CFIWSE: A Hybrid Preprocessing Approach for Defect Prediction on Imbalance Real-World Datasets

Jiaxi Xu, Jingwei Shang*, and Zhichang Huang
China Electronic Product Reliability and Environmental Testing Research Institute, Guangzhou, Guangdong, China
hickeyhsu@buaa.edu.com, shangjingwei@ceprei.com, huangzhichang@ceprei.com
*corresponding author

***Abstract*—Software Defect Prediction (SDP) predicts new defects through machine learning trained with historical defect data. The distribution of software defects is highly unbalanced, which hinders the construction of defect prediction models. In addition, previous studies were usually validated by public datasets based on code metrics instead of real-world data. In this work, SNA metrics and code metrics are computed on 9 representative real-world projects. A hybrid preprocessing approach for defect prediction named CFIWSE is proposed to improve SDP performance through feature selection, minority sample synthesis and noise reduction, consisting of CFS and IWSE. CFS uses correlation analysis and nearest neighbor theory for feature selection. IWSE utilizes information weights and edited nearest neighbor rule to alleviate overfitting and noise introduced from minority sample synthesis. The proposed method is verified by experiments on real-world data, and the contribution of the method components and parameter sensitivity are explored.**

Keywords- Software defect prediction; Feature selection; SMOTE; Minority sample synthesis.

I. INTRODUCTION

Software defect prediction (SDP) is now a hot subject matter of software dependability because it gives direction for software testing, minimizes overhead, and enhances the cost-effectiveness ratio. Currently, forming a high number of software tests during the software development process is an essential method for enhancing software reliability. With the increasing expansion of software size and complexity, machine learning-based software defect prediction technology has become an integral component of software reliability assurance activities. By utilizing the historical defect data to its fullest extent, it is possible to identify the potential defect modules that may be hidden in the software. This enables software testers to test the software more effectively and pertinently, increase the likelihood of discovering hidden defects, and ensure the quality of the software.

A. Motivation

Numerous academics have undertaken numerous experiments on classifier algorithms for constructing defect prediction models for a very long time. They applied various classifiers to public datasets and evaluated the effectiveness of the model using metrics. Various articles provide varying conclusions regarding the appropriate classifier algorithm for software defect prediction. In 2018, Agrawal et al.[1] suggested that the enhancement of training data quality also considerably enhanced the performance of software defect prediction.

Some papers have added relevant processing methods to address the issue of dataset quality. Most articles utilize recursive feature elimination with cross-validation (*RFECV*) to address the problem of redundant and incorrect features. Typically, SMOTE technology is employed to fix the class imbalance. Numerous noise reduction technologies are available for the potential noise samples in the dataset.

Current research frequently uses CK measurement data from NASA and PROMISE public sources to confirm their findings. These public datasets have a small number of defect features, and their class imbalance problem is less severe than that of real-world software data. In a recent study, Gong et al.[2] suggested that software network analysis (SNA) metrics should be considered in SDP scenarios alone or in combination with code metrics, whereas almost no SNA metrics have been used in recent software defect prediction model studies.[3]

In real-world datasets using SNA metrics, datasets tend to have more features and more severe class imbalances, and the effectiveness of the commonly used feature filtering and SMOTE methods cannot be guaranteed. To process the software defect dataset, a novel hybrid software defect dataset processing method (*CFIWSE*) is proposed, which combines correlation analysis-based feature selection (*CFS*) and information weighted synthetic sampling approach with edited nearest neighbor rule (*IWSE*). Ensemble learning classifiers are used to build prediction models.

To evaluate the proposed method on real-world datasets, we have also developed a defect data extraction tool that creates hybrid dependency graphs and calculates SNA metric data from software source codes for various programming languages. As ground truth, we utilized 32 versions of defect reports from 9 software versions in the JIRA error reporting system collected by Yatish et al.[4]. We then generated datasets from the source code retrieved from the Apache Software repository and conducted a number of experimental validations.

B. Contribution

Overall, our work mainly includes the following contributions:

1. Considering the similarity and difference of SNA features, *CFIWSE* combines feature-feature correlation

2693-9371/22/$31.00 ©2022 IEEE
DOI 10.1109/QRS-C57518.2022.00064

and feature-defect correlation for feature selection, effectively reducing irrelevant and redundant features.

2. To repair the class imbalance problem of defect datasets, avoid the over-generalization of minority class samples and reduce noise samples, *CFIWSE* improves minority sample synthesis through three aspects: sample selection, sample generation and sample cleaning.
3. To illustrate the effectiveness of our proposed *CFIWSE* method, we developed a cross-language software SNA metric data extraction tool to generate datasets from 32 versions of code from 9 real-world software and conduct extensive experiments. The experimental results show that our proposed *CFIWSE* method improves the AUC on this dataset by an average of 4-13% over *SMOTUNED*[1], the currently widely accepted state-of-the-art in SDP, and 1-15% over *SYMPROD*[5], the state-of-the-art in data balancing.
4. Ablation studies were also conducted to experiment with feature selection methods, sample synthesis methods, sample synthesis ratios tuning in *CFIWSE*. The conclusions show that both *CFS* and *IWSE* contribute positively to the defect prediction performance and are better to use combined. Fine-tuning the sample synthesis ratio can further improve the model performance in most cases.

C. Paper Organization

The remainder of the paper is organized as follows: Section II gives brief reviews of software defect prediction and data processing. Section III introduces our proposed method *CFIWSE*. Section IV provides the experimental setting. The results are provided and analyzed in Section V. Section VI discloses the threats to validity. Section VII draws the conclusions.

II. Related Works

This section mainly discusses the current literature work in the field of software defect prediction based on machine learning, including software defect data, defect feature selection, sample sampling technology and classifier algorithms.

Machine learning is the mainstream method of defect prediction models at present[6]. Many researchers have verified the effectiveness of machine learning for software module risk prediction, and their conclusions on the optimal classifier algorithm are different. In the latest research, Alazba et al.[7] Believe that the best classifier model can be obtained by optimizing the hyperparameters of the tree-based ensemble learning models.

Software metrics: Software metrics refer to quantified features extracted from software from different dimensions as input to a machine learning classifier.

The earliest software metric applied to SDP was the measurement of code and its complexity. As early as 1978, Halstead et al. [21] proposed the Halstead metric for code size. In 1994, Chidamber and Kmerer [22] proposed the famous CK metric element, which became one of the standard metric tuples in the SDP field; in the same year, Abreu et al. [23] proposed the MOOD metric element. In 1996, Ohlsson et al. [24] applied McCabe cyclomatic complexity to an Ericsson telecom system. In 2007, Menzies et al. [25] compared the impact of various code metrics on SDP. Today, code metrics, especially CK metrics, are still one of the mainstream metrics in the SDP field.

Although code metrics and process metrics have been widely used in SDP, as technology advances and software structures continue to become more complex, the shortcomings of these metrics, such as the lack of representation of software structures and software element associations, have become a bottleneck in the development of SDP. In 2008, Zimmermann et al. [29] predicted defects by relying on graph-centric metrics.

In the subsequent period from 2009-2015, several researchers have applied SNA metrics to different software and classifiers ([7], [14]–[17]). However, its application in the field of software defect prediction is still rare due to the difficulty of software network construction and analysis, which often requires software bytecodes. Gong et al.[2] conducted extensive experiments on software defect prediction techniques based on dependency graphs and suggested that SNA metrics alone or together with code metrics should be considered in SDP scenarios, while recent software defect prediction model studies have rarely used the SNA metric.

Dataset: Public datasets such as MDP and PROMISE are widely used for experimental verification in the research of machine learning-based software defect prediction. However, many researchers have questioned the quality and credibility of public datasets. In 2009, Christian Bird et al.[18] questioned the fairness and authenticity of public datasets, and found that these defective data posed a serious challenge to the validity and universality of defect prediction models. In 2013, Shepperd et al.[19] compared the defect data from two versions of MDP and PROMISE, and found that quality problems such as data loss, value anomaly, and instance repetition, and their differences had significant influence on software defect prediction results. David Gray et al.[20] analyzed the characteristics of 13 NASA raw datasets and designed a data cleaning process. Each dataset has 6% -90% of records to be cleaned. Research should therefore be validated in real-world datasets.

Feature selection: By eliminating redundant and unnecessary features and lowering the time and space complexity of the algorithms, feature selection aims to increase the accuracy of software defect prediction models. The two most common types of feature selection techniques are filter-based and wrapper-based.

The filter-based feature selection methods did not consider the correlation within the feature, and because its evaluation criteria are independent of specific learning algorithms, the classification accuracy is low. The feature generality of wrapper-based feature selection methods is not strong, the subset space exponentially explodes with the increase in the number of features, and the algorithm performance decreases.

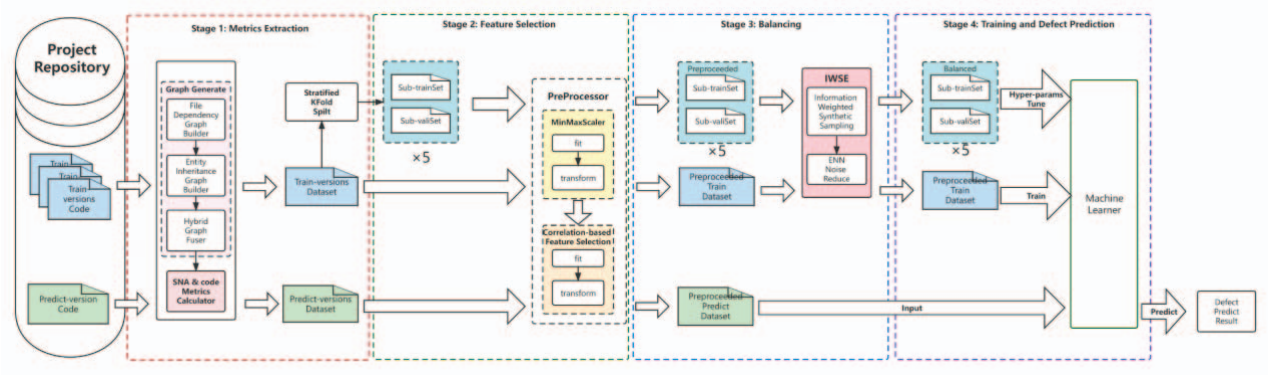

Figure 1 Framework of *CFIWSE*

Balancing: The problem of class imbalance has been regarded as a key problem in machine learning and data mining, which refers to the serious imbalance of the proportion of different classes of instances in the dataset. Initially, people balanced the proportion of datasets by reducing the sampling of the major class (undersampling), but this method would greatly reduce the amount of data. The SMOTE algorithm proposed by Chawla et al. [21] is one of the most commonly used class balance methods in academia, but this method may increase the risk of overfitting. To this end, people have proposed variants of SMOTE. Batista et al. [22] proposed SMOTE_TomekLinks and SMOTE_ENN. Han et al. [23] proposed Borderline-SMOTE. He et al. [24] added different weights for different minority instances (ADASYN). Douzas et al. applied k-means clustering [25] and self-organizing map [26] to the SMOTE method. Lee et al.[27] added Gaussian random variables to the SMOTE synthesis sample process. Barua et al. [28] gave the minority sample weight based on the distance to the nearest majority sample. In 2017, Rivera [29] introduced propensity score matching to the balancing and noise reduction methods.

III. METHODOLOGY

In this section, we introduce our novel hybrid software defect data-processing approach *CFIWSE*, with two main components: correlation-based feature selection (*CFS*) and information weighted synthetic sampling approach with edited nearest neighbors' rule (*IWSE*).

A. Framework

This section will introduce the framework of our method, which is divided into four stages, as shown in Figure 1: 1. Graph Generation and Metrics Calculation; 2. Correlation-based Feature Selection (*CFS*); 3. Information weighted Sample Synthesis approach with Edited nearest neighbors rule (*IWSE*); 4. Training and predicting.

In the first phase, software file dependency and entity inheritance are analyzed based on the open source analysis tool EMERGE[1] and fused to generate a hybrid dependency graph. Then, based on the formulas proposed by Chidamber[9] et al and Yang et al.[30], SNA metrics and code metrics are calculated from the hybrid dependency graph to generate training datasets and predicted datasets.

In the second stage, we improved the *CHIFS* algorithm based on Wang et al[31] and propose correlation-based feature selection (*CFS*). Logistic regression was used to analyze the feature-defect correlation, excluding the unrelated features. The Pearson correlation coefficient is used to evaluate the feature-feature correlation, and the K nearest neighbor algorithm is introduced instead of the full join method used by Wang et al. to generate the feature correlation connection graph. After spectral clustering, the feature quality coefficients are calculated according to the feature-feature correlation and the feature-defect correlation, and the feature quality coefficients are used as the search order to select the prefix features.

In the third stage, we propose information weighted synthetic sampling approach with edited nearest neighbors' rule (*IWSE*), which introduce information weight in *SMOTE*[21] to select more important samples, and use the Edited Nearest-neighbors algorithm (*ENN*) [22] to exclude outliers from the synthetic samples and reduce the noise introduced by *SMOTE*.

In the fourth stage, cross validation is used to optimize the machine learning classifier, and the training set is used to train the optimized machine learning classifier for the defect prediction task of the dataset to be predicted.

B. Data Extraction

One of the most crucial components of SDP based on machine learning is data. As indicated in Section II, many questions have been raised about the reliability and validity of the commonly utilized public datasets. Recently, Yatish et al.[4] established a carefully selected benchmark defect dataset by mining the closed issues reports from 9 representative open source software in the JIRA problem management system, linking these problem reports to the specified earliest affected version.

We use Yatish's dataset[4] as the ground truth for a number of reasons: 1. Software types in the dataset include search engine libraries, programming languages, databases, and integration frameworks. 2. The range of the class imbalance ratio is broad. 3. Software size ranges from large (Camel, Derby), medium (ActiveMQ, HBase, Hive, Wicket), to tiny (Groovy, JRuby, Lucene). We obtained the

1 https://github.com/glato/emerge

corresponding source code for each version for metric calculation from GitHub and Apache Repository.

Code Metrics: We use the code metrics calculator developed by Aniche[32] to calculate the code metrics consisting of 7 CK metrics and 36 code statistical metrics.

SNA Metrics: First, the open source code analysis tool EMERGE is used to extract file dependence and entity inheritance relations from the software code. EMERGE is a source code static analysis tool that, unlike methods employed by other research institutions in SNA Metrics, does not require software code to be built, allowing it to be used for software development processes. We fuse the file dependency graph and entity inheritance graph from EMERGE analysis through file path matching, attach the entity inheritance relation to the file element where the entity is located, and obtain a hybrid dependency graph with inheritance relationships.

We calculated a total of 30 SNA metrics from the hybrid dependency graph based on the SNA metrics calculation method proposed by Yang et al.[30].

Table 1 Metrics we calculated

Calculator	Metric Type	Source Paper	Count
code metrics calculator	CK Metrics	[22]	7
	Code Statistical Metrics	[32]	36
SNA metric calculator	SNA metrics	[30]	30

C. Correlation-based Feature Selection (CFS)

After introducing the SNA metric, the dataset contains 73 features in total. To eliminate irrelevant and redundant features, feature selection is necessary. Section II introduces the commonly used wrapping-based and filtering-based feature selection methods and their disadvantages. To eliminate these drawbacks, we start with feature correlation and combine the wrapped and filtered feature selection methods.

The feature correlation we consider includes: 1) correlation between features and defect distribution (Feature-defect Correlation); 2) the correlation between one feature and another (Feature-feature Correlation). We propose a conjecture that features with low feature-defect correlation are likely to be irrelevant features; feature subsets with high feature-feature correlation may have redundant features. We improve the *CHIFS* method proposed by Wang et al.[31], and propose a correlation-based feature selection approach(*CFS*).

1) *Feature-defect Correlation Analysis*

First, the irrelevant features are excluded by a filter-based method. Some studies[33] have shown that logistic regression has good performance in predicting software defects, so we used logistic regression models to analyze the correlation of each feature with software defects. We define $\mathrm{X} = \{(x_i, \mathrm{y}_i)|\mathrm{i} = 1,2,\dots,\mathrm{n}\}$ as a software defect dataset consisting of n samples. For each defect sample $x_i = (x_i^1, x_i^2, \dots, x_i^m)$, m is feature amount of x_i , and $\mathrm{y_i}$ is the defect class of x_i (1 for positive, 0 for negative). To avoid the model parameters being dominated by data with a large or small distribution range, the dataset is first normalized using a min-max normalization algorithm to unify the scale of each feature.

After the normalized dataset $\overline{\mathrm{X}}$ is obtained, the probability $\mathrm{P_i}$ that sample x_i is positive is calculated using multivariate logistic regression:

$$\mathrm{P_i} = \Pr\left(\mathrm{y_i} = 1|\left(x_i^1, x_i^2, \dots, x_i^{\mathrm{m}}\right)\right) = \frac{e^{\theta_0+\theta_1 x_i^1+\dots+\theta_m x_i^m}}{1+\mathrm{e}^{\theta_0+\theta_1 x_i^1+\dots+\theta_m x_i^m}} \quad (1.)$$

A multivariate logistic regression model can be fitted by using the maximum likelihood estimate to obtain the regression parameter$\theta = (\theta_0, \theta_1, \dots, \theta_{\mathrm{m}})$. The odd ratio (OR) is a commonly used indicator of logistic regression model, which reflects the strength of the association between the independent variable and the dependent variable. OR>1 suggests that the characteristics and software defects were positively correlated, OR<1 suggests that the feature and software defects were negatively correlated, and OR=1 suggests the feature has a low correlation with software defects. As suggested by Wang et al.[31], we considered features with OR values in the interval (0.67, 1.50) as not significantly related to software defects and removed them from the feature set.

2) *Feature-feature Correlation Analysis*

The model adaptability decreases when there is a set of features with strong internal correlation in the feature set. Therefore, we used Pearson's correlation coefficient to evaluate the correlation between features. For the two features a and b, a_i and $b_i$are the values of features a and b in the ith sample. The Pearson coefficient between a and b is calculated as:

$$r(a,b) = \frac{\sum(a_i-\bar{a})(b_i-\bar{b})}{\sqrt{\sum(a_i-\bar{a})^2 \cdot \sqrt{(b_i-\bar{b})^2}}}, r(a,b) \in [-1,1] \quad (2.)$$

Features are positively correlated when the Pearson coefficient is positive and negatively correlated when the Pearson coefficient is negative. When the Pearson coefficient is equal to zero, the two variables are not correlated.

Wang et al. joined all features pairwise to form a fully connected graph so that the similarity matrix $\mathrm{W} = \{\mathrm{W}_{ij}|\mathrm{W}_{ij} = \mathrm{r}(f_i, f_j); \mathrm{i}, \mathrm{j} = 1,2,\dots,\mathrm{m}\}$, f_i refers to the i-th feature. However, in the real-world dataset with SNA features, the feature dimension is high, and the dataset size is large, which causes great difficulties with the iteration of the fully connected graph. We introduce the K-nearest neighbors (KNN) algorithm into the construction of the similarity matrix to make the similarity matrix sparser:

$$\mathrm{W}_{ij} = \mathrm{W}_{ji} = \begin{cases} \mathrm{r}(f_i, f_j), & f_i \in KNN(f_j) \text{ or } f_j \in KNN(f_i) \\ 0, & \text{else} \end{cases} \quad (3.)$$

Degree matrix: $D_i = \sum W_{ij}$

Laplacian matrix: $\mathrm{L} = \mathrm{D} - \mathrm{W}$

Normalized Laplacian matrix: $L_{rw} = D^{-1}L = I - D^{-1}W$

For that number of cluster cores $\mathrm{k} = 1,2,\dots,\mathrm{m}$, compute the first k eigenvector of L_{rw}, and form the matrix $\mathrm{U} = [\mathrm{u}_1, \mathrm{u}_2, \dots, \mathrm{u}_\mathrm{k}] \in R^{m\times k}$.

3) *Feature Selection*

The row vectors $\{\mathrm{y_i} \in \mathrm{U}|\mathrm{i} = 1,2,\dots,\mathrm{m}\}$ are clustered using the K-means clustering algorithm. The product of the Silhouette coefficient and the Cohesion degree is used to evaluate the cluster results and determine the optimal cluster core number k.

Then, according to the calculation method proposed by Wang et al., the feature-feature metric parameter $FFCi = r(f_i, core_j)$and the feature-defect metric parameter $FTCi = r(f_i, target)$ are calculated, where $core_j$ is the cluster core where f_i is located, and $target$ is the defect distribution. The quality coefficient of feature f_i is calculated as follows:

$$Qi = \frac{FTCi}{exp(\theta(FFCi - 0.8)) + 1} \quad (4.)$$

Sequential forward selection was performed based on Qi, and the optimal feature subset was selected based on the average AUC of K-fold cross-validation.

D. Information Weighted Sample Synthesis with Edited Nearest Neighbors Rule

At present, the most popular dataset balancing method in machine learning is *SMOTE*[21]. However, this method cannot avoid the problems of overfitting and noise amplification due to the synthetic data strategy. Inspired by *MWMOTE*[28] and *SMOTE+ENN*[22], this work introduced both information weights and Wilson's Edited Nearest Neighbor Rule (*ENN*) into the Synthetic Minority Over-sampling Technique.

Information weighted sample synthesis with edited nearest neighbors' rule (*ISWE*) is proposed, which improves minority sample synthesis through three aspects: sample selection, sample generation and sample cleaning.

Han et al.[23], who proposed *Borderline-SMOTE*, suggested that samples located near the decision boundary had a significant impact on model performance. Barua et al.[28], who proposed *MWMOTE*, implied that samples belonging to the small-sized cluster (s) were also important. In line with their conclusions, *IWSE* uses the *ENN* algorithm to clean up noise after minority sample synthesis instead of deleting samples before synthesis, in order to preserve as much information as possible for a few classes of samples.

First, for each minority sample x_i, KNN is used to compute the majority of its k-nearest-neighbors $\mathrm{N_{i,maj}} = \{\mathrm{x_{(i,1)}}, x_{(i,2)}, \dots, x_{(i,k)}\}$. $S_{maj} = \{x_j^{maj}\}$ of all samples in $\mathrm{N_{i,maj}}$ indicates the boundary of most classes. For each majority boundary sample $x_i^{maj} \in S_{maj}$, its minority k-nearest-neighbors set $\mathrm{N_{i,min}} = \{\mathrm{x_{(i,1)}}, x_{(i,2)}, \dots, x_{(i,k)}\}$ is calculated. The minority sample set S_{min} of all $\mathrm{N_{i,min}}$ indicates the minority boundary.

After minority boundary samples selection, the importance of samples is evaluated by information weights. For each minority sample $x_i \in S_{min}$, calculate its information weight with every majority boundary sample $x_j^{maj} \in S_{maj}$:

$$\mathrm{I_w}\left(x_j^{maj}, x_i\right) = \mathrm{C_f}\left(x_j^{maj}, x_i\right) \times \mathrm{D_f}\left(x_j^{maj}, x_i\right) \quad (5.)$$

$$C_f\left(x_j^{maj}, x_i\right) = \begin{cases} f\left(\frac{l}{dist\left(x_j^{maj}, x_i\right)}\right) \\ \hline C_f^{therhold} \end{cases} \times C_{MAX}, \quad x_i \in \mathrm{N_{min}}(x_j^{maj}) \quad (6.)$$

$$0, \quad else$$

$$D_f\left(x_j^{maj}, x_i\right) = \frac{C_f\left(x_j^{maj}, x_i\right)}{\sum_{q \in S_{min}} C_f\left(x_j^{maj}, q\right)} \quad (7.)$$

$C_f^{therhold}$ and C_{MAX} are set by users. Sum the information weights to obtain the selection weights:

$$S_w(x_i) = \sum_{x_i^{maj} \in S_{maj}} I_w\left(x_j^{maj}, x_i\right) \quad (8.)$$

The larger $S_w(x_i)$ is, the more important x_i is. Normalize the selection weight to obtain the sample selection probability:

$$\mathrm{p}(x_i) = \frac{S_w(x_i)}{\sum_{x_i \in S_{min}} S_w(x_i)} \quad (9.)$$

Unlike *SMOTE*, S_{min} is first clustered before synthetic samples are generated. The minority sample x_i is extracted according to the sample selection probability $\mathrm{p}(x_i)$, and another minority sample x_j is randomly selected from the cluster to which x_i belongs. The synthetic sample is randomly generated on the connecting line of the two samples. The synthetic sample generation step was repeated until synthetic samples were enough.

After finishing sample synthesis, we use Wilson's Edited Nearest Neighbor Rule (*ENN*) to reduce the noise introduced. Unlike the method proposed by Batista et al., we only applied *ENN* to synthetic samples. For each synthetic sample s_i, find its three nearest neighbors. Delete s_i if there is more than 1 sample of the nearest neighbors belonging to the majority.

Through *IWSE*, minority samples are synthesized to repair class imbalance, and the noise introduced by sample synthesis is reduced.

E. Training and prediction

Through *CFS* and *IWSE*, a training dataset with feature selection and balancing is constructed. We trained the defect prediction classifiers on this training set. Alazba et al.[7] suggested that the optimized tree-based ensemble learning model performs well in software defect prediction tasks. Therefore, we trained three ensemble classifiers, Random Forest (RF), Extra Trees (ET), and AdaBoost (AB). We also trained Logistic Regression (LoR), Stoical Gradient Descent Classifier (SGDC), Decision Tree (DT), and Passive Aggressive Classifier (PA).

IV. EXPERIMENT

To validate our proposed approach, we presented and conducted the following research questions (RQs). We addressed the research problems and validated our proposed strategy by analyzing the experimental observations.

A. Research Questions

RQ1: Does the proposed method improve the performance of software defect prediction?

RQ2: How does the proposed CFS contribute to performance improvement? (Ablation study)

RQ3: How do different oversampling methods affect prediction performance? (Ablation study)

RQ4: How sensitive are the sampling proportion? (Parameter sensitivity)

RQ1 intends to verify the performance of the proposed CFIWSE approach. RQ2 & RQ3 conducted ablation studies to discuss the contribution of the proposed CFS and IWSE approaches. RQ4 explored the sensitivity of sampling proportion.

B. Datasets

For the reproducibility of experiments and the convenience of comparison with the benchmark, we use Yatish's dataset[4] as the ground truth due to the reasons mentioned in Section III.B. The corresponding source code for each version is obtained from GitHub and Apache Repository for metric calculation. We construct a hybrid dependency graph for each version of the source code and calculate SNA metrics as well as code metrics.

The first 80% of each project's dataset is the train-set, while the last 20% of each project's dataset is the test-set. Table 2 show the details of our dataset.

Table 2 Datasets

Project	Versions	Total samples	Imbalance Rate	Train samples	Test samples
ActiveMQ	5	10691	10.20	8553	2138
camel	4	23688	34.76	18951	4737
derby	2	5380	2.72	4304	1076
groovy	3	2152	10.11	1722	430
hive	3	4628	6.68	3703	925
HBase	3	3962	2.79	3170	792
jruby	3	3615	7.79	2892	723
lucene	3	1976	2.98	1581	395
wicket	3	4910	14.46	3928	982

C. Experiment Design

For RQ1, to verify the performance of the proposed *CFIWSE*, it is compared with the following methods: a. no data processing (*RAW*); b. state-of-the-art: *SMOTUNED*[1]; c. state-of-the-art: *SYMPROD*[5]. *RFECV* is used for feature selection with *SMOTUNED* and *SYMPROD*.

For RQ2, to study the contribution of the proposed CFS to the defect prediction effect, it is compared with the following feature selection methods: a. no feature selection (*noFS*); b. common-used wrappers approach: recursive feature elimination with cross-validation (*RFECV*). All feature selection methods were used with the proposed balancing method IWSE to be fair.

For RQ3, in order to study the contribution of the proposed IWSE to the defect prediction effect, it is compared with the following balancing methods: *SMOTE_ENN*[22], *SMOTUNED*[1], *SYMPROD*[5], *Random Undersampler*, *Random Oversampler*, *MSMOTE*[35], *NRAS*[29], *CURE_SMOTE, kmeans_SMOTE*[25], *CCR*[36] and *SMOTE_FRST_2T*[37]. All balancing methods would be used with the proposed feature selection method *CFS* to be fair.

For RQ4, we tend to study the sensitivity of the oversampling proportion.

D. Evaluation

The F1 score and AUC score are commonly utilized in the SDP field[30][38]. The confusion matrix is based on the number of predicted positives and negatives versus the number of actual positives and negatives, as Table 3 shows.

Table 3 Confusion matrix

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

The formulas for calculating precision and recall are listed in the following:

$$Precision = \frac{TP}{TP + FP} \quad (10.)$$

$$Recall = \frac{TP}{TP + FN} \quad (11.)$$

Precision can be an effective metric of model performance when the consequence of a false positive is serious. Recall is concerned with circumstances in which real faulty classes are predicted to be non-defective. F1 is a harmonic mean of accuracy and recall, as defined by (13).

$$\mathcal{F}1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (12.)$$

The Receiver Operating Characteristic (ROC) curve provides a powerful tool for understanding the trade-off between true and false positive rates. In practice, the area under an ROC curve (AUC) is an effective way to summarize the curve into one single value.

V. Results and Analysis

Experiments' results and analysis are presented in this section.

A. RQ1: Does the proposed method improve the performance of software defect prediction?

The efficacy of the proposed *CFIWSE* and compared methods were reported in terms of average AUC and F1 among 7 classifiers over 9 projects, as shown in Table 4&Table 5, in which the best results are bolded.

Table 4 Average AUC for RQ1

	RAW	SMOTUNED	SYMPROD	CFIWSE
activemq	0.566±0.036	0.623±0.043	0.649±0.046	**0.695±0.023**
camel	0.535±0.046	0.588±0.075	0.565±0.058	**0.656±0.063**
derby	0.696±0.08	0.624±0.101	0.722±0.049	**0.729±0.039**
groovy	0.706±0.068	0.77±0.096	0.775±0.081	**0.873±0.027**
hive	0.624±0.052	0.613±0.092	0.661±0.028	**0.722±0.022**
hbase	0.622±0.029	0.604±0.068	0.663±0.04	**0.693±0.023**
jruby	0.665±0.057	0.629±0.086	0.68±0.047	**0.762±0.059**
lucene	0.72±0.059	0.66±0.127	0.715±0.039	**0.734±0.04**
wicket	0.571±0.034	0.688±0.089	0.577±0.017	**0.733±0.037**

Table 5 Average F1 for RQ1

	RAW	SMOTUNED	SYMPROD	CFIWSE
activemq	0.225±0.105	0.351±0.067	0.389±0.053	**0.424±0.019**
camel	0.089±0.09	0.231±0.087	0.171±0.092	**0.319±0.08**
derby	0.572±0.152	0.612±0.092	0.652±0.058	**0.667±0.038**
groovy	**0.449±0.065**	0.285±0.139	0.402±0.107	0.387±0.048
hive	0.347±0.108	0.34±0.093	0.399±0.032	**0.445±0.039**
hbase	0.403±0.051	0.421±0.04	0.473±0.059	**0.492±0.023**
jruby	0.434±0.082	0.307±0.063	0.423±0.057	**0.443±0.086**
lucene	**0.485±0.085**	0.374±0.106	0.453±0.087	0.445±0.064
wicket	0.228±0.087	0.282±0.076	0.242±0.038	**0.39±0.035**

It can be observed that *CFIWSE* outperforms *RAW*, *SMOTUNED* and *SYMPROD* for 7/9 of the datasets. We conducted a t-test on the results to verify the significance of the advantage, and the results showed that the significance of *CFIWSE* was accepted ($p \leq 0.05$) in most cases.

Specifically, for the AB, ET, LoR and SGDC models, *CFIWSE* obtains significantly higher AUC and F1 values than *RAW*, *SMOTUNED* and *SYMPROD* methods. For the PA and

Table 6 Average AUC for RQ2

	CFS	RFECV	noFS
activemq	**0.698±0.027**	0.695±0.021	0.677±0.031
camel	**0.675±0.028**	0.671±0.035	0.661±0.042
derby	**0.729±0.035**	0.704±0.041	0.722±0.062
groovy	**0.872±0.027**	0.839±0.022	0.838±0.023
hive	0.723±0.02	**0.738±0.013**	0.724±0.012
hbase	0.69±0.02	**0.694±0.022**	0.688±0.033
jruby	**0.762±0.056**	0.748±0.063	0.749±0.036
lucene	**0.736±0.037**	0.713±0.087	0.704±0.046
wicket	**0.726±0.034**	0.675±0.081	0.726±0.033

Table 7 Average F1 for RQ2

	CFS	RFECV	noFS
activemq	**0.435±0.016**	0.434±0.026	0.424±0.031
camel	**0.347±0.03**	0.338±0.039	0.305±0.051
derby	**0.664±0.037**	0.619±0.076	0.664±0.05
groovy	**0.384±0.057**	0.358±0.08	0.353±0.078
hive	0.439±0.036	**0.454±0.036**	0.444±0.041
hbase	0.496±0.02	**0.499±0.027**	0.491±0.037
jruby	**0.432±0.078**	0.429±0.063	0.428±0.078
lucene	**0.448±0.059**	0.413±0.104	0.417±0.061
wicket	**0.396±0.038**	0.313±0.093	0.35±0.06

RF classifiers, *CFIWSE* obtains slightly higher AUC values. From the perspective of the project, *CFIWSE* obtains significantly higher AUC and F1 values except for groovy and lucene.

B. RQ2: How does the proposed CFS contribute to performance improvement? (Ablation study)

We conducted an ablation study on software defect prediction to evaluate the contributions of the components of our *CFIWSE* approach. RQ2 aims to explore the contribution of defect prediction performance improvement through *CFS*. For comparison, we replaced the feature selection approach with *noFS* (no feature selection) and the most widely used approach *RFECV* (Recursive Feature Elimination with Cross-Validation), leaving the rest of the components unchanged.

The efficacy of the proposed *CFS* and compared methods were reported in terms of average AUC and F1 among 7 classifiers over 9 projects as shown in Table 6&Table 7, in which the best results are bolded.

It can be observed from Table 6&Table 7 that *CFS* outperforms *noFS* and *RFECV* for most of the datasets, except for hbase and hive. Compared with the most widely used *RFECV*, *CFS* leads AUC by 0.25%-5.16% and F1 by 0.11%-8.23% in seven projects. On hbase and hive, *CFS* is 0.25%-1.55% behind.

We tested the dominance significance and found it less significant than the results in RQ1. This indicates that *CFS* alone is not enough to significantly improve the prediction effect.

C. RQ3: How do different oversampling methods affect prediction performance? (Ablation study)

RQ3 aims to explore the contribution of *IWSE* to the improvement of defect prediction performance. For comparison, we replaced the sampling component of the method with 10 other oversamplers (*SMOTE_ENN*[22], *SMOTUNED*[1], *SYMPROD*[5], *Random Oversampling*, *MSMOTE*[35], *NRAS*[29], *CURE_SMOTE*[39], *kmeans_SMOTE*[25], *CCR*[36], *SMOTE_FRST_2T*[37]) and

Table 8 Average AUC for RQ3

	SMOTE_ENN	SMOTUNED	Random UnderSampler	Random OverSampler
activemq	0.682±0.038	0.623±0.059	0.679±0.029	0.673±0.062
camel	0.652±0.046	0.588±0.069	0.655±0.074	0.613±0.08
derby	0.727±0.052	0.624±0.13	0.699±0.099	0.723±0.06
groovy	0.849±0.035	0.77±0.128	0.794±0.052	0.806±0.062
hive	**0.73±0.015**	0.613±0.107	0.68±0.079	0.689±0.055
hbase	**0.693±0.028**	0.604±0.082	0.679±0.037	0.669±0.046
jruby	0.743±0.107	0.629±0.122	**0.77±0.039**	0.713±0.088
lucene	0.712±0.038	0.66±0.116	0.709±0.05	0.699±0.07
wicket	0.722±0.049	0.688±0.048	0.698±0.075	0.664±0.089
	Borderline SMOTE2	**MSMOTE**	**NRAS**	**CURE SMOTE**
activemq	0.676±0.046	0.677±0.049	0.635±0.05	0.643±0.049
camel	0.617±0.082	0.625±0.085	0.568±0.053	0.591±0.067
derby	0.721±0.066	0.721±0.063	0.69±0.016	0.705±0.075
groovy	0.843±0.054	0.819±0.05	0.765±0.058	0.768±0.049
hive	0.691±0.025	0.704±0.02	0.689±0.04	0.683±0.031
hbase	0.685±0.046	0.665±0.047	0.668±0.039	0.656±0.035
jruby	0.763±0.073	0.736±0.069	0.749±0.051	0.696±0.077
lucene	0.716±0.036	0.719±0.034	0.725±0.028	0.727±0.036
wicket	0.667±0.056	0.659±0.06	0.603±0.046	0.588±0.02
	Kmeans SMOTE	**CCR**	**SMOTE FRST_2T**	**IWSE**
activemq	0.593±0.014	0.643±0.057	0.663±0.052	**0.695±0.023**
camel	0.512±0.013	0.592±0.075	0.624±0.074	**0.656±0.063**
derby	0.696±0.08	0.715±0.067	0.727±0.057	**0.729±0.039**
groovy	0.694±0.079	0.685±0.111	0.811±0.06	**0.873±0.027**
hive	0.593±0.047	0.65±0.057	0.699±0.039	0.722±0.022
hbase	0.621±0.02	0.655±0.051	0.661±0.038	0.693±0.023
jruby	0.668±0.034	0.683±0.083	0.726±0.075	0.762±0.059
lucene	0.651±0.075	0.69±0.081	0.695±0.068	**0.734±0.04**
wicket	0.565±0.022	0.637±0.076	0.685±0.076	**0.733±0.037**

Table 9 Average F1 for RQ3

	SMOTE_ENN	SMOTUNED	Random UnderSampler	Random OverSampler
activemq	0.412±0.025	0.351±0.065	0.379±0.04	0.395±0.057
camel	0.297±0.024	0.231±0.086	0.251±0.061	0.233±0.112
derby	**0.667±0.051**	0.612±0.083	0.652±0.069	0.649±0.078
groovy	0.355±0.056	0.285±0.167	0.269±0.099	0.354±0.095
hive	0.445±0.049	0.34±0.103	0.392±0.08	0.418±0.059
hbase	0.491±0.032	0.421±0.057	0.477±0.058	0.465±0.053
jruby	0.431±0.097	0.307±0.087	0.471±0.075	0.434±0.109
lucene	0.427±0.062	0.374±0.105	0.418±0.068	0.434±0.122
wicket	0.358±0.039	0.282±0.062	0.306±0.078	0.314±0.078
	Borderline SMOTE2	**MSMOTE**	**NRAS**	**CURE_SMOTE**
activemq	0.4±0.028	0.413±0.042	0.378±0.091	0.392±0.071
camel	0.242±0.096	0.238±0.111	0.179±0.107	0.234±0.124
derby	0.642±0.11	0.637±0.108	0.581±0.036	0.588±0.145
groovy	0.424±0.097	0.435±0.057	**0.48±0.107**	0.456±0.063
hive	0.439±0.026	0.437±0.026	0.44±0.035	0.429±0.032
hbase	0.491±0.057	0.467±0.052	0.479±0.056	0.461±0.062
jruby	0.493±0.095	0.496±0.102	**0.541±0.061**	0.477±0.13
lucene	0.443±0.082	0.454±0.077	0.432±0.059	0.447±0.074
wicket	0.335±0.059	0.341±0.048	0.275±0.075	0.266±0.05
	Kmeans SMOTE	**CCR**	**SMOTE_FRST_2T**	**IWSE**
activemq	0.308±0.034	0.372±0.075	0.373±0.037	**0.424±0.019**
camel	0.05±0.051	0.204±0.109	0.244±0.092	**0.319±0.08**
derby	0.563±0.163	0.641±0.082	0.657±0.069	0.667±0.038
groovy	0.447±0.108	0.339±0.198	0.388±0.07	0.387±0.048
hive	0.285±0.111	0.369±0.062	0.419±0.027	**0.449±0.039**
hbase	0.399±0.04	0.453±0.065	0.459±0.042	**0.492±0.023**
jruby	0.449±0.062	0.435±0.132	0.446±0.098	0.443±0.086
lucene	0.386±0.132	0.41±0.156	0.432±0.11	**0.457±0.064**
wicket	0.211±0.05	0.306±0.087	0.328±0.043	**0.39±0.035**

Table 10 Average AUC for each proportion level

	0.1	0.2	0.5	1	2	5
activemq	0.619	0.645	0.674	0.681	**0.692**	0.652
camel	0.560	0.602	0.647	**0.679**	**0.679**	0.645
derby	0.675	0.685	0.709	**0.724**	0.693	0.635
groovy	0.789	0.812	0.816	**0.862**	0.830	0.700
hive	0.648	0.657	0.703	**0.722**	0.700	0.637
hbase	0.639	0.668	0.688	**0.703**	0.659	0.593
jruby	0.720	0.753	**0.763**	0.746	0.750	0.687
lucene	0.707	0.711	**0.736**	0.734	0.726	0.638
wicket	0.594	0.600	0.684	0.701	**0.751**	0.725

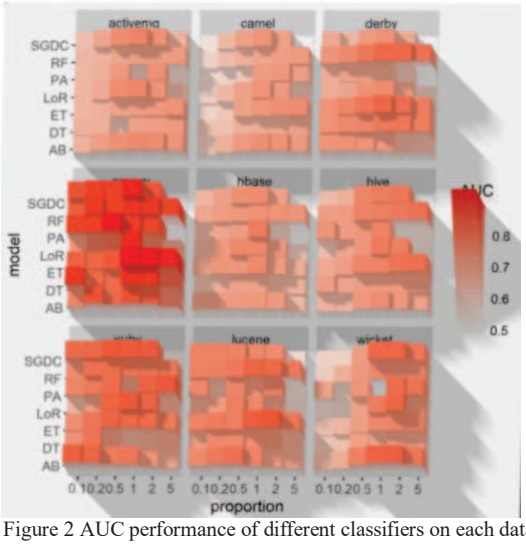

Figure 2 AUC performance of different classifiers on each dataset at different proportion

an under-sampler (*Random Undersampling*), leaving the rest of components unchanged.

The efficacy of the proposed *IWSE* and compared sampling methods were reported in terms of average AUC and F1 among 7 classifiers over 9 projects as shown in Table 8&Table 9, in which the best results are bolded.

It can be observed from Table 8&Table 9 that *IWSE* outperforms other sampling methods for 6/9 of the datasets. The most competitive competitor is *SMOTE_ ENN*, which uses a similar noise reduction method. However, *IWSE* still defeats it in most cases.

D. RQ4: How sensitive are the of the sampling proportion? (Parameters sensitivity)

RQ4 tends to further study how parameters setting influence defect prediction performance. We focus on the parameter-setting issues that are discussed and concerned in the proportion of minority sample synthesis.

Many minority sample synthesis techniques have been proposed in various fields. It should be noted that the number of samples generated by most techniques is freely chosen, and generally defaults to generating minority samples until the number of minority and majority samples is equal. Agrawal et al. suggested that the parameters of sample synthesis had an impact on defect prediction, and conducted experiments for the parameter tuning of *SMOTE*. Being inspired, we explored the proportion of the proposed *IWSE* and carried out experimental verification of 6 levels of proportion.

We set the proportions of six levels: [0.1, 0.2, 0.5, 1, 2, 5]. Average AUC values for each proportion level on seven classifiers are reported in Table 10**Error! Reference source not found.** and highlighted using color depth. The results show that the optimal scale settings are mostly concentrated between 0.5 and 2, with 1 in most cases, which is in line with the recommended settings in many other articles.

Figure 2 details the AUC performance of different classifiers on each dataset at different proportions, showing the following:

a) Optimal proportions for different classifiers vary (but are usually concentrated between 0.5 and 2;

b) Different datasets have different sensitivities to proportion, and the average difference between the best and worst performance of AUC is 7%-16%.

c) The worst performances are usually proportion=0.1 or proportion=5.

When the proportions are too low, there is too little sample synthesis to improve the quality of the dataset. When the proportions are too high, the number of synthetic samples are far more than the real samples, which not only causes class imbalance in the other direction, but also inevitably causes serious overfitting and noise introduction problems. We recommend *that* users tune proportions within 0.5 to 2 when applying IWSE to different datasets, or use the default proportion=1 if there are no tuning resources.

VI. THREATS TO VALIDITY

A. Construct Validity.

Threats to construct validity relate to dataset selection. We used Yatish et al.[4] dataset as the ground truth when conducting our experiments. We built hybrid dependency graphs from the source code and calculated the SNA metrics and code metrics instead of directly using the statistical metrics provided by Yatish et al. Although Yatish et al.[4] sought to eliminate some of the associated noise with the aforementioned inconsistencies, some amount of noise in the dataset cannot be avoided.

Another threat to construct validity is that we quantify the correlation of metrics by the Pearson Correlation Coefficient. Spearman's and Kendall's Correlation Coefficients and Mutual Information Coefficient (MIC) is also commonly used to measure the correlation. As such, it presents a threat to the *CFS* that we proposed in Section III.C. However, Wang et al.[31] compared the performances of Pearson's, Spearman's, Kendall's correlation coefficients and MIC, concluding that Pearson's is the best. Nevertheless, we encourage future studies to revisit the performance of our studies with different correlation coefficients.

B. External Validity.

In nine fixed open source software projects developed in JAVA, we analyzed the validity of SNA measurement compared with code metrics. Although the research projects are varied, our findings may not be extended to projects with different module sizes and versions.

Moreover, since the source code of the state-of-the-art *SMOTUNED* was not opened, we reimplemented the

SMOTUNED based on the pseudo-code Agrawal et al.[1] provided. Since Agrawal et al. [1] did not disclose the absolute value of the experimental results of *SMOTUNED* (provided delta values instead), it is difficult to evaluate whether there is a gap between our reimplementation and the original version.

C. Internal validity.

Threats to the internal validity relate to hyperparameter settings when fine-tuning our *CFIWSE* approach. For the nearest neighbor setting in *CFS*, we used the value suggested by Wang et al[31]. For the nearest neighbor setting in *IWSE*, we used the most common value of 5. The user-defined parameters $C_f^{therhold}$ and C_{MAX} are set to default values of 5 and 10.

We only study the oversampling proportion and within project scenario. Therefore, we encourage future studies to explore the impact of the hyperparameters on the usefulness of *CFIWSE* across different SDP contexts and scenarios.

VII. Conclusion

As far as SDP, feature selection and data balance are key contemporary research issues. In this paper, a hybrid preprocessing approach for defect prediction named *CFIWSE* is proposed to improve SDP performance through feature selection, minority sample synthesis and noise reduction, consisting of *CFS* and *IWSE*. *CFS* uses feature correlation and nearest neighbor theory to remove irrelevant and redundant features. *IWSE* is applied to synthesize a few samples and clean up the introduced noise samples to solve the problem of data imbalance. The proposed method is tested on SNA metric and code metric data calculated in real-world software. Several evaluation metrics were deployed to capture the performance of the methods tested on seven classifiers. The experimental results show that our method is better than the most advanced *SMOTUNED* and *SYMPROD* methods.

Additionally, the contribution of method components has been ablated and the results show that the performance improvement achieved by combining *CFS* with *IWSE* is significantly higher than that achieved by using alone. Sensitivity studies on sampling proportions have also been carried out and the conclusions are as follows: proportions are recommended to be set between 0.5 and 2; B) if there were no prior knowledge, proportion is recommended to be set as 1.

For future works, we plan to further study parameter-setting in *CFIWSE* to improve SDP performance.

Acknowledgment

The authors would like to thank all reviewers for their questions and constructive suggestions.

References

[1] A. Agrawal and T. Menzies, "Is 'better data' better than 'better data miners'?: On the benefits of tuning SMOTE for defect prediction," *Proc. - Int. Conf. Softw. Eng.*, pp. 1050–1061, 2018, doi: 10.1145/3180155.3180197.

[2] L. Gong, G. K. K. Rajbahadur, A. E. Hassan, and S. Jiang, "Revisiting the Impact of Dependency Network Metrics on Software Defect Prediction," *IEEE Trans. Softw. Eng.*, pp. 1–19, 2021, doi: 10.1109/TSE.2021.3131950.

[3] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 2, pp. 111–147, 2019, doi: 10.1109/TSE.2017.2770124.

[4] S. Yatish, J. Jiarpakdee, P. Thongtanunam, and C. Tantithamthavorn, "Mining Software Defects: Should We Consider Affected Releases?," *Proc. - Int. Conf. Softw. Eng.*, vol. 2019-May, pp. 654–665, 2019, doi: 10.1109/ICSE.2019.00075.

[5] I. Kunakorntum, W. Hinthong, and P. Phunchongharn, "A Synthetic Minority Based on Probabilistic Distribution (SyMProD) Oversampling for Imbalanced Datasets," *IEEE Access*, vol. 8, pp. 114692–114704, 2020, doi: 10.1109/ACCESS.2020.3003346.

[6] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 356–370, 2011, doi: 10.1109/TSE.2010.90.

[7] A. Alazba and H. Aljamaan, "Software Defect Prediction Using Stacking Generalization of Optimized Tree-Based Ensembles," *Appl. Sci.*, vol. 12, no. 9, 2022, doi: 10.3390/app12094577.

[8] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA, NY, USA: Elsevier Science Inc., 1977.

[9] S. Chidamber and C. F. Kemerer, "A Metric suite for object oriented design," *IEEE Trans. Softw. Eng.*, 1994.

[10] F. B. e Abreu and R. Carapuça, "Candidate metrics for object-oriented software within a taxonomy framework," *J. Syst. Softw.*, vol. 26, no. 1, pp. 87–96, 1994, doi: 10.1016/0164-1212(94)90099-X.

[11] N. Ohlsson and H. Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches," *IEEE Trans. Softw. Eng.*, 1996, doi: 10.1109/32.553637.

[12] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, 2007, doi: 10.1109/TSE.2007.256941.

[13] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings - International Conference on Software Engineering*, 2008, pp. 531–540. doi: 10.1145/1368088.1368161.

[14] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, 2008, doi: 10.1016/j.jss.2007.07.040.

[15] H. Hata, T. Kikuno, and O. Mizuno, "A systematic review of software fault prediction studies and related techniques," *Comput. Softw.*, vol. 29, no. 1, pp. 106–117, 2012, doi: 10.11309/jssst.29.1_106.

[16] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," in *IEEE Transactions on Software Engineering*, 2008, vol. 34, no. 4, pp. 485–496. doi: 10.1109/TSE.2008.35.

[17] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings - International Conference on Software Engineering*, 2015, vol. 1, pp. 789–800. doi: 10.1109/ICSE.2015.91.

[18] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 2009, pp. 109–119. doi: 10.1109/ISSRE.2009.17.

[19] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, 2013, doi: 10.1109/TSE.2013.11.

[20] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Reflections on the NASA MDP data sets," *IET Softw.*, vol. 6, no. 6, pp. 549–558, 2012, doi: 10.1049/iet-sen.2011.0132.

[21] W. P. Chawla, N. V and Bowyer, K. W and Hall, L. O and Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, vol. 16, no. 1, pp. 321–357, 2002, doi: 10.1613/jair.953.

[22] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explor. Newsl.*, vol. 6, no. 1, p. 20, 2004, doi: 10.1145/1007730.1007735.

[23] H. Han, W. Y. Wang, and B. H. Mao, "Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning," *Lect. Notes*

Comput. Sci., vol. 3644, no. PART I, pp. 878–887, 2005, doi: 10.1007/11538059_91.

[24] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proceedings of the International Joint Conference on Neural Networks*, 2008, no. 3, pp. 1322–1328. doi: 10.1109/IJCNN.2008.4633969.

[25] G. Douzas, F. Bacao, and F. Last, "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE," *Inf. Sci. (Ny).*, vol. 465, pp. 1–20, 2018, doi: 10.1016/j.ins.2018.06.056.

[26] G. Douzas and F. Bacao, "Self-Organizing Map Oversampling (SOMO) for imbalanced data set learning," *Expert Syst. Appl.*, vol. 82, no. Japkowicz 2000, pp. 40–52, 2017, doi: 10.1016/j.eswa.2017.03.073.

[27] H. Lee, J. Kim, and S. Kim, "Gaussian-based SMOTE algorithm for solving skewed class distributions," *Int. J. Fuzzy Log. Intell. Syst.*, vol. 17, no. 4, pp. 229–234, 2017, doi: 10.5391/IJFIS.2017.17.4.229.

[28] S. Barua, M. M. Islam, X. Yao, and K. Murase, "MWMOTE - Majority weighted minority oversampling technique for imbalanced data set learning," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 2, pp. 405–425, 2014, doi: 10.1109/TKDE.2012.232.

[29] W. A. Rivera, "Noise Reduction A Priori Synthetic Over-Sampling for class imbalanced data sets," *Inf. Sci. (Ny).*, vol. 408, pp. 146–161, 2017, doi: 10.1016/j.ins.2017.04.046.

[30] Y. Yang, J. Ai, and F. Wang, "Defect Prediction Based on the Characteristics of Multilayer Structure of Software Network," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2018, pp. 27–34. doi: 10.1109/QRS-C.2018.00019.

[31] F. Wang, J. Ai, and Z. Zou, "A Cluster-Based Hybrid Feature Selection Method for Defect Prediction," in *Proceedings - 19th IEEE International Conference on Software Quality, Reliability and Security, QRS 2019*, 2019, pp. 1–9. doi: 10.1109/QRS.2019.00014.

[32] M. Aniche, "Java code metrics calculator (CK)." 2015.

[33] G. K. Armah, G. Luo, K. Qin, and A. S. Mbandu, "Applying Variant Variable Regularized Logistic Regression for Modeling Software Defect Predictor," *Lect. Notes Softw. Eng.*, vol. 4, no. 2, pp. 107–115, 2016.

[34] D. R. Cox, "The regression analysis of binary sequences," *J. R. Stat. Soc. Ser. B*, vol. 20, no. 2, pp. 215–232, 1958.

[35] S. Hu, Y. Liang, L. Ma, and Y. He, "MSMOTE: Improving classification performance when training data is imbalanced," *2nd Int. Work. Comput. Sci. Eng. WCSE 2009*, vol. 2, pp. 13–17, 2009, doi: 10.1109/WCSE.2009.756.

[36] M. Koziarski and M. Wozniak, "CCR: A combined cleaning and resampling algorithm for imbalanced data classification," *Int. J. Appl. Math. Comput. Sci.*, vol. 27, no. 4, pp. 727–736, 2017, doi: 10.1515/amcs-2017-0050.

[37] E. Ramentol *et al.*, "Fuzzy-rough imbalanced learning for the diagnosis of High Voltage Circuit Breaker maintenance: The SMOTE-FRST-2T algorithm," *Eng. Appl. Artif. Intell.*, vol. 48, pp. 134–139, 2016, doi: 10.1016/j.engappai.2015.10.009.

[38] W. Fu and T. Menzies, "Revisiting unsupervised learning for defect prediction." pp. 72–83, 2017. doi: 10.1145/3106237.3106257.

[39] L. Ma and S. Fan, "CURE-SMOTE algorithm and hybrid algorithm for feature selection and parameter optimization based on random forests," *BMC Bioinformatics*, vol. 18, no. 1, pp. 1–18, 2017, doi: 10.1186/s12859-017-1578-z.