A Classification Model for Software Bug Prediction Based on Ensemble Deep Learning Approach Boosted with SMOTE Technique

ThaerThaher[a], Faisal Khamayseh[a]

[a]*Department of Computer Science and Engineering, Palestine Polytechnic University, Hebron, Palestine*
thaer.thaher@gmail.com, faisal@ppu.edu

Abstract

In the software development process, the testing phase plays a vital role in assessing software quality. Limited resources pose a challenge in achieving this purpose efficiently. Therefore, early-stage procedures such as Software Fault Prediction (SFP) are utilized to facilitate the testing process in an optimal way. SFP aims to predict fault-prone components early based on some software metrics (features). Machine Learning (ML) techniques have proven superior performance in tackling this problem. However, there is no best classifier to handle all possible classification problems.Thus, building a reliable SFP model is still a challenge and open for research. The primary purpose of this paper is to introduce an efficient classification framework to improve the performance of the SFP. For this purpose, an ensemble of Multi-layer Perceptron (MLP) deep learning algorithm boosted with Synthetic Minority Oversampling Technique (SMOTE) is proposed. The proposed model is benchmarked and assessed using sixteen real-world software projects selected from the PROMISE software engineering repository. The comparative study revealed that ensemble MLP achieved promising prediction quality on the majority of datasets compared to other traditional classifiers as well as those in preceding works.

Keywords: Software Fault Prediction, deep learning, neural networks, Multi-layer perceptron, ensemble learning, SMOTE, imbalanced data

1. Introduction

Software Engineering (SE) is not limited to programming and systems development. It is in itself a set of systematic engineering procedures implemented to develop a reliable and bugs-free software [1]. The methodology with clearly defined phases for designing and producing high-quality software is called Software Development Life Cycle (SDLC) [2]. SDL process involves several successive stages that take software from the ideation phase to delivery. It encompasses the detailed steps for planning, designing, testing, and deploying a software product that ensures customer's expectations. Various SDLC models have been proposed to guide professionals during the software development process. The most common models are waterfall, iterative, spiral, V-shaped, and Agile models [2].

During the SDLC process, the testing stage plays a significant role in evaluating software quality [3]. In this stage, developers investigate whether the code and programming work according to customer requirements. Various types of testing including Quality Assurance (QA) testing, System Integration Testing (SIT) and User Acceptance Testing (UAT) are involved with the aim of minimizing the number of errors within the software [2]. QA testing concerns with standardized procedures that ensure delivering the desired product quality before it is released for the public. However, limited resources constitute a significant challenge to provide the rapid test results required by the

development team. For this purpose, early-stage procedures such as Software Fault Prediction (SFP) are utilized to facilitate the testing process in an optimal way [4].

SFP is the process of predicting fault-prone software modules based on some characteristics (metrics) of the software project [4]. It is an early step that is conducted before the actual start of the testing phase to obtain a high-quality product with minimum cost and effort [5]. Some faulty-modules may be passed during design and development without being detected and fixed, which will adversely affect the later produced versions of the software. Therefore employing SFP early in the software development process has attracted considerable attention to conduct the SDLC healthily [6]. The application of SFP techniques can primarily contribute to reducing the number of potential defects, and hence improve the overall quality of produced software. Besides, the early prediction of faults will reduce time, effort, and cost should be spent during the development process [7]. The development of SFP models depends on either design features (metrics) gathered during the design stage or historical fault datasets accumulated during the implementation of previous versions of similar projects [8]. These models are especially useful when dealing with limited resources projects or those that are complex and difficult to test [5].

The analysis of gathered software metrics, as well as historical data of projects, have found its wide application in enhancing the SDLC processes [5]. However, analyzing the complicated and vast amounts of data poses a significant challenge. For this purpose, Machine Learning (ML) techniques have been introduced to serve as potential computational methods for solving data mining related problems such as SFP. These techniques have proven superior performance in building efficient SFP models [9]. The researchers have exploited various ML algorithms to detect the software whether it is defected or normal based on datasets that include information about the previous versions of software. Some of these algorithms include Support Vector Machine (SVM), Decision Trees (DT), Artificial Neural Networks (ANN), and Navie Bayes (NB) [4].The effectiveness of the SFP model depends on several factors, including the exploited ML algorithm and the quality of the dataset [10]. One of the main challenges in this domain is the lack of high-quality datasets. That's to say, the available datasets may have irrelevant features and imbalanced distribution of ordinary and defected instances. These two common aspects significantly impact the performance of ML techniques [11] [12]. For this reason, pre-processing methods, such as Feature Selection (FS) and sampling methods such as Synthetic Minority Oversampling Technique (SMOTE), are highly required.

The main aim of this study is to introduce an efficient SFP model based on ensemble deep learning algorithm incorporated with SMOTE technique to enhance the overall prediction quality. The major contributions are summarized as follows:

- The SMOTE is utilized as a pre-processing technique for the sake of re-balancing datasets.
- A filter-based feature selection technique is employed to present the essential object-oriented metrics in predicting defected software.
- An ensemble classification model is constructed based on a Multi-Layer Perceptron (MLP) neural network to enhance the performance of the traditional MLP classifier.
- Sixteen real datasets are used to assess the performance of the proposed model.
- In comparison to the traditional classifiers as well as the state-of-the-art approaches, the proposed model showed a clear superiority.

The structure of this paper is organized as follows: The related works are reviewed in Section 2, including ML-based and FS based approaches. Section 3 introduces the theoretical concepts applied in this research, including Imbalanced data problem, feature subset selection methods, and

classification paradigms. The proposed methodology is presented in Section 5. Section 6 presents the experimental design, results, and analysis. Finally, Section 7 concludes the overall results as well as future work directions.

2. Review of Related works

The development SE has led to the proposing of several algorithms to predict software faults. ML techniques have attracted considerable interest from the research community to serve as potential computational methods for extracting knowledge from the software projects [4]. Moreover, various SFP datasets available as benchmarks are employed to assess the performance of proposed models. The most frequently used benchmarks in this domain are PROMISE repository, NASA datasets, and Qualitas corpus [4] [13]. This section presents a review of ML techniques for bugs detection in SE projects as well as about the deployment of sampling and feature selection methods in this area.

2.1. ML based techniques

The literature is rich with various traditional ML techniques to handle SFP problem including statistical classification techniques, supervised, and unsupervised techniques. Examples of these techniques including Support Vector Machine (SVM) [14], Decision Trees (DT) [15], Bayesian Networks (BN) [16], Naive Bayes (NB) [17], K-Nearest Neighbours (KNN) [18], Multi-layer Perceptron (MLP) [19], Artificial Neural Networks (ANN) [7] [20], Logistic Regression (LR) [21], and Multi-nomial Logistic Regression (MLR) [19]. For instance, Caglayan et. al employed BN, NB and LR methods to predic faults for an industrial software. Experimental results revealed that the proposed methods provide higher accuracy in predicting the overall defect proneness. Singh and Malhotra [22] conducted an empiric study to evaluate the performance of SVM classifier in determining the relationship between some software Object-Oriented (OO) design matrices and fault proneness. A dataset from NASA repository (KC1) and Receiver Operating Characteristic (ROC) were used to evaluate the proposed model. The study shows that SVM classifier is feasible and helpful in predicting faulty classes in OO based systems.

Some of the commonly used algorithms, such as DT, NB, and Neural Networks (NN) has provided improvements to prediction accuracy. However, it still has some limitations; on the one hand, expert knowledge is required for processing data. On the other hand, a massive amount of data is necessary for training operations, which becomes a significant challenge, especially in a dynamic environment [23]. To overcome these limitations and achieve better performance, researchers moved to take advantage of deep learning-based and ensemble techniques to tackle the SFP problem. For instance, Al Qasem and Akour [6] examined the performance of two deep learning algorithms to handle the SFP problem. They investigated the MLP and Convolutional Neural Network (CNN) on four NASA benchmarks from the PROMISE repository. The experimental results revealed the superiority of deep learning algorithms for the SFP problem.

Rathore and Kumar [24] introduced A model based on the principle of ensemble learning methods to predict the number of software faults. In this research, linear regression based combination rule (LRCR) and gradient boosting regression based combination rule (GBRCR) approaches were employed to ensemble the output of Genetic Programming (GP), MLP, and linear regression (LR) algorithms. Moreover, eleven datasets belong to six software projects were accumulated from the PROMISE data repository to assess the performance of the proposed ensemble models. Results of different performance evaluation measures including Average Absolute Error (AAE) and Average Relative Error (ARE) provided evidence that ensemble techniques are able to produce better results for the prediction of software faults compared to individual fault prediction techniques.

2.2. Feature Selection based approaches for SFP

In general, Feature Selection (FS) is an essential pre-processing step utilized to eliminate the irrelevant and/or redundant attributes that may adversely influence the performance of ML techniques [25]. Recently, various filter-based and wrapper-based FS methods are increasingly utilized to enhance the prediction quality of SFP models. Few studies have been conducted on the filter-based FS approach for the SFP problem. For instance, Balogun et al. [26] investigated the influence of applying eighteen filter FS methods comprising four feature ranking and fourteen feature subset selection methods on the performance of SFP models. Considering various machine learning classifiers and five datasets from NASA repository, the experimental results show that FS methods can improve the prediction rate of ML-based SFP models. It was observed that Information gain (IG) is the best feature ranking method. In contrast, Consistency Feature Subset Selection based on Best First Search (BFS) recorded the best effect on SFP models compared with other similar strategies. Catal and Diri [27] applied a correlation-based feature selection approach to capture highly relevant matrices for the SFP problem. The authors found that the Random Forest classifier (RF) outperforms other classifiers such as DT, NB, and Artificial Immune Recognition Systems (AIRS) when using this FS approach.

Besides, many wrapper FS methods based on evolutionary and swarm intelligence algorithms have been conducted for the SFP problem. For instance, Thaher and Arman [10] adopted a multi-swarm Harris Hawks Optimization algorithm (HHO) as a search strategy to capture the highly informative metrics for fifteen datasets from the PROMISE repository. They investigated three basic classifiers, namely, DT, KNN, and Linear Discriminant Analysis (LDA), incorporated with the Adaptive Synthetic Oversampling Technique (ADASYN). The empirical results revealed that the proposed approach with the LDA classifier improved the prediction quality for the most tested datasets. Turabieh et al. [5] introduced a wrapper feature selection method to improve the performance of a layered recurrent neural network (L-RNN) classifier for predicting faults in software modules. Three optimization algorithms comprising each of Binary Genetic Algorithm (BGA), Binary Particle Swarm Optimization (BPSO), and Binary Ant Colony Optimization (BACO) were applied in iterative manner to eliminate features (matrices) that are not correlated with the software defects. Considering a set of challenging software fault projects belongs to PROMISE data repository, the results confirmed that the proposed wrapper approach is better compared to other approaches.

2.3. Motivation of the study

Mostly, it is clear from the reviewed studies that most of them were dedicated to traditional classification paradigms. Few of them deployed neural networks and deep learning algorithms. Besides, the Non-Free-Lunch (NFL) theorem for optimization [28] can also be derived for ML and classification [29]. That is to say; there is no best classifier to handle all possible classification problems. Therefore, how to build a reliable SFP model is still a challenge and open for research. To the best of our knowledge, the ensemble of deep learning MLP classifier has not been utilized for tacking the SFP problem yet. The MLP ensembles will be a promising strategy to overcome the drawbacks of a single MLP network. These facts motivated the author of this work to propose an efficient classification framework for the early prediction of software fault-prone by employing ensemble MLP classification technique augmented with filter-based FS and SMOTE oversampling techniques.

3. Theoretical Background

This section introduces the main theoretical concepts used in this research, including feature subset selection, the problem of imbalanced data as well as SMOTE oversampling technique, and supervised classification paradigms such as ensemble methods and MLP.

3.1. Imbalanced data and SMOTE technique

The quality of data is considered as a significant factor that has a profound impact on the performance of ML techniques. imbalanced datasets are distinguished as a challenging aspect that may degrade the prediction quality of classification methods [12]. This issue emerges in most real-world problems in which the target classes are not represented equally. In other words, in binary class datasets, most of the instances are labeled with the first class (called majority class), while few of them are labeled with the other one (called minority class). In such a case, the classifier is trained on highly imbalanced data and thus has a tendency to pick up the patterns in the dominant classes, which leads to inaccurately prediction of the minority class [30].

Class imbalance problem poses as a major challenge in the filed of software defect prediction since the available datasets are highly imbalanced. that is to say, the occurrences of defected cases is very low compared to normal cases (see Figure 5). Therefore, dealing with this problem is becoming highly required. Various strategies can be employed to handle this problem, such as Cost-Sensitive, kernel-based, and sampling methods [30]. Sampling methods are distinguished into two types; oversampling, which increases the rate of the minority class, and under-sampling, which reduces the frequency of the minority class. The latter causes information to be lost, which leads to poor prediction quality [31]. In this research, we utilized an oversampling technique called SMOTE to rebalance the used SFP datasets.

The SMOTE is a promising oversampling method that proved its superiority in dealing with imbalanced data. It is originally introduced by Chawla et al. [32]. This technology is distinguished in that it preserves the original data without losing information, and it also increases the rate of the minority class without duplication. New synthetic samples labeled with the minority class are generated using the k-nearest neighbors method [12].

3.2. Feature Selection (FS)

One of the most common questions when applying ML algorithms is whether all features (factors) are relevant for classification rule. As a response to this question, a problem called FS emerged. FS is defined as the process of reducing the dimension of data by eliminating the irrelevant, noisy, and redundant features. In other words, it is the task of searching the most informative subset of features. It is an essential pre-processing technique that aims to enhance the performance of ML tasks [25].

FS approaches are classified into wrapper and filter based on the evaluation function used to measure the selected subset of features [33]. In wrapper-based methods, a search algorithm (deterministic or heuristic) is employed to generate subsets of features for examination. Then the effectiveness of each suggested subset of features is evaluated by a given classifier (learning algorithm). The evaluation is conducted in terms of several measures such as accuracy, area under the ROC, etc. FS is treated as a binary optimization problem in which the search algorithm is guided using the reported error by a classifier [34].

In the filter-based approach, the learning algorithm is not involved in the evaluation function. The effectiveness of a subset of features can be evaluated based on the intrinsic properties of the data. Statistical measures are used to score the dependency or correlation between features, which can be filtered to select the most informative features. Several ranking techniques have been introduced for feature evaluations, such as gain ration and information gain [35]. The filter-based approach is more effective compared to a wrapper-based approach in terms of the required computational time. In this study, we employed the filter-based FS approach using information gain measure to identify the most important metrics for building the SFP model.

3.3. Classification paradigms

In the SFP problem, we aim to predict fault-prone software modules early before the actual errors occur based on the designed metrics of the software project. Prediction of software modules, whether it is faulty or non-faulty, is a binary classification problem. several supervised classification paradigms can be used to tackle this kind of problems. In this study, we proposed a deep learning model using a multi-layer perceptron based ensemble technique. The following subsections we introduce the main characteristics of MLP and ensemble methods.

3.3.1. Multilayer Perceptrons (MLP) neural networks

MLP is a type of Feedforward Artificial Neural Network (FANN). It is basically a mathematical model inspired by the functioning of biological neural networks (NNs) that forms the human brain [36]. In this kind of NNs, the neurons are organized in parallel fully interconnected layers such that the connection between neurons is one-directional and one-way, as illustrated in Figure 1 . MLP consists of three levels of layers; the first layer represents the input layer, the last layer represents the output layer, and the other layers between them are called hidden layers. The MLP that consists of multiple hidden layers is called deep NNs, which have become popular due to the promising performance in dealing with complex ML tasks.

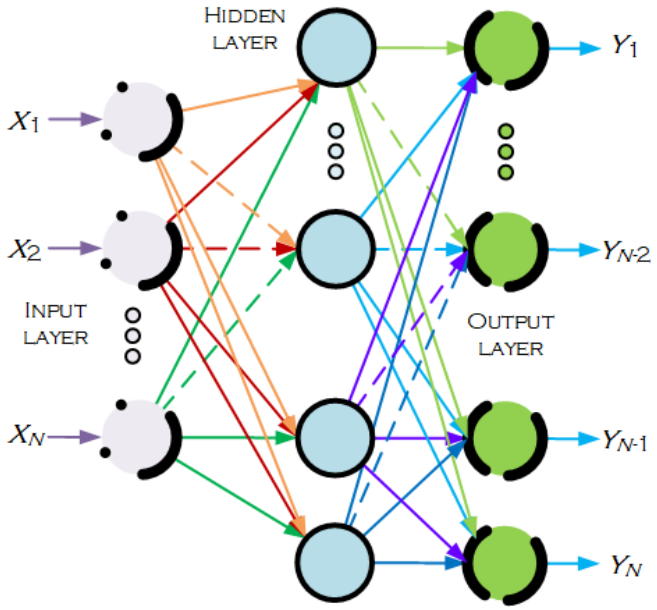


Figure 1: Simple MLP with one hidden layer

In MLP, the vector of features (input variables) is submitted to the input layer. For the other layers, each neuron has a summation function and an activation function. The summation function is employed to calculate the weighted sum of inputs (i.e., the sum of products of inputs and weights). The resultant summation is submitted to the activation function. The process of assigning proper values of weights to find the mathematical relation linking inputs to outputs is called the training process [37].

3.3.2. Ensemble learning methods

Ensemble learning combines several traditional ML models to enhance the predictive performance, generalizability, and robustness over a single classifier. It is not creating a new algorithm, but instead assembling together several ML models to create an ensemble learner, as demonstrated in Figure 2.

Ensemble learners often have lower error than every individual classifiers by themselves. Besides, they offer less overfitting as well as less biases caused by traditional learners [38].

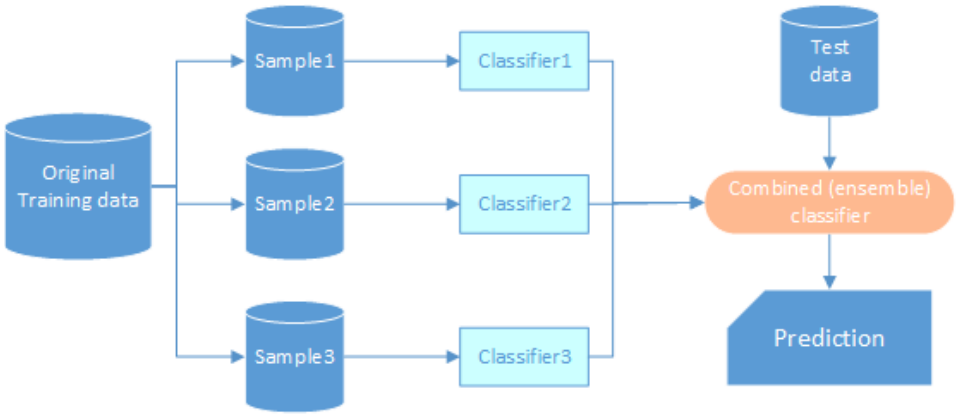


Figure 2: Ensemble Learning (bagging method)

Ensemble methods can be distinguished into three families, namely, bagging, boosting, and stacking methods. In the bagging method (also called bootstrap aggregation), for each model, a random sample of the original training dataset is provided by applying a bootstrap re-sampling technique. In other words, the considered classifiers are independently trained in parallel to create models for each one. The main idea of bagging is to aggregate multiple fitted models using several methods, such as averaging for regression problems and hard-voting for classification problems. The boosting process is similar to bagging being aimed to build a robust model by aggregating several models. However, the combined models in boosting are fitted sequentially. Models are iteratively fitted such that the training of the model at a specific step depends on the model trained in the previous steps. The main goal is to emphasize the observations that were misclassified by the previous models [38]. The stacking ensemble is usually used to combine heterogeneous classifiers based on training a meta-model (combiner). The main idea of stacking is to train the heterogeneous classifiers based on the available dataset, and then the combiner is fitted used the outputs (predictions) of these classifiers as additional inputs for training.

4. Investigated Software Projects

In this research, we used open research datasets in software engineering. They are gathered from PROMISE (PRedictOr Models In Software Engineering) repository, which is created to encourage the SE community to build verifiable, repeatable, refutable, and improvable predictive models [39]. The PROMISE is the most common repository for the SFP problem-related research [40]. It contains datasets of about 65 different software projects. Sixteen releases fall into six different projects (Ant, Camel, jEdit, Log4j, Lucene, and Xalan) have been selected for the experimental work in this study. They are of OO defect projects. Table 1 shows the characteristics of the selected datasets. As presented in the table, we selected 1 version of Ant and Lucene projects, four versions of Camel, five versions of jEdit, two versions of Log4j, whereas three versions of Xalan. These were selected because they are highly imbalanced. The main description of the investigated projects is summarized as follows [40] [41]:

- Apache Ant: An open-source java-based library and command-line tool used to build java applications. It provides several built-in tasks to compile, assemble, run, and test Java applications.

- Apache Camel: An efficient open-source integration framework. It based on Enterprise Integration Patterns (EIP) that can be utilized to integrate several systems providing or consuming data easily.
- jEdit: A free text editor software is written in java. It is open-source and has several features that make it outperform the most expensive development tools.
- Apache Log4j: A powerful java-based, and open-source logging package (APIs) written in java by the Apache Software Foundation.
- Apache Lucene: An open-source, java-based search software. It provides efficient indexing and search mechanisms, as well as hit highlighting, spellchecking, and advanced tokenization capabilities.
- Xalan: An open-source software library implements the XSLT transformation language to convert XML documents into HTML or other XML document types.

Table 1: Details of the 16 software projects (datasets) from PROMISE repository

Dataset	version	No. metrics	No. instances	No. normal instances	No. defective instances	Rate of defective instances
ant	1.7	20	745	579	166	0.223
camel	1.0	20	339	326	13	0.038
	1.2	20	608	392	216	0.355
	1.4	20	872	727	145	0.166
	1.6	20	965	777	188	0.195
jedit	3.2	20	272	182	90	0.331
	4.0	20	306	231	75	0.245
	4.1	20	312	233	79	0.253
	4.2	20	367	319	48	0.131
	4.3	20	492	481	11	0.022
log4j	1.0	20	135	101	34	0.252
	1.1	20	109	72	37	0.339
lucene	2.0	20	195	104	91	0.467
xalan	2.4	20	723	613	110	0.152
	2.5	20	803	416	387	0.482
	2.6	20	885	474	411	0.464

Software metrics (factors) are usually introduced to analyze and evaluate the quality of the software project. They generally classified into various categories, including traditional, Object-oriented (OO), and dynamic metrics [4]. OO metrics are calculated from software created utilizing the OO development strategy. Various software design metrics can be used in fault prediction models. The datasets utilized in this study are mainly OO defect projects. That is to say, a set of OO metrics used to judge whether the system is faulty or non-faulty. All investigated datasets consist of 20 metrics belong to different suites. The metrics are CK suite proposed by *Chidamber and Kemerer* [42] which includes Weighted Method Count (WMC), Number of Children (NOC), Coupling between Object class (CBO), Depth of Inheritance Tree (DIT), Lack of Cohesion in Methods (LCOM), and Response for a Class (RFC). *Martin* [43] suggested to metrics called Afferent couplings (CA) and Efferent couplings (CE). One metric introduced by *Henderson-seller* [44] called Lack of cohesion in methods (LCOM3). The suite suggested by *Bansiy and Davis* including [45] Number of Public Methods (NPM), Data Access Metric (DAM), Measure of Aggregation (MOA), Measure of Functional Abstraction (MFA), Cohesion Among Methods of Class (CAM). The quality-oriented suite suggested by *Tang et al.* [46]: Inheritance Coupling (IC), Coupling Between Methods (CBM), Average Method

Complexity (AMC). Maximum cyclomatic complexity (MAX_CC) and average cyclomatic complexity (AVG_CC) metrics introduced by *McCabe* [47]. The description of these metrics are summarized in Table 2.

Table 2: Description of object-oriented metrics

Metrics	Descriotion
WMC	Number of methods defined in a class.
DIT	provides a measure of the inheritance levels from the object hierarchy top for each class .
NOC	Number of immediate descendants of a class.
CBO	Count the number of classes coupled to a given class.
RFC	Count the number of distinct methods invoked by a class in response to a received message.
LCOM	Count the number of methods that do not share a field to the method pairs that do.
CA	Count the number of dependent classes for a given class.
CE	Count the number of classes on which a class depends.
NPM	Number of public methods defined in a class.
LCOM3	Count the number of connected components in a method graph.
LOC	Count the total number of lines of code of a class.
DAM	Computes the ratio of private attributes in a class.
MOA	Count the number of data members declared as class type.
MFA	Shows the fraction of the methods inherited by a class to the methods accessible by the functions defined in the class.
CAM	Computes the cohesion among methods of a class based on the parameters list.
IC	Count the number of coupled ancestor classes of a class.
CBM	Count the number of new or redefined methods that are coupled with the inherited methods.
AMC	Measures the average method size for each class.
MAX_CC	Maximum counts of the number of logically independent paths in a method.
AVG_CC	Average counts of the number of logically independent paths in a method.

5. The proposed methodology

This study aims to create an efficient classification model for the early prediction of faults in software modules. Four major components are employed for this purpose: SFP datasets, preprocessing techniques, learning algorithms (classification technique), and performance evaluation measures. Sixteen versions of different OO based projects are used to assess the proposed model. The list of the investigated datasets, as well as the OO metrics, are presented in Section 4. The proposed SFP methodology is illustrated in Figure 3.

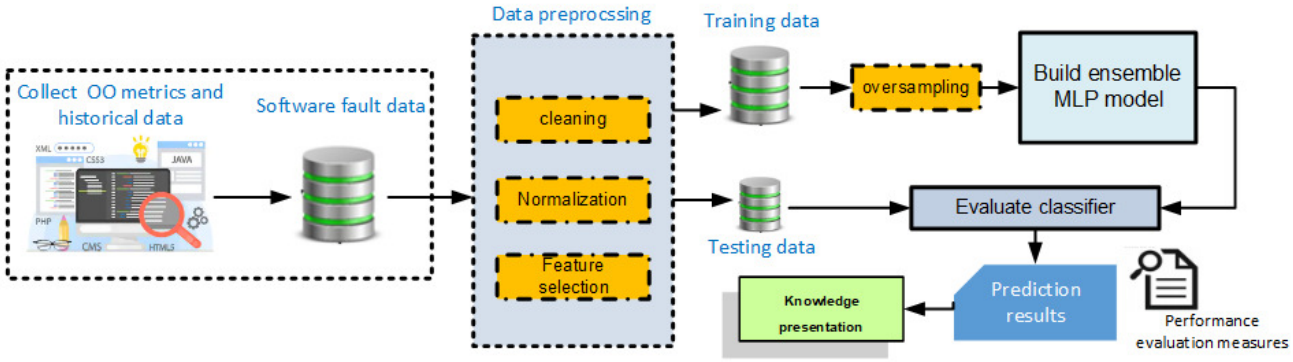


Figure 3: Software Fault Prediction process

5.1. preprocessing techniques

After exploring the datasets, we have noticed that they are highly imbalanced. The rate of defective instances is very low compared to normal ones. Thus, it is essential to resolve this problem

by balancing the datasets classes before training the model. To deal with data imbalance propblem, we applied SMOTE oversampling technique. Another essential aspect investigated in this stage is checking the most informative OO metrics for building the SFP model. In this study, we applied the Filter-based FS method using the information gain ranking technique.

5.1.1. proposed ensemble MLP for learning

Deep learning neural networks, including MLPs, provide promising performance in tacking various complex problems compared to other traditional techniques. They also provide enough flexibility for non-linearly separable problems. However, there are some limitations with MLPs such as premature convergence (local minima) and likely of over-fitting problems [48]. Therefore, utilizing ensemble learning will solve these problems. In this study, An ensemble classification model based on MLP as a base classifier is introduced to handle the SFP problem. We applied the bagging method to build the ensemble classifier such that each individual MLP classifier is trained on a random subset of the original training data generated using the bootstrap technique. Then, the predictions of the fitted models are aggregated using a voting method to form the final prediction. figure 4 shows the architecture of the ensemble MLP model.

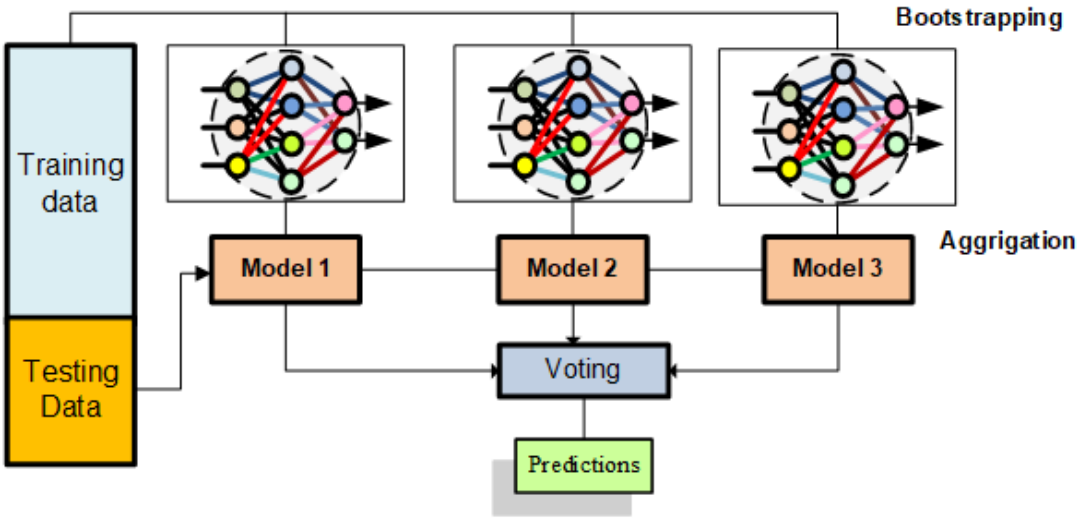


Figure 4: Ensemble MLP learner architecture

5.1.2. Evaluation measures

Various evaluation metrics can be used to measure the performance of the classification problems such as accuracy, Area Under the ROC (AUC), True Positive Rate (TPR), True Negative rate (TNR), precision, and F-measure. Accuracy is the most widely used metric to evaluate classic models [31]. However, it can be profoundly misleading of judging a model when dealing with imbalanced training data [12] [25]. For this purpose, we relied on AUC, TPR, and TNR to measure the performance of the proposed model. These measures are popular in the case of imbalanced data [5]. The confusion matrix of binary classification used to calculate the evaluation measures is demonstrated in Table 3. Formuals of calculating TPR, TNR, nad AUC are shown in Eqs. (1), (2), and (3), respectively.

Table 3: Confusion matrix for binary classification.

	Predicted positive	Predicted negative
Actual positive	True Positive (TP)	False Negative (FN)
Actual negative	False Positive (FP)	True Negative (TN)

- TPR: The proportion of positive cases that are correctly predicted as positive.

$$TPR = TP/(TP + FN) \tag{1}$$

- TNR: The proportion of negative cases that are correctly predicted as negative.

$$TNR = TN/(TN + FP) \tag{2}$$

- AUC: a measure of how well a model can distinguish between defected and normal groups.

$$AUC = (TPR + TNR)/2 \tag{3}$$

refer to paper 116 for methodology

6. Evaluation results and discussion

In this part, we intensely performed a set of experiments to investigate the efficiency of the proposed model. The experimental work was carried out in four stages. In the first stage, we explored, analyzed, and prepared the data to be compatible with ML techniques. Then we applied the resampling method and performed a comparison to assess the performance of MLP classifier on the original data and the balanced data in terms of AUC, TPR, and TNR. In the second phase, extensive experiments were conducted to modify the main parameters of the MLP classifier and choose those provided better prediction quality. Also, different classification techniques were implemented and compared to the MLP classifier. In the third phase, we investigated the impact of the ensemble learning approach on the performance of MLP as well as other implemented classification methods. Finally, the proposed ensemble MLP model was confirmed by comparing it with different approaches from the literature that used the same datasets employed in this research.

In all experiments, we used the validation set approach to estimate the prediction quality. All datasets are randomly split into 66% for training the model and 34% for testing the fitted model. Hence, the models are fitted and tested on completely separated samples. To reduce the impact of random components, we reported the average results for total runs of methods. Hence, we repeated the experiments 30 times for each algorithm. In reported results, the best values are highlighted with **boldface**. In order to record all results based on fair conditions, we employed a single computing system. Details of the utilized system are exposed in Table 4. We used the Python programming language to implement the classification framework. Various libraries were used to facilitate the implementation of ML algorithms such as Panda, Numpy, Matplotlib, and SKlearn (Scikit-learn).

6.1. Data Analysis and pre-processing

After inspecting the employed datasets, we found that they are free of noise and missing values. They are structured well to be mined. All features are numeric with different scales. Therefore, the Min-max normalization method was applied to standardize the range in the interval [0, 1] as given in Eq. (4) . Normalization is a highly recommended pre-processing step and useful to avoid bias towards some dominant features.

$$x^n = \frac{x - min}{max - min} \tag{4}$$

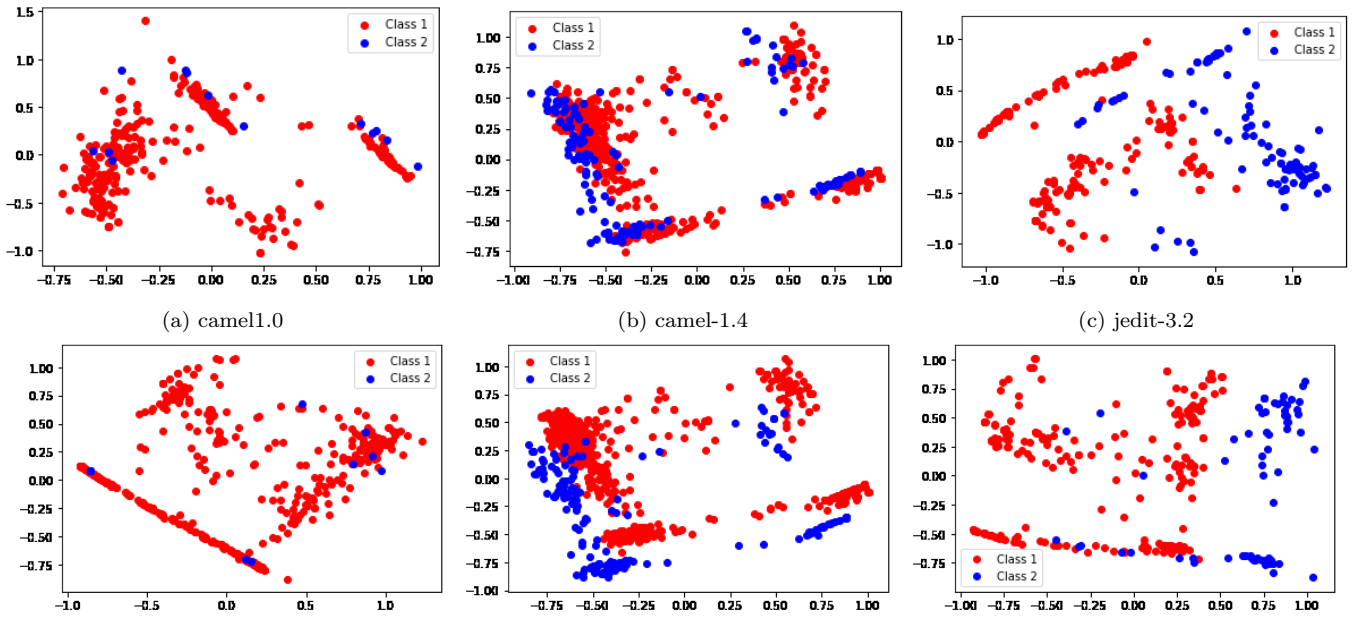
where x^n is the normalized value of x within the interval $[min, max]$.

Table 4: The system properties

Name	Setting
Hardware	
CPU	Intel Core(TM) i7-8550U
Frequency	2.2 GHz
RAM	8 GB
Hard drive	1 TB
Software	
Operating system	Windows 10 64bit
Language	Python 3.8.3

6.1.1. Data visualization

The 2D visualization for selected datasets using Principle Component Analysis (PCA) are shown in Figure 5. It is clear that the data are highly imbalanced. Moreover, the data are not linearly separable. Therefore, more complex learning algorithms are required to provide better performance.



(a) camel1.0 (b) camel-1.4 (c) jedit-3.2

(d) jedit-4.3 (e) camel-1.6 (f) jedit-4.0

Figure 5: Visualization of target class distribution based on the first 2 principal components of the dataset features.

6.1.2. Feature Selection (FS)

Figure 6 demonstrates the average ranks for the OO metrics after applying filter FS approach using information gain measure. In other words, it shows the most relevant metrics among all datasets. It can be observed that rfc metric achieved the best rank, followed by wmc, loc, lcom, nbm, lcom3, ce, cam, cbo, max-cc, respectively [top 10 ranks]. rfc metric represents the number of methods including the inherited methods per class, wmc is the sum of complexities for all methods per class which represents the development and maintenance cost of the class, loc indicates the number of code lines in a class. This confirms that these three metrics are relevant and should not be ignored for SFP data.

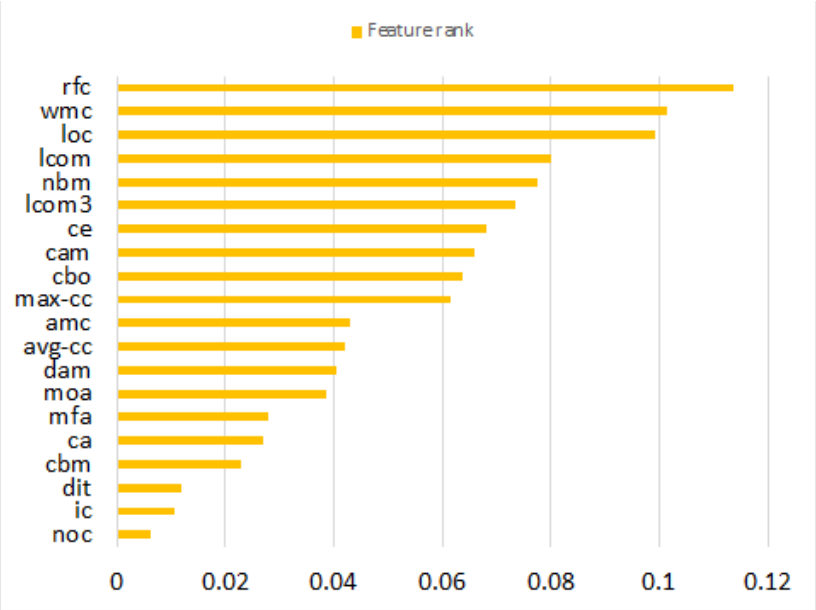


Figure 6: Average rank of features for all datasets using filter-based FS method

6.2. Evaluation of MLP classifier

In this part, the feed-forward MLP neural network that uses back-propagation learning is applied to solve the SFP problem. Besides, various classification models, namely K-Nearest Neighbors (KNN), Naive Bayes (NB), Linear Discriminant Analysis (LDA), Linear Regression, Decision Tree (DT), and Support Vector Machine (SVM) are also examined. A deep comparison in terms of AUC rates is presented.

6.2.1. Assessing the impact of SMOTE technique

In the current subsection, the efficiency of the SMOTE technique is appraised. Table 5 reveals the prediction performance of MLP classifier before and after applying SMOTE by inspecting the average AUC, TPR, and TNR on all datasets. By assaying the reported results, it can be observed that the integration between MLP classifier and SMOTE method achieves better performance in almost 94% of the datasets (15 out of 16). Additionally, it can be seen significant improvements in the quality of predicting the faulty instances (i.e., TNR). These results prove the importance of balanced data to enhance the overall performance of the model. Therefore, the subsequent experiments are performed on the balanced datasets.

6.2.2. Hyper-parameters tuning

One of the major challenges when dealing with MLP is a large number of free parameters such as the number of epochs, hidden layers, activation function, the optimizer, and learning rates. The performance of the MPL is influenced strongly by the chosen parameters. Therefore, this part exhibits a comprehensive experimental design in which fourteen experiments with different combinations of the main parameters (optimizer, epochs, hidden layers, and activation function) were conducted. 6 presents the considered parameters as well as their values. Table A.10 in the Appendix Appendix A reports the average AUC rates achieved by MLP using different configurations. by inspecting the table, it can be easily concluded that the MLP model can provide better performance on most datasets when the parameters epochs, hidden layers, activation function, and optimizer is set to (1000, 3, ReLU, adam) respectively. The learning rate in this study is set to be adaptive. Therefore, these settings are fixed for the subsequent experimental work.

Table 5: Assessing the impact of SMOTE method in terms of TPR, TNR, and AUC measures [Averaged over 30 independent runs]

Dataset	original			SMOTE		
	TPR	TNR	AUC	TPR	TNR	AUC
ant-1.7	0.9387	0.4053	0.6720	0.7555	0.7674	**0.7614**
camel-1.0	0.9899	0.0540	0.5220	0.8728	0.9908	**0.9318**
camel-1.2	0.9666	0.0906	0.5286	0.6597	0.6032	**0.6314**
camel-1.4	0.9655	0.1508	0.5581	0.7521	0.7296	**0.7409**
camel-1.6	0.9696	0.1473	0.5584	0.6956	0.7709	**0.7333**
jedit-3.2	0.8499	0.6263	0.7381	0.7916	0.7919	**0.7917**
jedit-4.0	0.9665	0.2200	0.5933	0.7592	0.8007	**0.7799**
jedit-4.1	0.9229	0.4813	0.7021	0.8142	0.7851	**0.7996**
jedit-4.2	0.9623	0.3388	0.6505	0.7844	0.8341	**0.8092**
jedit-4.3	0.9930	0.0111	0.5020	0.9119	0.9996	**0.9557**
log4j-1.0	0.8481	0.5869	0.7175	0.7558	0.8311	**0.7934**
log4j-1.1	0.8424	0.6205	0.7314	0.7676	0.8096	**0.7886**
lucene-2.0	0.7148	0.6298	0.6723	0.6688	0.6990	**0.6839**
xalan-2.4	0.9636	0.2210	0.5923	0.7360	0.8257	**0.7809**
xalan-2.5	0.6933	0.5853	0.6393	0.6720	0.6105	**0.6413**
xalan-2.6	0.8207	0.6447	**0.7327**	0.7846	0.6738	0.7292

Table 6: The fourteen experimental scenarios conducted to tune the main parameters of MLP classifier

scenario	optimizer	Epochs	hidden Layers	activation function	best
Sen_1	**adam**	1000	2	tanh	adam
Sen_2	SGD				
Sen_3	adam	**1000**	2		1000
Sen_4		2000			
Sen_5		3000			
Sen_6		5000			
Sen_7	adam	1000	2	tanh	3
Sen_8	adam		**3**		
Sen_9			4		
Sen_{10}			5		
Sen_{11}	adam	1000	3	tanh	ReLU
Sen_{12}				**ReLU**	
Sen_{13}				logistic	
Sen_{14}				identity	

6.2.3. Comparison with traditional classification techniques

For an intense investigation about the performance of the MLP classifier, it is compared to six traditional classifiers KNN, NB, LDA, LR, SVM, and DT in terms of test AUC rates. The obtained results are reported in Table 7. Further, the over-fitting problem of models is considered. For providing a fair comparison between these classifiers and finding the overall rank, the average ranking values of the Friedman test (F-Test) are also discussed. The results in Table 7 reveals that MLP outperforms the other competitors in achieving higher AUC on 63% of the utilized dataset. Regarding DT performance, there is clear evidence of the over-fitting problem. According to the overall rank and F-test, MLP is in the first rank, followed by KNN, LDA, SVM, LR, and NB, respectively. These results confirm the superiority of MLP classifier to handle the SFP problem.

Table 7: Comparison of MLP neural network against other traditional classifiers in terms of test AUC rates

Dataset	MLP	KNN	NB	LDA	LR	SVM	DT	
	test	test	test	test	test	test	train	test
ant-1.7	0.7896	**0.8137**	0.7026	0.7589	0.7281	0.7291	1.0000	0.7964
camel-1.0	**0.9638**	0.9008	0.8135	0.8344	0.8005	0.8254	1.0000	0.9225
camel-1.2	0.6649	**0.6724**	0.5554	0.6142	0.5898	0.5812	0.9898	0.6755
camel-1.4	**0.7834**	0.7663	0.5920	0.6996	0.6612	0.6818	1.0000	0.8108
camel-1.6	0.7577	**0.7743**	0.5833	0.6706	0.6495	0.6392	0.9987	0.7936
jedit-3.2	**0.8039**	0.7942	0.7359	0.7890	0.7822	0.7907	1.0000	0.7751
jedit-4.0	**0.8226**	0.8221	0.6699	0.7406	0.7136	0.7119	0.9979	0.7926
jedit-4.1	**0.8444**	0.8144	0.7043	0.7976	0.7579	0.7601	1.0000	0.7861
jedit-4.2	**0.8648**	0.8531	0.7410	0.8203	0.7772	0.7925	1.0000	0.8615
jedit-4.3	**0.9708**	0.9422	0.7178	0.8695	0.7694	0.7844	0.9915	0.9546
log4j-1.0	**0.8316**	0.7879	0.7522	0.7685	0.7681	0.7794	1.0000	0.7864
log4j-1.1	0.7980	**0.8150**	0.7597	0.7602	0.7769	0.7818	1.0000	0.7603
lucene-2.0	0.6682	0.6513	0.6468	0.6848	0.6928	**0.7109**	0.9976	0.6295
xalan-2.4	**0.8275**	0.8258	0.6462	0.7386	0.7331	0.7323	1.0000	0.8296
xalan-2.5	**0.6486**	0.6292	0.5576	0.6002	0.5927	0.5901	0.9983	0.6415
xalan-2.6	0.7332	**0.7362**	0.6909	0.7195	0.7236	0.7096	0.9955	0.7200
Rank (F-Test)	**1.2**	1.88	5.94	3.31	4.38	4	Overfitting	

6.3. Evaluation results using ensemble methods

Extensive experiments were conducted to investigate whether ensemble learning exceeds individuals classifiers in dealing with the SFP problem. A paired comparison between each individual classifier and the corresponding ensemble bagging method is presented in Table 8. The resultant table clarifies that, in most cases, ensemble methods exceed the traditional classifiers in achieving higher AUC values. Regarding the MLP classifier, it can be seen that ensemble MLP outperforms the basic MLP in 15 out of 16 datasets (94%). Regarding the overall rank of all methods in the table, the top 5 methods are ensemble MLP, RF, ensemble DT, MLP, ensemble KNN. Overall, the proposed ensemble MLP resents the best performance compared with the other approaches. This confirms that ensemble MLP makes a powerful model that provides superior performance compared to other classifiers when dealing with the utilized datasets. For an intense investigation about the performance of the MLP classifier, it is compared to six traditional classifiers KNN, NB, LDA, LR, SVM, and DT in terms of test AUC rates and the F-test ranking for all coded algorithms.

6.4. Comparing with state-of-the art methods

In the current section, we validate the AUC results of the proposed ensemble MLP by comparing it with those reported in preceding works. For this purpose, we compared the proposed method with

Table 8: Evaluation results using bagging (ensemble) methods

Dataset	MLP		KNN		NB		LDA		LR		SVM		DT		RF
	basic	ensemble	basic	ensemble	basic	ensemble	basic	ensemble	basic	ensemble	basic	ensemble	basic	ensemble	
ant-1.7	0.790	0.821	0.814	0.812	0.703	0.703	0.759	0.756	0.728	0.726	0.729	0.734	0.796	0.849	**0.853**
camel-1.0	**0.964**	0.961	0.901	0.901	0.814	0.819	0.834	0.851	0.801	0.802	0.825	0.824	0.922	0.944	0.954
camel-1.2	0.665	**0.699**	0.672	0.665	0.555	0.558	0.614	0.610	0.590	0.582	0.581	0.588	0.676	0.687	0.694
camel-1.4	0.783	0.829	0.766	0.775	0.592	0.598	0.700	0.699	0.661	0.662	0.682	0.671	0.811	0.860	**0.860**
camel-1.6	0.758	0.783	0.774	0.783	0.583	0.588	0.671	0.674	0.649	0.647	0.639	0.644	0.794	0.836	**0.848**
jedit-3.2	0.804	**0.814**	0.794	0.787	0.736	0.725	0.789	0.800	0.782	0.775	0.791	0.776	0.775	0.804	0.797
jedit-4.0	0.823	**0.850**	0.822	0.823	0.670	0.699	0.741	0.729	0.714	0.714	0.712	0.723	0.793	0.813	0.829
jedit-4.1	0.844	**0.862**	0.814	0.817	0.704	0.706	0.798	0.797	0.758	0.764	0.760	0.759	0.786	0.845	0.838
jedit-4.2	0.865	**0.881**	0.853	0.852	0.741	0.735	0.820	0.817	0.777	0.786	0.793	0.803	0.861	0.878	0.880
jedit-4.3	0.971	**0.973**	0.942	0.941	0.718	0.704	0.869	0.878	0.769	0.776	0.784	0.784	0.955	0.968	0.972
log4j-1.0	0.832	**0.862**	0.788	0.799	0.752	0.752	0.768	0.770	0.768	0.775	0.779	0.771	0.786	0.818	0.839
log4j-1.1	0.798	0.812	**0.815**	0.790	0.760	0.780	0.760	0.787	0.777	0.781	0.782	0.769	0.760	0.781	0.807
lucene-2.0	0.668	0.681	0.651	0.658	0.647	0.675	0.685	0.681	0.693	0.690	**0.711**	0.688	0.630	0.656	0.693
xalan-2.4	0.828	**0.855**	0.826	0.828	0.646	0.654	0.739	0.740	0.733	0.739	0.732	0.735	0.830	0.847	0.854
xalan-2.5	0.649	**0.668**	0.629	0.633	0.558	0.547	0.600	0.602	0.593	0.595	0.590	0.581	0.641	0.657	0.660
xalan-2.6	0.733	0.750	0.736	0.738	0.691	0.693	0.719	0.721	0.724	0.721	0.710	0.711	0.720	**0.763**	0.755
F-Test	4.81	**2.13**	6.06	5.94	14.5	13.63	9	8.06	11.13	10.63	10.19	10.75	7.13	3.81	2.25
Rank	4	**1**	6	5	15	14	9	8	13	11	10	12	7	3	2

VEBHHO [25], EBMFOV3 [5], (LR, NB, 5NN, C4.5) [49], Bayesian networks [50], and (L-RNN, NB) [7]. Based on Table 9 results, it can be recognized that the proposed model outperforms the other competitors in most cases. Additionally, ensemble MLP displays a superiority in the classification of around 85 % of the datasets over the recently published VEBHHO [25] approach and 100 % of the datasets over the recently published EBMFOV3 [5] . In this way, our proposed approach proves its efficiency and accuracy in classifying the majority of the datasets over the works in literature.

Table 9: Comparison between the proposed approach and the state-of-the-art methods in terms of AUC rates

Dataset	Our results		Thaher and Arman [25]	Tumar et. al [5]	Shatnawi [49]				Okutan and Yildiz [50]	Turabieh and Mafarja [7]	
	ensemble MLP	RF	VEBHHO	EBMFOV3	LR	NB	5NN	C4.5	Bayesian networks	L-RNN without CV	NB
ant-1.7	0.821	**0.853**	0.7727	0.7615	0.830	0.790	0.760	0.740	0.820	0.523	0.7261
camel-1.0	**0.961**	0.954	0.8107	0.7552	-	-	-	-	-	0.607	0.8881
camel-1.2	**0.699**	0.694	0.6467	0.6215	0.570	0.560	0.640	0.520	-	0.443	0.5531
camel-1.4	0.829	**0.860**	0.7029	0.6918	0.700	0.670	0.670	0.600	-	0.521	0.6603
camel-1.6	0.783	**0.848**	0.6762	0.6580	0.650	0.590	0.660	0.540	-	0.505	0.6878
jedit-3.2	0.814	0.797	**0.8270**	0.8053	-	-	-	-	-	0.518	0.6777
jedit-4.0	**0.850**	0.829	0.7661	0.7161	0.770	0.700	0.810	0.720	-	0.549	0.7055
jedit-4.1	**0.862**	0.838	0.8133	0.7868	0.820	0.750	0.800	0.690	-	-	
jedit-4.2	**0.881**	0.880	0.8290	0.7973	0.840	0.750	0.770	0.640	-	0.519	0.8352
jedit-4.3	**0.973**	0.972	0.8081	0.7499	-	-	-	-	0.658	0.684	0.8273
log4j-1.0	**0.862**	0.839	0.8297	0.7937	-	-	-	-	-	0.535	0.8455
log4j-1.1	0.812	0.807	0.8395	0.7986	-	-	-	-	-	0.522	**0.897**
lucene-2.0	0.681	0.693	-	-	0.770	0.750	0.700	0.670	-	0.553	**0.8136**
xalan-2.4	**0.855**	0.854	0.7521	0.7495	-	-	-	-	-	0.536	0.7621
xalan-2.5	**0.668**	0.660	-	-	-	-	-	-	0.624	0.510	0.6575
xalan-2.6	0.750	**0.755**	-	-	-	-	-	-	-	0.506	0.64

7. Conclusion and future works

In this paper, a classification framework based on the aggregation of multiple MLP classifiers (ensemble MLP) and SMOTE technique was proposed with the aim of enhancing the prediction performance for the SFP problem. Ensemble learner was employed as a classification model, while SMOTE was utilized to handle the problem of imbalanced data. Sixteen real-world object-oriented

projects from the PROMISE repository were utilized to evaluate the proposed model. The experimental results demonstrated that the MLP classifier is significantly sensitive to its parameters. It was also noted that the ensemble methods have a more significant influence on the performance of the SFP model than individual classifiers used in comparisons. Comparison results revealed that our proposed approach is efficient in handling the SFP problem compared to well-known classifiers such as KNN, LDA, LR, DT, and SVM as well as previous works. ALL in all, the proposed ensemble learner using MLP as a base classifier combined with SMOTE is recommended for the SFP problem in terms of prediction quality.

The future work will investigate swarm intelligence meta-heuristics to train MLP. We will also exploit other ensemble methods such as boosting and stacking. Another exciting challenge we will plan to check is the adoption of SFP for the rapidly changing environment of Agile-based models such as extreme programming where enough data to train the model is not available at the early stages of the SDLC.

References

[1] P. Ralph, Software engineering process theory: A multi-method comparison of sensemaking-coevoiution-implementation theory and function-behavior-structure theory, Information and Software Technology 70 (2016) 232–250.

[2] I. Sommerville, Software Engineering, Pearson, 10th edition, 2015.

[3] N. Honest, Role of testing in software development life cycle, International Journal of Computer Sciences and Engineering 7 (2019) 886–889.

[4] S. Rathore, S. Kumar, A study on software fault prediction techniques, Artificial Intelligence Review (2017) 1–73.

[5] I. Tumar, Y. Hassouneh, H. Turabieh, T. Thaher, Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction, IEEE Access 8 (2020) 8041–8055.

[6] O. Qasem, M. Akour, Software fault prediction using deep learning algorithms, International Journal of Open Source Software and Processes 10 (2019) 1–19.

[7] H. Turabieh, M. Mafarja, X. Li, Iterated feature selection algorithms with layered recurrent neural network for software fault prediction, Expert Systems with Applications 122 (2018).

[8] D. Gupta, K. Saxena, Software bug prediction using object-oriented metrics, Sadhana - Academy Proceedings in Engineering Sciences 42 (2017) 655–669.

[9] P. Deep Singh, A. Chug, Software defect prediction analysis using machine learning algorithms, in: 2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence, pp. 775–781.

[10] T. Thaher, N. Arman, Efficient multi-swarm binary harris hawks optimization as a feature selection approach for software fault prediction, in: 2020 11th International Conference on Information and Communication Systems (ICICS), pp. 249–254.

[11] T. Khoshgoftaar, J. Van Hulse, A. Napolitano, Comparing boosting and bagging techniques with noisy and imbalanced data, IEEE Transactions on Systems, Man, and Cybernetics, Part A 41 (2011) 552–568.

[12] T. Thaher, M. Mafarja, B. Abdalhaq, H. Chantar, Wrapper-based feature selection for imbalanced data using binary queuing search algorithm, in: 2019 2nd International Conference on new Trends in Computing Sciences (ICTCS), pp. 1–6.

[13] A. Singh, R. Bhatia, A. Singhrova, Taxonomy of machine learning algorithms in software fault prediction using object oriented metrics, Procedia computer science 132 (2018) 993–1001.

[14] F. Xing, P. Guo, M. Lyu, A novel method for early software quality prediction based on support vector machine, volume 2005, pp. 10 pp.–.

[15] T. M. Khoshgoftaar, N. Seliya, Software quality classification modeling using the sprint decision tree algorithm, International Journal on Artificial Intelligence Tools 12 (2003) 207–225.

[16] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, K. Ganesan, An application of fuzzy clustering to software quality prediction, in: Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on, IEEE, pp. 85–90.

[17] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE transactions on software engineering 33 (2007) 2–13.

[18] V. kumar Dwivedi, M. K. Singh, Software defect prediction using data mining classification approach, International Journal of Technical Research and Applications 4 (2016) 31–35.

[19] G. Carrozza, D. Cotroneo, R. Natella, R. Pietrantuono, S. Russo, Analysis and prediction of mandelbugs in an industrial software system, in: Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on, IEEE, pp. 262–271.

[20] M. Bisi, N. Goyal, Early prediction of software fault-prone module using artificial neural network, International Journal of Performability Engineering 11 (2015) 43–52.

[21] B. Caglayan, A. Tosun, A. Bener, A. Miranskyy, Predicting defective modules in different test phases, Software Quality Journal 23 (2014).

[22] S. Yogesh, K. Arvinder, R. Malhotra, Software fault proneness prediction using support vector machines, Lecture Notes in Engineering and Computer Science 2176 (2009).

[23] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, R. X. Gao, Deep learning and its applications to machine health monitoring, Mechanical Systems and Signal Processing 115 (2019) 213 – 237.

[24] S. Rathore, S. Kumar, Towards an ensemble based system for predicting the number of software faults, Expert Systems with Applications 82 (2017).

[25] T. Thaher, A. A. Heidari, M. Mafarja, J. S. Dong, S. Mirjalili, Binary Harris Hawks Optimizer for High-Dimensional, Low Sample Size Feature Selection, Springer Singapore, Singapore, pp. 251–272.

[26] A. O. Balogun, S. Basri, S. J. Abdulkadir, A. S. Hashim, Performance analysis of feature selection methods in software defect prediction: A search method approach, Applied Sciences 9 (2019) 2764.

[27] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Information Sciences 179 (2009) 1040–1058.

[28] D. Wolpert, W. Macready, No free lunch theorems for optimization, Evolutionary Computation, IEEE 1 (1997) 67–82.

[29] P. Larranaga, B. Calvo, R. Santana, C. Bielza, J. Galdiano, I. Inza, J. Lozano, R. Armañanzas, G. Santafé, A. Pérez, V. Robles, Machine learning in bioinformatics, Briefings in bioinformatics 7 (2006) 86–112.

[30] H. He, E. Garcia, Learning from imbalanced data, Knowledge and Data Engineering, IEEE Transactions on 21 (2009) 1263 – 1284.

[31] R. Mohammed, J. Rawashdeh, M. Abdullah, Machine learning with oversampling and undersampling techniques: Overview study and experimental results, in: 2020 11th International Conference on Information and Communication Systems (ICICS), pp. 243–248.

[32] N. Chawla, K. Bowyer, L. O. Hall, W. P. Kegelmeyer, Smote: Synthetic minority over-sampling technique, J. Artif. Intell. Res. (JAIR) 16 (2002) 321–357.

[33] V. Kumar, S. Minz, Feature selection: A literature review, Smart Computing Review 4 (2014) 211–229.

[34] M. M. Mafarja, S. Mirjalili, Hybrid whale optimization algorithm with simulated annealing for feature selection, Neurocomputing 260 (2017) 302 – 312.

[35] E. Amrieh, T. Hamtini, I. Aljarah, Mining educational data to predict student's academic performance using ensemble methods, International Journal of Database Theory and Application 9 (2016) 119–136.

[36] L. Fausett, Fundamental of Neural Networks: Architectures, Algorithms, and Applications, 1993.

[37] H. Faris, I. Aljarah, S. Mirjalili, Training feedforward neural networks using multi-verse optimizer for binary classification problems, Applied Intelligence 45 (2016) 322–332.

[38] I. Franc, N. Macek, M. Bogdanoski, D. Đokić, Detecting malicious anomalies in iot: Ensemble learners and incomplete datasets, in: The Eight International Conference on Business Information Security (BISEC).

[39] J. Sayyad Shirabad, T. Menzies, The PROMISE Repository of Software Engineering Databases., School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

[40] E. Erturk, E. A. Sezer], Iterative software fault prediction with a hybrid approach, Applied Soft Computing 49 (2016) 1020 – 1033.

[41] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, volume 9, p. 9.

[42] S. R. Chidamber, C. F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (1994) 476–493.

[43] R. D. Martin, Object oriented design quality metrics: an analysis of dependencies.

[44] B. Henderson-seller, Object-oriented metrics: Measures of complexity (1996).

[45] J. Bansiya, C. G. Davis, A hierarchical model for object-oriented design quality assessment, IEEE Transactions on Software Engineering 28 (2002) 4–17.

[46] Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen, An empirical study on object-oriented metrics, in: Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403), pp. 242–249.

[47] T. J. McCabe, A complexity measure, IEEE Transactions on Software Engineering SE-2 (1976) 308–320.

[48] T. Windeatt, Ensemble MLP Classifier Design, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 133–147.

[49] R. Shatnawi, The application of roc analysis in threshold identification, data imbalance and metrics selection for software fault prediction, Innovations in Systems and Software Engineering 13 (2017) 201–217.

[50] A. Okutan, O. Yildiz, Software defect prediction using bayesian networks, Empirical Software Engineering 19 (2014).

Appendix A.

Table A.10: Average AUC results for MLP classifier using different values of main parameters

parameter	Optimizor		No. hidden layers				Activation function			
Dataset	SGD	adam	2	3	4	5	tanh	ReLU	logistic	identity
ant-1.7	0.7281	**0.7614**	0.7614	0.7708	0.7790	**0.7802**	0.7708	**0.7896**	0.7597	0.7643
camel-1.0	0.8158	**0.9318**	0.9318	**0.9534**	0.9475	0.9347	0.9534	**0.9638**	0.7947	0.8653
camel-1.2	0.5437	**0.6314**	0.6314	**0.6456**	0.6356	0.6422	0.6456	**0.6649**	0.5827	0.5986
camel-1.4	0.6541	**0.7409**	0.7409	**0.7612**	0.7609	0.7493	0.7612	**0.7834**	0.6919	0.6841
camel-1.6	0.6302	**0.7333**	0.7333	**0.7350**	0.7250	0.7335	0.7350	**0.7577**	0.6683	0.6546
jedit-3.2	0.7467	**0.7917**	0.7917	0.7912	0.8009	**0.8009**	0.7912	**0.8039**	0.7041	0.7799
jedit-4.0	0.6999	**0.7799**	0.7799	0.8012	0.8023	**0.8217**	0.8012	**0.8226**	0.7345	0.7430
jedit-4.1	0.7584	**0.7996**	0.7996	**0.8210**	0.8011	0.8068	0.8210	**0.8444**	0.7698	0.7990
jedit-4.2	0.7971	**0.8092**	0.8092	0.8136	**0.8367**	0.8301	0.8136	**0.8648**	0.7585	0.8060
jedit-4.3	0.7924	**0.9557**	0.9557	**0.9608**	0.9560	0.9548	0.9608	**0.9708**	0.8785	0.8890
log4j-1.0	0.7558	**0.7934**	0.7934	0.8054	**0.8435**	0.8198	0.8054	**0.8316**	0.7570	0.7713
log4j-1.1	**0.7901**	0.7886	0.7886	**0.8026**	0.7651	0.7999	**0.8026**	0.7980	0.7865	0.7730
lucene-2.0	0.6796	**0.6839**	**0.6839**	0.6728	0.6633	0.6636	0.6728	0.6682	0.6619	**0.6790**
xalan-2.4	0.7359	**0.7809**	0.7809	0.7917	0.7921	**0.7922**	0.7917	**0.8275**	0.7414	0.7323
xalan-2.5	0.5602	**0.6413**	0.6413	0.6322	**0.6465**	0.6409	0.6322	**0.6486**	0.6057	0.6164
xalan-2.6	0.7229	**0.7292**	**0.7292**	0.7203	0.7242	0.7194	0.7203	**0.7332**	0.6862	0.7257
Rank (F-Test)	1.94	**1.06**	3.25	**2.19**	2.31	2.25	2	**1.19**	3.75	3.06