN model

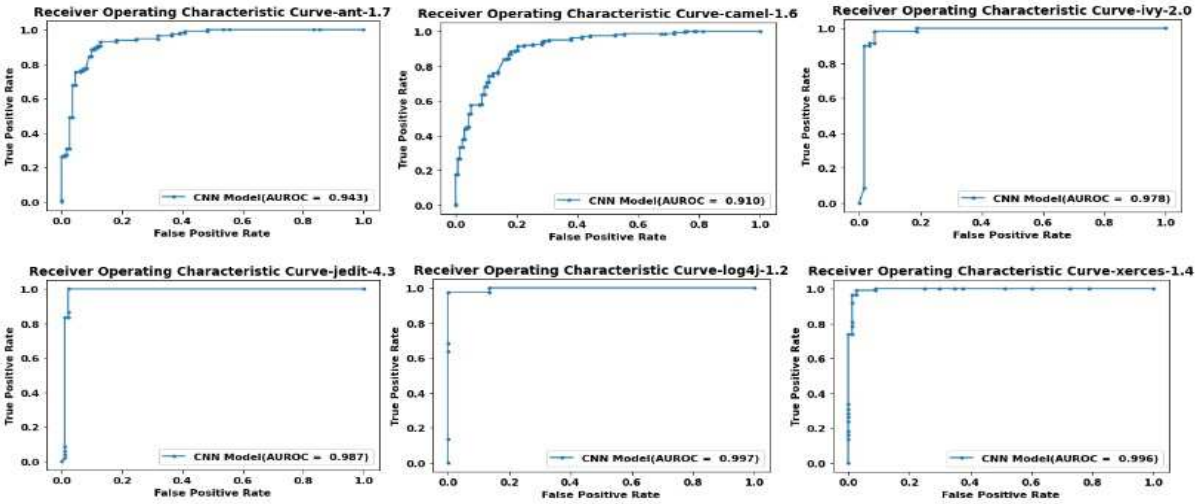

Figure 15. ROC curves for the balanced data sets - CNN model

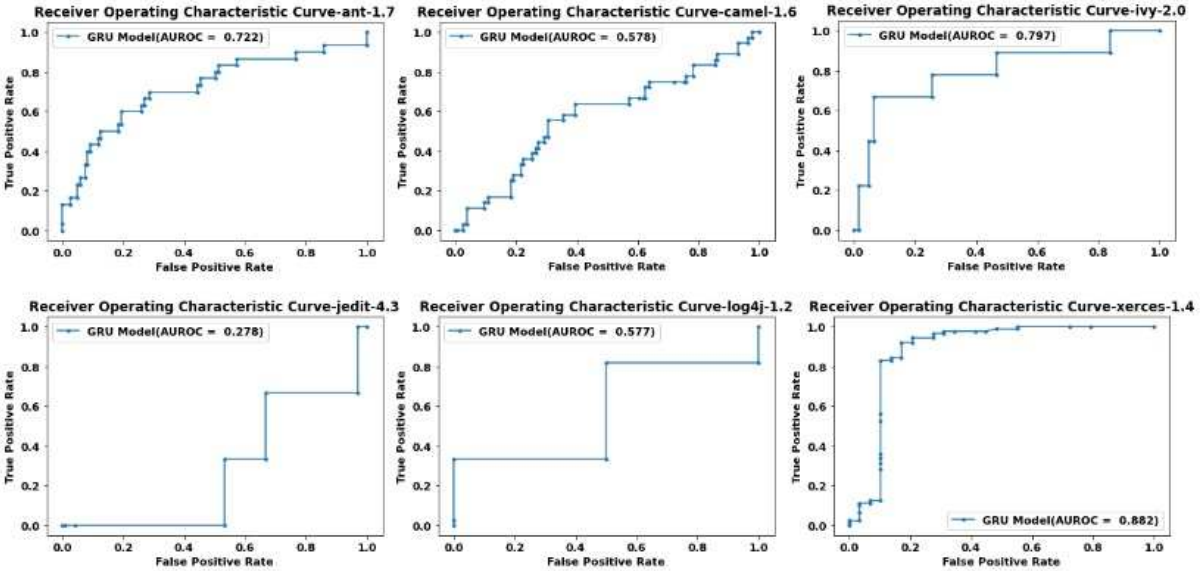

Figure 16. ROC curves for the original data sets - GRU model

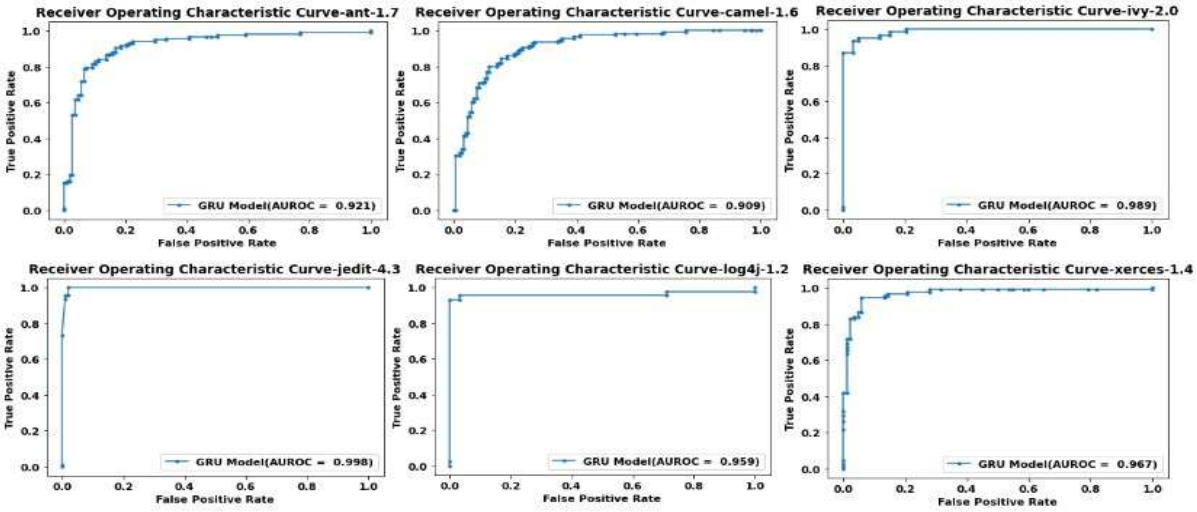

Figure 17. ROC curves for the balanced data sets - GRU model

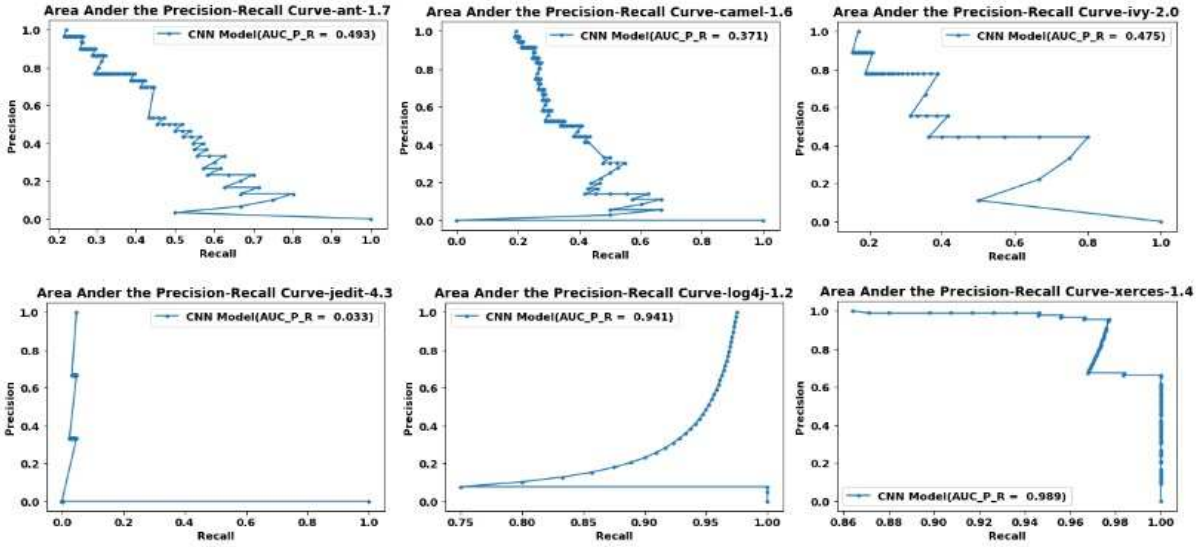

Figure 18. AUCPR for the original data sets - CNN model

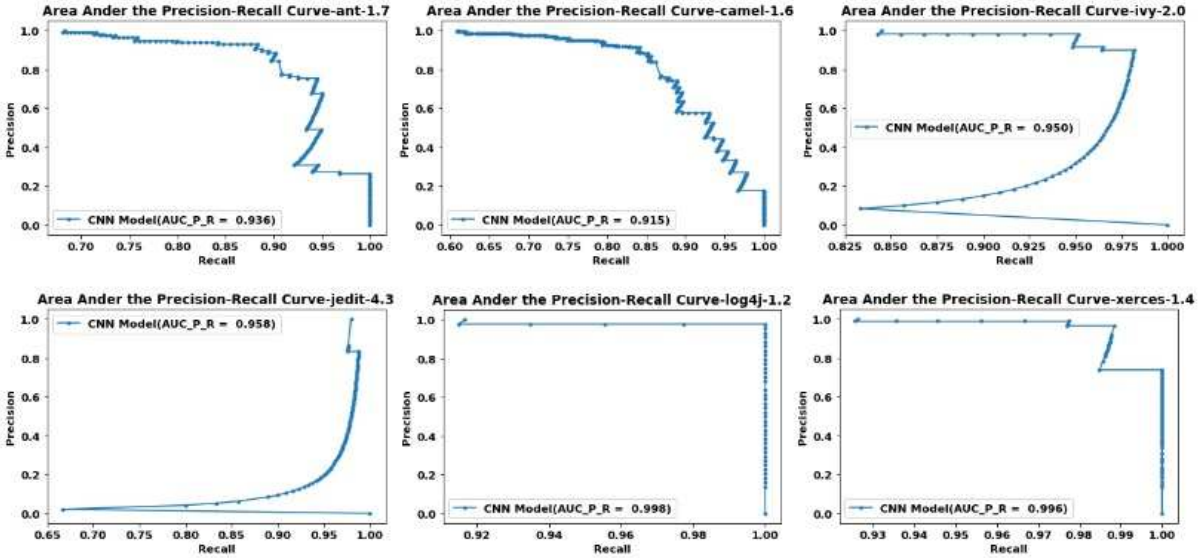

Figure 19. AUCPR for the balanced data sets - CNN model

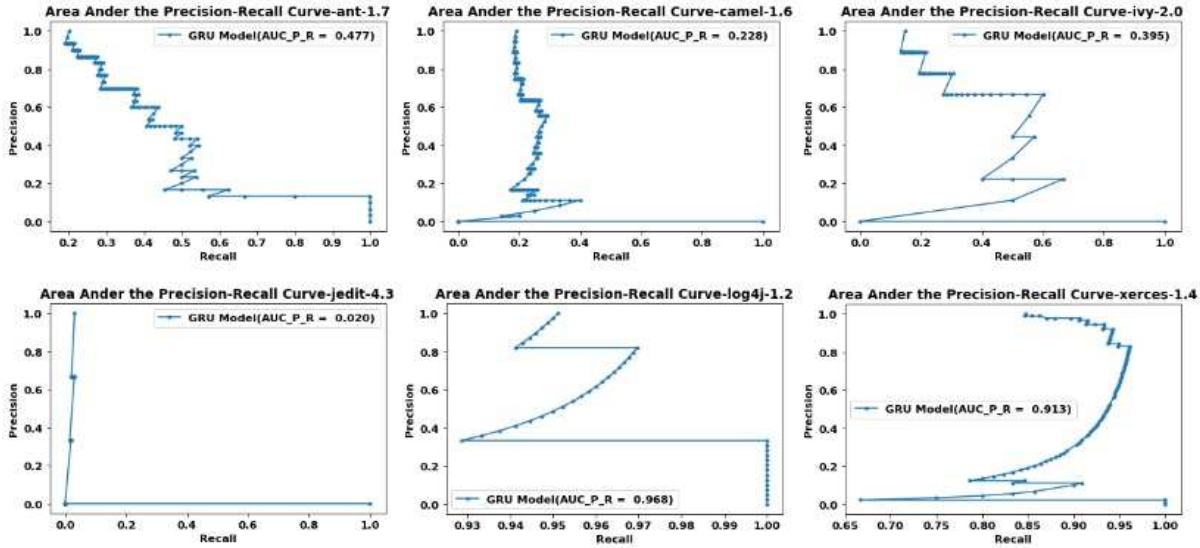

Figure 20. AUCPR for the original data sets - GRU model

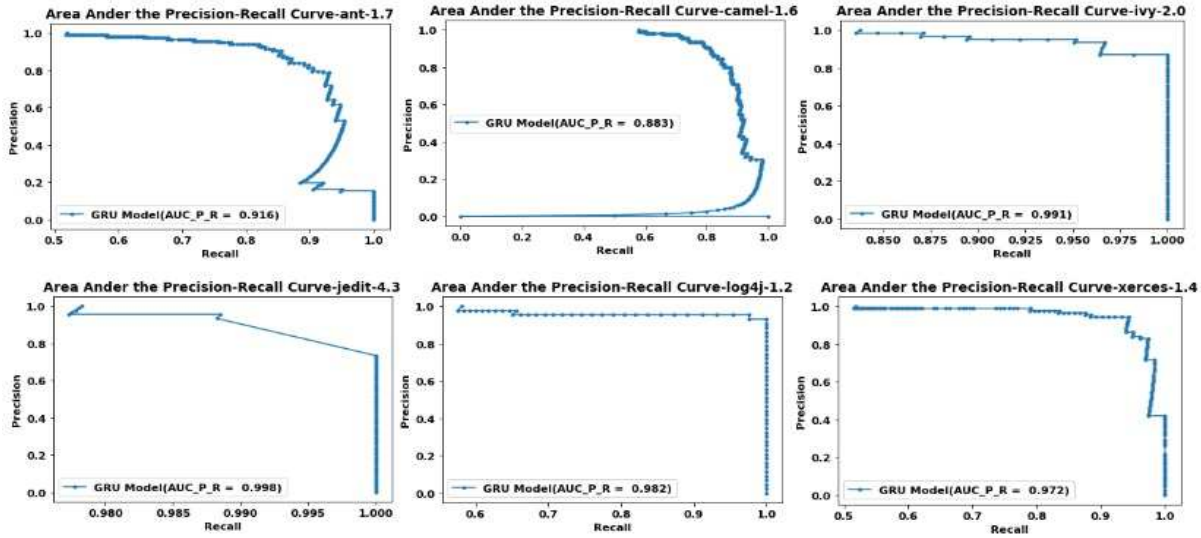

Figure 21. AUCPR for the balanced data sets - GRU model

7. Discussion

To answer the research questions - RQ2, we compared the results produced using our models with the results obtained using the baseline model (RF) based on five performance measures: accuracy precision, recall, f-Measure and AUC. Table 10 summarize the comparison between our models and the baseline model (RF). The best values are indicated in bold in the table. According to Table 10, our models outperform the baseline model in some datasets. We also compared the results produced using our models with the results obtained in previous studies based on five performance measures: accuracy precision, recall, f-Measure and AUC. Table 11 compares the values of performance measures obtained by our models and the performance values in previous studies. The best values are indicated with bold text and "- " to indicate the approaches that did not provide results in a particular data set. According to Table 11, some of the results in the previous studies are better than ours, but in the most cases, our models outperform the state-of-the-art approaches and provides better predictive performance.

Table 10 Performance measures of baseline model (RF) and proposed models

Models	Datasets	Performance Measures				
		Accuracy	Precision	Recall	F-Measure	AUC
	ant	0.81	0.54	0.47	0.50	0.68
	camel	0.81	0.50	0.22	0.31	0.58
	ivy	0.89	0.57	0.44	0.50	0.69

RF	jedit	0.97	0.00	0.00	0.00	0.50
	log4j	0.98	0.97	**1.00**	**0.99**	0.75
	xerces	0.95	0.95	0.99	0.97	0.90
Averages		**0.90**	**0.58**	**0.52**	**0.54**	**0.58**
CNN with SMOTE Tomek	ant	0.89	0.90	0.88	0.89	0.94
	camel	0.86	0.84	0.91	0.87	0.91
	ivy	0.95	0.92	0.98	0.95	0.97
	jedit	0.98	0.96	1.00	0.98	0.98
	log4j	0.99	1.00	0.98	0.99	0.99
	xerces	0.98	0.99	0.97	0.98	0.99
Averages		**0.94**	**0.93**	**0.95**	**0.94**	**0.96**
GRU with SMOTE Tomek	ant	0.86	0.85	0.89	0.87	0.92
	camel	0.84	0.83	0.85	0.84	0.90
	ivy	0.95	0.95	0.95	0.95	0.98
	jedit	0.99	0.98	1.00	0.99	0.99
	log4j	0.96	1.00	0.93	0.96	0.95
	xerces	0.94	0.94	0.93	0.94	0.96
Averages		**0.92**	**0.92**	**0.92**	**0.92**	**0.95**

Table 11 Comparison of the proposed models with other existing approaches

Approaches	**Datasets**	**Performance Measures**				
		Accuracy	**Precision**	**Recall**	**F-Measure**	**AUC**
LSTM [12]	Camel, Jedit, Log4j, Xerces	-	-	-	0.37, 0.44, 0.52, 0.26	-
LR [15]	Ant, Camel, IVY	-	-	-	0.52, 0.34, 0.30	-
K-NN [15]	Ant, Camel, IVY	-	-	-	0.53, 0.37, 0.30	-
MLP [15]	Ant, Camel, IVY	-	-	-	0.50, 0.38, 0.25	-
SVM [15]	Ant, Camel, IVY	-	-	-	0.50, 0.084, 0.28	-
HyGRAR [16]	JEdit, Ant	0.98, 0.96	0.70, 0.98	0.63, 0.85	-	0.81, 0.92
Hybrid Neural Network model [17]	JEdit, IVY, Ant, Camel	0.97, 0.88, 0.81, 0.81	**1.00**, 0.99, 0.93, **1.00**	**1.00**, 0.88, 0.84, 0.81	0.98, 0.93, 0.88, 0.89	-
CBIL [28]	Camel, JEdit, Xerces	-	-	-	0.93, 0.85, 0.95	0.96, 0.91, 0.98
DP-ARNN [29]	Camel, Xerces, JEdit	-	-	-	0.51, 0.27, 0.56	0.79, 0.76, 0.82
LSTM [30]	Camel		0.51	0.41	0.46	
CNN with SMOTE Tomek	ant, camel, ivy, jedit, log4j, xerces	0.89, 0.86, 0.95, 0.98, **0.99**, 0.98	0.90, 0.84, 0.92, 0.96, **1.00**, 0.99	0.88, 0.91, 0.98, **1.00**, 0.98, 0.97	0.89, 0.87, 0.95, 0.98, **0.99**, 0.98	0.94, 0.91, 0.97, 0.98, **0.99**, **0.99**
GRU with SMOTE Tomek	ant, camel, ivy, jedit, log4j, xerces	0.86, 0.84, 0.95, **0.99**, 0.96, 0.94	0.85, 0.83, 0.95, 0.98, **1.00**, 0.94	0.89, 0.85, 0.95, **1.00**, 0.93, 0.93	0.87, 0.84, 0.95, **0.99**, 0.96, 0.94	0.92, 0.90, 0.98, **0.99**, 0.95, 0.96

8. Implication of the findings

The results have implications for researchers and practitioners. They are interested in understanding quantitatively the effectiveness and efficiency of applying data balancing methods with ML techniques in SDP. Furthermore, the formers are concerned about the qualitative perspective of the results. So, we reported the implications related

to effectiveness, efficiency, comparison and relation with previous work in the previous sections (experimental results and discussion).

9. Threats to validity

This section discusses the threats to validity and experiments limitations of our study and how we mitigate them. It is important to assess the threats to validity such as construct, internal, external and experiments limitations, particularly constraints on the search process and deviations from the standard practice.

9.1. Construct validity

Construct validity concerns design of the study and its possibility to reflect the actual goal of the research. To avoid threats in study design we have applied a procedure of systematic literature review. To assure that researched area is relevant for study goal, we have cross-checked research questions and adjusted them several times to address the business needs. Besides, the metrics considered may be a threat to our study. We only adopt the static code metrics to predict defects. Thus, we cannot claim that we could generalize our conclusion to other types of metrics. However, the static code metrics were also widely adopted in many previous studies [44, 49]. Another threat is the construction of the ML models, for which we took several aspects into account that could have possibly influenced the study, i.e., data pre-processing, which features to consider, how to train the models, etc. However, the procedures followed in this respect are precise enough to ensure the validity of the study.

9.2. Internal validity

Threats to internal validity are related to the correctness of the experiments outcome or the process of performing the study. The main threat to internal validity is datasets. The reference datasets are imbalanced datasets that show a lack in the real distribution of the percentage of defects classes and non-defective classes. We manage this threat by modifying the original datasets in order to increase the realism of the data in terms of defect actual presence in the software system. The distribution of the dataset is modified by applying two data sampling techniques. Another threat is the most of our datasets have a small number of defects. These small number of defects create the problem that it may be difficult to generate statistically significant results, we tried to minimize that threat by applying standard performance measures for SDP. We however acknowledge that a number of statistical tests [50] can be applied to verify the statistical significance of our conclusions, Therefore, we plan to conduct more statistical tests in our future work.

9.3. External validity

External validity concerns relate to the generalization of our results. We tried to select and gathered different types of datasets from different projects of PROMISE repository to test our experiment. our criteria in project selection were based on the ratio of defects. So, we chose projects with a high percentage of defects and a low percentage of defects (projects with an unbalanced classes) to help us apply data balancing techniques. We built our model to adapt the combine of ML with balancing techniques in SDP. We selected six open-source Java projects of the PROMISE dataset as our evaluation datasets. However, we cannot declare that our results can be generalized. Future replications of this study are necessary to confirm the generalizability of our findings.

9.4. Experiments limitations

The limitations of the experiments are summarized as follows. First, the datasets used in our experiments is limited to only six open-source Java projects. Second, our findings may not be enough to generalize to all software in the industrial domain.

10. Conclusion

Recently, various ML and DL techniques have been used to build SDP models. Software defects have a major impact on the software development life cycle and defect prevention plays an important role in software quality assurance and effective help of software maintenance. SDP is a process of generating models or tools to predict software defects based on historical data. Early defect prediction helps prioritize and optimize effort and costs for inspection and testing. Historical software metrics that indicate defective data are primary inputs to the models. To improve the existing state-of-the-art approaches used to predict software defects, we proposed a novel approach based on CNN and GRU combined with SMOTE Tomek to predict defects in the source code. To evaluate the effectiveness of proposed models, we performed a series of experiments on six public software defect datasets. The results were compared with random forest (RF) as baseline model and compared with existing state-of-the-art SDP approaches. We found that the proposed models on the balanced datasets with average precision of 93% for CNN and 92% for GRU compared with RF (58%). Our results showed that the proposed models on the balanced datasets improves the average precision by 35% and 34%, respectively compared to RF. Our proposed

models outperform existing state-of-the-art SDP approaches significantly and substantially in most cases. The robustness and accuracy of SDP will be evaluated on various datasets in our future work.

Acknowledgments

The authors gratefully acknowledge the financial assistance from the Institute of Information Science, Faculty of Mechanical Engineering and Informatics, University of Miskolc.

Author Contributions

Both authors developed the whole work, discussed the results, and contributed to the final manuscript. The authors read and approved the final manuscript.

Funding

Open Access funding provided by University of Miskolc.

Declarations

Competing interests

The authors declared that they have no competing of interests to this work. We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

Ethics Approval and Consent to Participate

Not applicable.

Consent for Publication

Not applicable.

References

1. H.K.Dam, T.Pham, S.W.Ng, T.Tran, J.Grundy, A.Ghose, T.Kim and C.J.Kim, "A deep tree-based model for software defect prediction", arXiv preprint arXiv:1802.00921.https://doi.org/10.48550/arXiv.1802.00921
2. 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, 2018. https://doi.org/10.1016/j.infsof.2017.11.008
3. Y.Kumar and V.Singh, "A Practitioner Approach of Deep Learning Based Software Defect Predictor", Annals of the Romanian Society for Cell Biology, Vol. 25, No. 6, pp.14615-14635, 2021
4. G.Fan, X.Diao, H.Yu, K.Yang and L.Chen, "Software defect prediction via attention-based recurrent neural network", Scientific Programming, Vol. 2019. https://doi.org/10.1155/2019/6230953
5. Z.Li, X.Y. Jing and X.Zhu, "Progress on approaches to software defect prediction", IET Software, Vol. 12, No. 3, pp. 161-175, 2018. doi: 10.1049/iet-sen.2017.0148
6. S.Omri and C.Sinz, "Deep learning for software defect prediction: a survey", In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pp.209-214, Association for Computing Machinery, New York, NY, USA, 2020. https://doi.org/10.1145/3387940.3391463
7. 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
8. R.Ferenc, D.Bán, T.Grósz and T.Gyimóthy, "Deep learning in static metric-based bug prediction", Array, Vol. 6, pp. 100021, 2020. https://doi.org/10.1016/j.array.2020.100021
9. R.Yedida and T.Menzies, "On the value of oversampling for deep learning in software defect prediction", IEEE Transactions on Software Engineering, Vol. 48, No. 8, pp.3103 - 3116, 2021. DOI: 10.1109/TSE.2021.3079841

10. K.Zhu, N.Zhang, S.Ying and D.Zhu, "Within-project and cross-project just-in-time defect prediction based on denoising autoencoder and convolutional neural network", IET Software, Vol. 14, No. 3, pp. 185-195, 2020. https://doi.org/10.1049/iet-sen.2019.0278
11. H.Cao, "A systematic study for learning-based software defect prediction", In Journal of Physics: Conference Series, Vol. 1487, No. 1, pp.012017, 2020
12. J.Deng, L.Lu and S.Qiu, "Software defect prediction via LSTM", IET Software, Vol. 14, No. 4, pp. 443-450, 2020. doi: 10.1049/iet-sen.2019.0149
13. A.V.Phan and M.Le Nguyen, November. Convolutional neural networks on assembly code for predicting software defects. In 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES), Hanoi, Vietnam, pp.37-42, 2017. DOI: 10.1109/IESYS.2017.8233558
14. L.Chen, B.Fang, Z.Shang and Y.Tang, "Negative samples reduction in cross-company software defects prediction", Information and Software Technology, Vol. 62, pp. 67-77, 2015. https://doi.org/10.1016/j.infsof.2015.01.014
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, pp.1-16, 2020. https://doi.org/10.1007/s42979-020-0119-4
16. D.L.Miholca, G.Czibula and I.G.Czibula, "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks", Information Sciences, Vol. 441, pp. 152-170, 2018. https://doi.org/10.1016/j.ins.2018.02.027
17. R.S. Kumar and B.Sathyanarayana, "Adaptive Genetic Algorithm Based Artificial Neural Network for Software Defect Prediction", Global Journal of Computer Science and Technology, Vol. 15, No. 1, 2015
18. S.Qiu, H.Xu, J.Deng, S.Jiang and L.Lu, "Transfer convolutional neural network for cross-project defect prediction", Applied Sciences, Vol. 9, No. 13, pp.2660, 2019. https://doi.org/10.3390/app9132660
19. J.Deng, L.Lu and S.Qiu, "Software defect prediction via LSTM", IET Software, Vol. 14, No. 4, pp. 443-450, 2020. doi: 10.1049/iet-sen.2019.0149
20. C.Pan, M.Lu, B.Xu and H.Gao, "An improved CNN model for within-project software defect prediction", Applied Sciences, Vol. 9, No. 10, pp. 2138, 2019. https://doi.org/10.3390/app9102138
21. J.Deng, L.Lu, S.Qiu and Y.Ou, "A suitable ast node granularity and multi-kernel transfer convolutional neural network for cross-project defect prediction", IEEE Access, Vol. 8, pp.66647-66661, 2020. DOI: 10.1109/ACCESS.2020.2985780
22. 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, pp.e0247444, 2021. https://doi.org/10.1371/journal.pone.0247444
23. J.Li, P.He, J.Zhu and M.R.Lyu, "Software defect prediction via convolutional neural network", In international conference on software quality, reliability and security (QRS), pp.318-328, IEEE, Prague, Czech Republic, 2017. DOI: 10.1109/QRS.2017.42
24. 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, Vol. 19, No. 13, pp. 2964, 2019. https://doi.org/10.3390/s19132964
25. 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, pp. 113085, 2020.https://doi.org/10.1016/j.eswa.2019.113085
26. Z.Yang and H.Qian, "Automated Parameter Tuning of Artificial Neural Networks for Software Defect Prediction", In Proceedings of the 2nd International Conference on Advances in Image Processing, New York, NY, United States, pp.203-209, 2018. https://doi.org/10.1145/3239576.3239622
27. L.Zhao, Z.Shang, L.Zhao, T.Zhang and Y.Y.Tang, "Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks", Neurocomputing, Vol. 352, pp. 64-74, 2019. https://doi.org/10.1016/j.neucom.2019.03.076
28. A.B.Farid, E.M.Fathy, A.S.Eldin and L.A.Abd-Elmegid, "Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM)", PeerJ Computer Science, Vol. 7, pp. e739, 2021. https://doi.org/10.7717/peerj-cs.739
29. G.Fan, X.Diao, H.Yu, K.Yang and L.Chen, "Software defect prediction via attention-based recurrent neural network", Scientific Programming, Vol. 2019. https://doi.org/10.1155/2019/6230953
30. 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
31. L.Qiao, X.Li, Q.Umer and P.Guo, "Deep learning based software defect prediction", Neurocomputing, Vol. 385, pp. 100-110, 2020. https://doi.org/10.1016/j.neucom.2019.11.067
32. X.Li, J.Li, Y.Qu and D.He, "Gear pitting fault diagnosis using integrated CNN and GRU network with both vibration and acoustic emission signals", Applied Sciences, Vol. 9, No. 4, pp.768, 2019. https://doi.org/10.3390/app9040768

33. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
34. https://www.kaggle.com/datasets/nazgolnikravesh/software-defect-prediction-dataset
35. X.Xia, D.Lo, S.J.Pan, N.Nagappan and X.Wang, "Hydra: Massively compositional model for cross-project defect prediction", IEEE Transactions on software Engineering, Vol. 42, No. 10, pp. 977-998, 2016. DOI: 10.1109/TSE.2016.2543218
36. M.M.Öztürk, "Which type of metrics are useful to deal with class imbalance in software defect prediction?", Information and Software Technology, Vol. 92, pp. 17-29, 2017. https://doi.org/10.1016/j.infsof.2017.07.004
37. A.Alsaeedi and M.Z.Khan, "Software defect prediction using supervised machine learning and ensemble techniques: a comparative study", Journal of Software Engineering and Applications, Vol. 12, No. 5, pp. 85-100, 2019.DOI: 10.4236/jsea.2019.125007
38. S.Agarwal and D.Tomar, "A feature selection based model for software defect prediction", International Journal of Advanced Science and Technology, Vol.65, pp.39-58, 2014. http://dx.doi.org/10.14257/ijast.2014.65.04
39. T.Shippey, D.Bowes and T.Hall, "Automatically identifying code features for software defect prediction: Using AST N-grams", Information and Software Technology, Vol. 106, pp. 142-160, 2019. https://doi.org/10.1016/j.infsof.2018.10.001
40. L.Zhao, Z.Shang, L.Zhao, A.Qin and Y.Y.Tang, "Siamese dense neural network for software defect prediction with small data", IEEE Access, Vol. 7, pp.7663-7677, 2018. DOI: 10.1109/ACCESS.2018.2889061
41. S.Jain and A.Saha, "Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection", Science of Computer Programming, Vol. 212, pp. 102713, 2021. https://doi.org/10.1016/j.scico.2021.102713
42. K.Bashir, T.Li and C.W.Yohannese, . An empirical study for enhanced software defect prediction using a learning-based framework. International Journal of Computational Intelligence Systems, Vol. 12, No. 1, pp. 282-298, 2018. https://doi.org/10.2991/ijcis.2018.125905638
43. H.Tong, S.Wang and G.Li, "Credibility based imbalance boosting method for software defect proneness prediction", Applied Sciences, Vol. 10, No. 22, pp.8059, 2020. https://doi.org/10.3390/app10228059
44. S.Feng, J.Keung, X.Yu, Y.Xiao and M.Zhang, "Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction", Information and Software Technology, Vol. 139, pp. 106662, 2021. https://doi.org/10.1016/j.infsof.2021.106662
45. T.Elhassan and M.Aljurf, "Classification of imbalance data using tomek link (t-link) combined with random under-sampling (rus) as a data reduction method", Global Journal of Technology & Optimization, Vol. 1, 2016. DOI: 10.4172/2229-8711.S1:111
46. E.F.Swana, W.Doorsamy and P.Bokoro, "Tomek Link and SMOTE Approaches for Machine Fault Classification with an Imbalanced Dataset. Sensors", Vol. 22, No. 9, pp. 3246, 2022. https://doi.org/10.3390/s22093246
47. B.Jonathan, P.H.Putra and Y.Ruldeviyani, "Observation imbalanced data text to predict users selling products on female daily with smote, tomek, and smote-tomek", In 2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT), Bali, Indonesia, pp.81-85, 2020. DOI: 10.1109/IAICT50021.2020.9172033
48. K.Nehéz and N.A.A.Khleel, "A new approach to software defect prediction based on convolutional neural network and bidirectional long short-term memory", Production Systems and Information Engineering", Vol. 10, No. 3, pp. 1-15, 2022. DOI: https://doi.org/10.32968/psaie.2022.3.1.
49. L.Chen, B.Fang, Z.Shang and Y.Tang, "Negative samples reduction in cross-company software defects prediction", Information and Software Technology, Vol. 62, pp. 67-77, 2015. https://doi.org/10.1016/j.infsof.2015.01.014
50. A.Arcuri and L.Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering", Software Testing, Verification and Reliability, Vol. 24, No. 3, pp. 219-250, 2014.