Kỷ yếu Hội nghị Khoa học công nghệ Quốc gia lần thứ XVI về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR); Đà Nẵng, ngày 28-29/9/2023
DOI: 10.15625/vap.2023.0046

A COMBINATION OF FEATURE SELECTION AND DATA SAMPLING TECHNIQUES FOR SOFTWARE FAULT PREDICTION

Ha Thi Minh Phuong[1], Nguyen Thanh Long[2], Nguyen Thanh Binh[3]

[1,3]The University of Danang - Vietnam-Korea University of Information and Communication Technology
[2]The University of Danang - University of Science and Technology

htmphuong@vku.udn.vn, ngthlo.doc@gmail.com, ntbinh@vku.udn.vn

ABSTRACT: *Software fault prediction (SFP) is the process of building models to predict faults in the early stage of software development. Prediction of software fault-prone modules can help developers allocate testing efforts more effectively and optimize maintenance cost. However, the performance of SFP models is influenced by the quality of software fault datasets. The irrelevant and redundant features of datasets may lead to negative impacts on the speed and accuracy of the trained models. Additionally, the presence of data imbalance that the number of faulty modules is significantly less than the number of non-faulty modules is the challenge in SFP. The study has applied 3 Generative adversarial networks (GAN) models including VanillaGAN, CTGAN and WGANGP along with 4 feature selection ranking methods including Chi-Squared, Information Gain, Fisher and Relief on four software fault datasets. The comparative analysis is performed by using 4 different classifiers to predict software faults. We have considered precision, recall, F1-score and Area Under the ROC (receiver operating characteristic curve) Curve (AUC) as performance evaluation metrics. The experimental results reveal that combinations of CTGAN, VanillaGAN and feature selection approaches outperformed the SFP models without applying data sampling and feature selection methods. The combinational pair of CTGAN and Relief demonstrated the best performance than other combinations with the highest average precision, recall, F1-score and AUC values of 0.857, 0.873, 0.856 and 0.767, respectively on Extra Tree.*

Keywords: *Software fault prediction, Feature selection, Data sampling, Promise.*

I. INTRODUCTION

Recently, software systems are growing in both complexity and size, thereby ensuring their reliability and quality becomes increasingly critical. According to Arar *et al.* [1], software testing and other software quality activities account for about 23% of the total software budget. Software faults need to be eliminated to improve software quality and reduce cost maintenance. Therefore, it would be beneficial to identify potential faults in the early stage of software development. As a result, software fault prediction techniques are proposed to predict the occurrence of faults and assist software developers allocate their testing resources effectively [2]. Software fault prediction [3, 4] is the process of building classification models to predict whether code regions contain faults or not using historical fault data. The typical fault prediction consists of two phases [5]: extracting code features from source files and developing classifiers using machine learning and deep learning models for training and testing. Several previous studies in fault prediction have used manually designed software metrics including Halstead metrics [6], McCabe metrics [7], CK metrics [8], etc. For the second phase, machine learning models such as Random Forest [9], Logistic Regression [10], Naive Bayes [11] were applied. However, the performance of software fault prediction models depends on various factors such as fault datasets, software metrics, machine learning techniques and dataset issues [12]. As shown from several studies [13-15], most historic fault datasets have issues including redundant, irrelevant features and imbalanced classes that lead to negative effects on predictive performance. Generally, in the software fault dataset, the number of non-faulty modules (the majority classes) outnumbers the number of faulty modules (the monitory classes), resulting in imbalanced classes [15]. This usually causes the machine learning models to give inaccurate results. Consequently, the problem of data imbalance is the most significant challenge impacting on the SFP's models and a growing number of approaches have been proposed to overcome it. Additionally, irrelevant and redundant features also affect the speed and accuracy of the trained classifiers [16]. Feature selection technique is required for selecting the best subsets of software metrics to achieve good prediction results [17]. Hence, in this paper, we applied four feature rankings to select the optimal software metrics/features that are effective to train the SFP models. In addition, another object of our study is to handle the class imbalance problem which can improve the predictive performance. In order to overcome both the class imbalance and irrelevant/redundant feature elimination, we conduct the implementation of a combination of three oversampling methods and four filter-based feature selection techniques to identify the best-performing combination for SFP models. We have considered precision, recall, Area Under the ROC Curve (AUC) and F1-score performance metrics to compare the performance of various combinations of feature selection and oversampling techniques. This paper provides the following contributions:

- The experiment was carried out to examine the combined approach of 20 different combinations of filter-based feature selection and oversampling techniques (4 feature selection techniques + 1 full metrics) $\times$ (3 oversampling techniques + 1 original dataset) on 4 fault datasets extracted from the PROMISE repository [18] with the employment of different classification algorithms such as Random Forest (RF), Extra Tree (ET), AdaBoost (AdaBoost), Histogram-based Gradient Boosting (HGB) to draw a conclusion of the best combinations.

- The performance of SFP models was compared with applying feature selection and sampling techniques to examine the significant effects of these techniques in handling irrelevant/redundant features and class imbalance.

The paper is structured as follows. Section II introduces related work in software fault prediction field. Section III provides the details of the experimental design and analysis. The experimental results are presented and discussed in Section IV. Finally, Section V presents the conclusion of our study.

II. RELATED WORK

In the field of software fault prediction, several studies have been done on investigating feature selection and sampling methods independently [19]. Nevendra *et al*. [20] proposed a new approach called AdaBoost.ET which developed boosting and bagging-based learners to improve the prediction rate. Mangla *et al*. [21] presented a sequential model to predict software faults on the 8 datasets extracted from the PROMISE and ECLIPSE repositories. The proposed approaches have built software fault prediction models based on software metrics. Jureczko [22] conducted a review of the effectiveness of various software metrics and evaluated the correlation between the software faults and metrics. They analyzed several CK metrics such as Depth of Inheritance Tree (DIT), Coupling between object classes (CBO), Weighted Methods per Class (WMC), Number of Children (NOC), etc. However, feature redundancy has affected on the speed and accuracy of SFP models, hence, several literature reviews showed feature selection is an important stage that leads to better performance of classifiers [23]. The aim of feature selection is to select the optimal set of metrics for optimizing the performance of SFP models. Thus, in this paper, we have applied four filter-based feature selection techniques over 4 datasets to generate the best software metrics that are useful for trained models. Additionally, the input datasets are usually imbalanced with the number of non-faulty classes (majority classes) significantly greater than the number of faulty classes (monitory classes). In this case, machine learning models may not learn sufficient data for faulty classes to produce reliable results. However, there are few studies that have been conducted to address both feature redundancy and class imbalance problems [12, 24]. Based on the above discussion, it is observed that there is a need for in-depth research to evaluate the performance of SFP models by combining sampling and feature selection techniques. Therefore, for the motivation of this study, we exploited the implication of 3 sampling strategies (VanillaGAN, CTGAN, WGANGP) and 4 feature selection techniques (Relief, Info gain, Fisher and Chi-squared) to tackle the class imbalance and feature redundancy problem. The performance of the developed SFP model is analyzed using precision, recall and F1-score evaluation metrics developed with the implication of sampling and feature selection techniques.

III. METHODOLOGY

In this paper, our objective is to determine the best sets of software metrics and handle imbalanced classes for building effective SFP models with high prediction performance. The steps of the proposed experimental methodology are shown in *Figure 1*. Overview of the proposed methodologyFirstly, we collected four fault datasets, namely CM1, KC1, KC2 and PC1 from the PROMISE repository. We applied data normalization using the z-normalization technique for pre-preprocessing stage. The normalized datasets are split into training and test sets. After applying three GAN oversampling models including VanillaGAN, CTGAN and WGANGP to balance training datasets, we utilized four filter-based feature selections, namely Chi-Squared, Information Gain, Fisher and Relief to extract the top log_2N features (N - the total number of software metrics in the full fault datasets). The balanced training datasets with optimal features were trained on Random Forest (RF), Extra Tree (ET), AdaBoost (AB) and HistGradientBoosting (HG). The test sets are then fed to SFP models for comparing to the performance of these models using combinations of feature selection and data sampling techniques in terms of precision, recall, F1-score and AUC. The following sections indicate in detail software fault dataset, filter-based feature selection, GAN oversampling techniques and performance evaluation measures.

A. Datasets

In this study, we have used 4 different datasets extracted from the PROMISE repository which are widely used in many studies [19, 20]. In terms of software fault prediction, each dataset contains the independent variables which are source code metrics such as Line of Code (LOC), Depth of Inheritance Tree (DIT), etc... and the dependent variable is faulty or non-faulty module. The detailed datasets are shown in *Table 1*. Description of the used software fault datasets.

Table 1. Description of the used software fault datasets

Dataset	Release	Instances	Metrics	Faulty Instances	Imbalanced Ratio (%)
PROMISE	CM1	498	21	48	9.7
	KC1	2107	21	325	15.4
	KC2	522	21	107	20.5
	PC1	1107	21	76	6.9

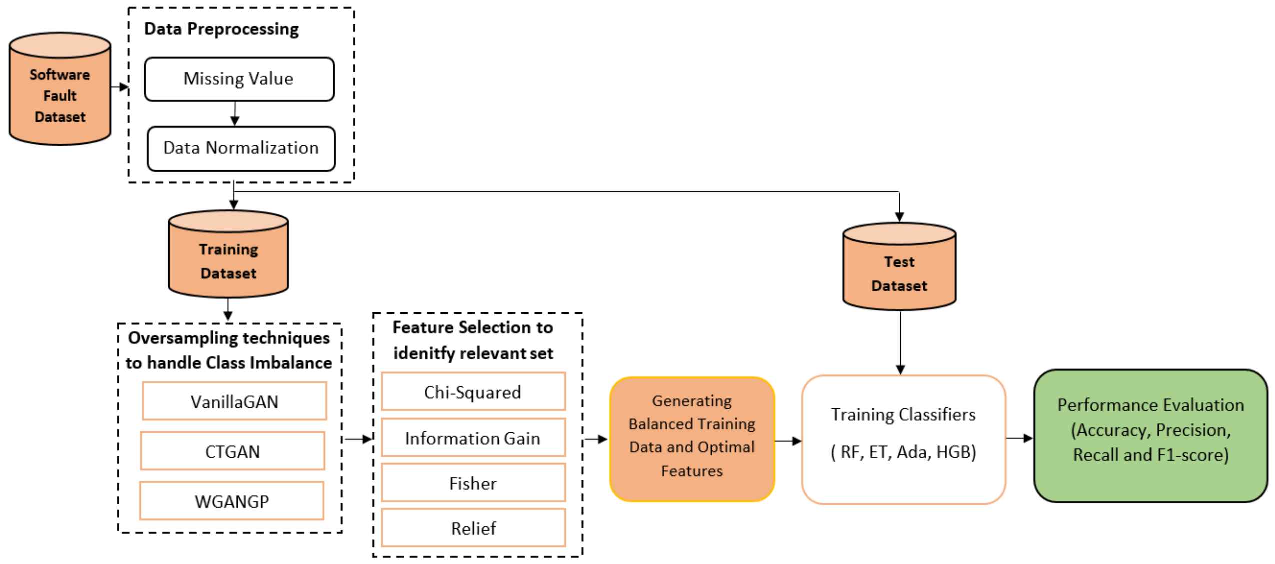
Software Fault Dataset
Data Preprocessing
Missing Value
Data Normalization
Training Dataset
Test Dataset
Oversampling techniques to handle Class Imbalance
VanillaGAN
CTGAN
WGANGP
Feature Selection to idenitfy relevant set
Chi-Squared
Information Gain
Fisher
Relief
Generating Balanced Training Data and Optimal Features
Training Classifiers
(RF, ET, Ada, HGB)
Performance Evaluation (Accuracy, Precision, Recall and F1-score)

Figure 1. Overview of the proposed methodology

B. Filter-based feature selection

The filter methods use ranking techniques to score the features and a threshold is utilized to eliminate features below the threshold. The ranking techniques are applied before the classification stage to filter out the redundant/irrelevant features [17]. The feature selection ranking methods used in this study are Chi-Squared, Information Gain, Fisher and Relief which select optimal software metrics. While Chi-Squared is used to rank features according to significance, Information Gain is an entropy-based which is calculated by comparing the entropy of the dataset before and after splitting on features. The aim of Relief is to distinguish between instances of different classes by estimating the importance or relevance of each feature. The Fisher score is derived from Fisher's criterion, which seeks to maximize the distance between the class means while minimizing the variance within each class [27].

C. Data oversampling

VanilaGAN is the simplest version of GAN. The generator network leans the probability of the training set by mapping the noise *z* which is drawn independent and identically distributed from *N* (0, 0.01) and added to both real and synthetic data to the probability distribution of training samples. Then, synthetic samples that are closer to real samples were generated by the generator network. It uses backpropagation to train both models. In VanillaGAN, we use neural networks to approximate complex, high-dimensional distributions for both generator and discriminator. The discriminator training is done by minimizing its prediction error, whereas the generator is trained by maximizing the prediction error by the discriminator. This can be formalized as follows:

$$min_{(\ominus Gen)} max_{(\ominus Dis)} \left(E_{x\sim p_D}[logDis(x)] + E_{z\sim p_z}\left[\log\left(1 - Dis\left(Gen(z)\right)\right)\right]\right) \tag{1}$$

where p_D is the real data distribution, p_z is the prior distribution of the generative network and $\ominus\, Gen$ and $\ominus\, Dis$ are the parameters of the generator (Gen) and discriminator (Dis). Given a strong discriminator, the generator's goal is achieved if p_D, the generator's distribution over x, equals p_z, the real distribution, which means that the Jensen-Shannon Divergence (JSD) is minimized.

WGANGP is short for Wasserstein GAN with Gradient Penalty, a variant of GAN. WGAN (Wasserstein GAN) helps to tackle issue of VanillaGAN that guarantees the model will converge at equilibrium. However, WGAN faces a vanishing gradient problem. Therefore, WGANGP was proposed to solve gradient vanishing, or exploding problems that were there in WGAN by using a gradient penalty instead of the weight-clipping. The GAN objective is defined as follows:

$$min_{(\ominus Gen)} max_{(\ominus Dis)} \left(E_{x\sim p_D}[logDis(x)] - E_{z\sim p_z}[Dis(Gen(z))]\right) - \lambda\, E_{\hat{x}\sim p_x}\left[(||\nabla_{\hat{x}}(\hat{x})| \quad |_2 - 1)^2\right] \tag{2}$$

where λ is the penalty coefficient. WGANGP uses a two-sided penalty based on empirical evidence that penalizes gradients with a norm less than 1. The gradient penalty serves as a regularizer, ensuring that a discriminator that has been trained to perfection has a smooth linear gradient that guides the generator to the data distribution while restricting the discriminator's power.

CTGAN is a GAN-based data synthesizer for single table data, which is able to learn from real data and generates synthetic data with high fidelity. Unlike the above two variations of GAN, CTGAN uses mode-specific normalization instead of min-max normalization for continuous columns. The mode-specific normalization technique is leveraged to deal with columns that contain non-Gaussian and multimodal distributions. For each continuous column, CTGAN uses a variational Gaussian mixture model (VGM) to predict the number of modes and fit a Gaussian mixture (GM).

D. Performance evaluation measures

In order to evaluate the performance feature selection ranking and data oversampling techniques, we considered evaluation metrics such as precision, recall, F1-score and AUC. *Table 2.* Confusion matrix presents a confusion matrix for two-class classifiers.

Table 2. Confusion matrix

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

- Precision: the ratio of positive instances from the total predicted positive instances.
- Recall: known as sensitive, it presents the ratio of positive instances and the total actual positive instances.
- F1-score: the harmonic mean of recall and precision.
- AUC: is a summary of the ROC (the Receiver Operating Curve) that evaluates the ability of a classifier to distinguish between classes.

IV. EXPERIMENTAL RESULTS

This section illustrates the experimental results on four fault datasets to evaluate the performance of combined techniques (filter-based feature selection and oversampling). As mentioned previously, 3 different oversampling methods are employed to balance 4 datasets with software metrics and 4 filter-based feature selection techniques have been applied to overcome the redundancy problem. The performance of SFP models with oversampling and feature selection techniques based on the selected and full software metrics is shown in Tables 4-7. The average performance of four classifiers (RF, ET, AB and HGB) was also computed in each table to examine the effectiveness of feature selection and data sampling in SFP in terms of precision, recall, F1-score and AUC. As shown in Tables 4-7, it is observed that the performance of the predictive models based on a combination of 4 filter-based feature selection and 2 oversampling methods (VanillaGAN and CTGAN) was better than when no feature selection and data sampling are utilized (as shown in *Table 3*). For instance, the average AUC values of the SFP model on the original datasets are 0.690, 0.711, 0.719 and 0.733 with the RF, ET, AB, and HGB model, respectively.

Table 3. Results of the SFP models without data sampling and feature selection

Dataset	Performance Measures	**RF**	**ET**	**AB**	**HGB**
CM1	Precision	0.511	0.580	0.589	0.577
	Recall	0.502	0.556	0.561	0.542
	F1-score	0.502	0.563	0.567	0.546
	AUC	0.634	0.623	0.703	0.718
KC1	Precision	0.719	0.707	0.689	0.714
	Recall	0.579	0.623	0.609	0.640
	F1-score	0.596	0.645	0.630	0.663
	AUC	0.702	0.710	0.723	0.728
KC2	Precision	0.726	0.694	0.702	0.725
	Recall	0.665	0.646	0.671	0.678
	F1-score	0.684	0.660	0.683	0.695
	AUC	0.701	0.724	0.707	0.721
PC1	Precision	0.763	0.730	0.653	0.714
	Recall	0.594	0.593	0.616	0.620
	F1-score	0.629	0.624	0.629	0.649
	AUC	0.726	0.788	0.746	0.766
	Average				
	Precision	0.679	0.677	0.658	0.682
	Recall	0.585	0.605	0.614	0.620
	F1-score	0.603	0.623	0.627	0.638
	AUC	0.690	0.711	0.719	0.733

When comparing the average values of the combinational pairs of filter-based feature selection and sampling techniques in *Table 4*, a combination of Chi-Squared and CTGAN exhibited the greatest value for all performance measures, followed by pair of Chi-Squared and VanillaGAN. In particular, the average results of these pairs gained the highest results for precision, recall, F1-score and AUC with 0.848, 0.867, 0.851 and 0.771, respectively. As shown in *Table 5*, it can be seen that the combination of Information Gain and CTGAN performed the best precision, recall and F1-score on the PC1, with the highest values of 0.924, 0.932 and 0.925. Again, the pair of Information Gain and CTGAN on Extra Tree also obtained the best average results for precision, recall and F1-score values of 0.849, 0.867 and 0.852. As the results of *Table 6,* it is evident that the combination of Fisher and CTGAN outperformed other models for all performance measures on most datasets. For instance, it achieved the highest values of precision, recall and F1-score with 0.925, 0.935 and 0.925, respectively. As depicted in Table 7. Results of the Relief&GAN methods, we observed that Relief and CTGAN achieved a better performance than other combinations of filter-based feature selection and oversampling techniques with the highest average precision, recall, F1-score and AUC values of 0.857, 0.873, 0.856 and 0.767, respectively on Extra Tree.

Table 4. Results of the Chi-Squared & GAN methods

Dataset	Performance Evaluation	Chi-Squared											
		VanillaGAN				CTGAN				WGANGP			
		RF	ET	AB	HGB	RF	ET	AB	HGB	RF	ET	AB	HGB
CM1	Precision	0.826	**0.856**	0.843	0.849	0.843	**0.856**	0.854	0.837	0.834	0.819	0.828	0.821
	Recall	0.712	0.824	0.780	0.810	0.860	**0.878**	0.848	0.846	0.804	0.832	0.726	0.816
	F1-score	0.758	0.834	0.806	0.827	0.849	**0.861**	0.849	0.840	0.816	0.824	0.766	0.817
	AUC	0.635	**0.739**	0.679	0.706	0.675	**0.739**	0.723	0.690	0.643	0.657	0.574	0.679
KC1	Precision	0.809	0.816	0.807	0.823	0.817	**0.827**	0.813	0.825	0.804	0.804	0.797	0.808
	Recall	0.759	0.826	0.802	0.831	0.845	**0.848**	0.841	0.850	0.814	0.808	0.815	0.802
	F1-score	0.778	0.820	0.804	0.826	0.820	**0.830**	0.817	0.830	0.803	0.802	0.801	0.799
	AUC	0.752	0.794	0.753	0.780	0.793	**0.793**	0.745	0.781	0.755	0.734	0.647	0.734
KC2	Precision	0.815	0.785	0.795	0.788	**0.798**	0.785	0.786	0.787	0.809	0.813	0.807	0.825
	Recall	0.809	0.798	0.798	0.796	**0.815**	0.806	0.8	0.806	0.638	0.672	0.657	0.668
	F1-score	0.811	0.788	0.794	0.788	**0.792**	0.788	0.788	0.791	0.665	0.698	0.682	0.693
	AUC	0.777	0.783	0.701	0.773	**0.785**	0.775	0.658	0.762	0.753	0.799	0.744	0.761
PC1	Precision	0.899	0.919	0.919	0.916	0.877	**0.925**	0.913	0.930	0.884	0.911	0.887	0.891
	Recall	0.824	0.916	0.907	0.916	0.922	**0.935**	0.921	0.938	0.817	0.814	0.682	0.823
	F1-score	0.854	0.917	0.911	0.915	0.897	**0.925**	0.916	0.929	0.837	0.836	0.739	0.844
	AUC	0.780	0.784	0.839	0.826	0.724	0.775	0.838	0.802	0.681	0.698	0.616	0.663
						Average							
	Precision	0.837	0.844	0.841	0.844	0.833	**0.848**	0.841	0.845	0.833	0.837	0.830	0.836
	Recall	0.776	0.841	0.822	0.838	0.860	**0.867**	0.852	0.860	0.768	0.781	0.720	0.777
	F1-score	0.800	0.840	0.829	0.839	0.839	**0.851**	0.842	0.847	0.780	0.790	0.747	0.788
	AUC	0.736	0.775	0.743	0.771	0.744	**0.771**	0.741	0.759	0.708	0.722	0.645	0.709

Table 5. Results of the Information Gain&GAN methods

Dataset	Performance Evaluation	Information Gain											
		VanillaGAN				CTGAN				WGANGP			
		RF	ET	AB	HGB	RF	ET	AB	HGB	RF	ET	AB	HGB
CM1	Precision	0.853	0.853	0.843	0.844	0.832	**0.863**	0.864	0.832	0.834	0.823	0.827	0.820
	Recall	0.722	0.840	0.790	0.812	0.870	**0.888**	0.866	0.850	0.858	0.852	0.782	0.790
	F1-score	0.769	0.843	0.812	0.826	0.847	**0.865**	0.861	0.839	0.844	0.836	0.801	0.802
	AUC	0.671	0.695	0.644	0.658	0.671	**0.695**	0.662	0.627	0.633	0.63	0.621	0.638
KC1	Precision	0.797	0.808	**0.811**	0.808	0.815	0.807	0.810	0.809	0.814	0.795	0.804	0.802
	Recall	0.836	0.830	**0.841**	0.832	0.845	0.832	0.840	0.836	0.827	0.796	0.807	0.789
	F1-score	0.801	0.815	**0.818**	0.816	0.822	0.815	0.814	0.817	0.817	0.792	0.804	0.793
	AUC	0.765	0.784	0.735	0.761	0.773	0.778	0.737	0.754	0.763	0.725	0.664	0.725
KC2	Precision	0.803	0.807	0.791	0.811	**0.821**	0.803	0.776	0.794	0.798	0.804	0.813	0.812
	Recall	0.804	0.815	0.791	0.813	**0.825**	0.815	0.787	0.808	0.64	0.643	0.628	0.634
	F1-score	0.800	0.807	0.789	0.807	**0.807**	0.801	0.778	0.796	0.661	0.668	0.655	0.657
	AUC	0.777	0.791	0.726	0.792	**0.812**	0.788	0.651	0.783	0.765	0.788	0.737	0.767
PC1	Precision	0.893	0.918	0.899	0.907	0.881	**0.924**	0.909	0.911	0.879	0.890	0.878	0.889
	Recall	0.892	0.915	0.907	0.917	0.919	**0.932**	0.922	0.922	0.799	0.877	0.818	0.858
	F1-score	0.891	0.915	0.902	0.911	0.897	**0.925**	0.913	0.914	0.829	0.876	0.838	0.866
	AUC	0.772	0.755	0.757	0.797	0.689	0.747	0.781	0.780	0.649	0.663	0.559	0.698
						Average							
	Precision	0.836	0.846	0.836	0.843	0.837	**0.849**	0.840	0.836	0.831	0.828	0.830	0.830
	Recall	0.813	0.850	0.832	0.844	0.864	**0.867**	0.854	0.854	0.781	0.792	0.759	0.768
	F1-score	0.816	0.845	0.830	0.840	0.843	**0.852**	0.842	0.842	0.788	0.793	0.775	0.780
	AUC	0.746	0.753	0.716	0.752	0.736	**0.753**	0.708	0.736	0.702	0.702	0.645	0.707

However, the combinations of each feature selection technique (Chi-Squared, Information Gain, Fisher, Relief) and WGAN produced the same performance when no feature selection and data sampling techniques were applied. When comparing the performance of the WGANP method with the AdaBoost technique, we found that the average AUC values of Chi-Squared, Information Gain, Fisher and Relief had the lowest AUC values with 0.645, 0.645, 0.666 and 0.691 respectively as shown in Tables 4-7.

Table 6. Results of the Fisher&GAN methods

Dataset	**Performance Evaluation**	**Fisher**											
		VanillaGAN				**CTGAN**				**WGANGP**			
		RF	**ET**	**AB**	**HGB**	**RF**	**ET**	**AB**	**HGB**	**RF**	**ET**	**AB**	**HGB**
CM1	Precision	0.833	0.842	0.836	0.828	**0.845**	0.837	0.836	0.840	0.818	0.814	0.828	HgB
	Recall	0.674	0.838	0.766	0.794	**0.850**	0.866	0.816	0.852	0.788	0.852	0.746	0.832
	F1-score	0.734	0.839	0.794	0.810	**0.844**	0.848	0.824	0.845	0.799	0.832	0.778	0.820
	AUC	0.632	0.678	0.615	0.689	0.670	0.707	0.619	0.659	0.613	0.653	0.637	0.824
KC1	Precision	0.809	0.827	0.813	0.828	0.818	0.833	0.813	0.829	0.803	0.800	0.806	0.708
	Recall	0.785	0.839	0.834	0.844	0.843	0.855	0.839	0.851	0.800	0.792	0.783	0.805
	F1-score	0.794	0.832	0.819	0.833	0.823	0.836	0.819	0.835	0.798	0.795	0.793	0.773
	AUC	0.763	0.798	0.749	0.770	0.783	0.798	0.748	0.778	0.755	0.747	0.686	0.785
KC2	Precision	0.795	0.810	0.794	0.825	0.784	0.792	0.795	0.794	0.818	**0.827**	0.806	0.755
	Recall	0.783	0.811	0.794	0.830	0.8	0.811	0.809	0.815	0.675	0.664	0.628	0.829
	F1-score	0.787	0.808	0.793	0.826	0.778	0.793	0.799	0.799	0.700	0.690	0.649	0.674
	AUC	0.774	0.793	0.708	0.763	0.781	0.787	0.666	0.761	0.724	**0.803**	0.728	0.699
PC1	Precision	0.902	0.919	0.927	0.920	0.893	**0.930**	0.925	0.935	0.890	0.903	0.880	0.732
	Recall	0.900	0.93	0.932	0.926	0.914	**0.939**	0.932	0.942	0.846	0.877	0.732	0.896
	F1-score	0.900	0.922	0.926	0.922	0.899	**0.932**	0.927	0.935	0.865	0.883	0.785	0.851
	AUC	0.867	0.808	0.817	0.889	0.749	0.816	0.867	0.879	0.700	0.732	0.611	0.868
							Average						
	Precision	0.835	0.848	0.842	0.850	0.835	**0.848**	0.842	0.849	0.832	0.836	0.830	0.840
	Recall	0.786	0.855	0.832	0.849	0.852	**0.867**	0.849	0.865	0.777	0.796	0.722	0.780
	F1-score	0.804	0.850	0.833	0.848	0.836	**0.852**	0.842	0.853	0.790	0.800	0.751	0.794
	AUC	0.759	0.769	0.722	0.778	0.746	**0.776**	0.725	0.769	0.698	0.734	0.666	0.733

Table 7. Results of the Relief&GAN methods

Dataset	**Performance Evaluation**	**Relief**											
		VanillaGAN				**CTGAN**				**WGANGP**			
		RF	**ET**	**AB**	**HGB**	**RF**	**ET**	**AB**	**HGB**	**RF**	**ET**	**AB**	**HGB**
CM1	Precision	0.845	0.848	0.843	0.835	0.824	**0.872**	0.855	0.844	0.829	0.834	0.839	0.831
	Recall	0.696	0.808	0.784	0.788	0.866	**0.890**	0.860	0.866	0.828	0.864	0.752	0.814
	F1-score	0.75	0.824	0.807	0.807	0.844	**0.868**	0.856	0.851	0.827	0.848	0.787	0.821
	AUC	0.647	0.716	0.672	0.673	0.676	0.711	0.716	0.670	0.648	0.637	0.661	0.690
KC1	Precision	0.815	0.824	0.811	0.827	0.815	**0.833**	0.825	0.843	0.815	0.821	0.816	0.824
	Recall	0.781	0.838	0.833	0.845	0.844	**0.854**	0.85	0.861	0.811	0.806	0.782	0.800
	F1-score	0.793	0.829	0.819	0.833	0.821	**0.836**	0.828	0.847	0.810	0.807	0.792	0.805
	AUC	0.773	0.798	0.737	0.776	0.788	0.797	0.757	0.788	0.759	0.746	0.720	0.740
KC2	Precision	0.811	0.789	0.789	0.798	0.781	0.802	0.785	0.789	0.816	0.809	0.797	0.812
	Recall	0.794	0.804	0.791	0.804	0.804	**0.817**	0.798	0.806	0.658	0.651	0.609	0.642
	F1-score	0.800	0.793	0.788	0.799	0.78	**0.800**	0.789	0.792	0.682	0.675	0.634	0.665
	AUC	0.796	0.789	0.705	0.772	0.781	0.783	0.652	0.772	0.780	0.791	0.738	0.760
PC1	Precision	0.900	0.913	0.916	0.915	0.871	**0.922**	0.914	0.929	0.893	0.902	0.893	0.898
	Recall	0.859	0.917	0.905	0.913	0.914	**0.932**	0.921	0.937	0.794	0.770	0.695	0.759
	F1-score	0.873	0.914	0.909	0.912	0.89	**0.921**	0.915	0.928	0.823	0.798	0.747	0.800
	AUC	0.782	0.783	0.839	0.814	0.691	0.775	0.841	0.802	0.680	0.707	0.646	0.685
							Average						
	Precision	0.843	0.843	0.840	0.844	0.822	**0.857**	0.845	0.851	0.838	0.841	0.836	0.841
	Recall	0.783	0.842	0.828	0.837	0.857	**0.873**	0.857	0.867	0.773	0.773	0.710	0.753
	F1-score	0.804	0.840	0.831	0.838	0.834	**0.856**	0.847	0.855	0.785	0.782	0.740	0.773
	AUC	0.757	0.767	0.738	0.759	0.734	**0.767**	0.742	0.758	0.717	0.720	0.691	0.719

V. CONCLUSION

Several software fault prediction models have proposed building high quality software with minimal testing resources by predicting faults at the early stage of software development. This study presents the combination of three different oversampling techniques (VanillaGAN, CTGAN and WGANGP) and four feature selection ranking methods (Chi-Squared, Information Gain, Fisher and Relief) to handle feature redundancy and data imbalance problems effectively. The experiment was carried out on four datasets extracted from the PROMISE repository. The experimental results showed that the performance of SFP models based on combinations of Chi-Squared&CTGAN, Information Gain & CTGAN, Fisher&CTGAN and Relief&CTGAN outperformed with the highest average precision, recall, F1-score and AUC values. Furthermore, all pairs of four feature selection ranking methods and VanillaGAN, CTGAN performance excels in comparison to the SFP models with the full features of datasets. In the future, we plan to generalize the findings of the presented work to more wrapper feature selection methods and data sampling techniques using more software fault datasets. And we will also examine the comparative performance of the presented methods for the cross-project defect prediction.

ACKNOWLEDGEMENTS

This research is funded by Funds for Science and Technology Development of the University of Danang under project number B2022-DN07-02.

REFERENCES

[1] O.F. Arar, K. Ayan, "Software defect prediction using cost-sensitive neural network," *Applied Soft Computing*, 33, 263-277 (2015).

[2] T. Menzies, et al., "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, 17, 375-407 (2010).

[3] J. Nam, "Survey on software defect prediction," Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep (2014).

[4] M. Tan, et al., "Online defect prediction for imbalanced data," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 99-108 (2015). IEEE.

[5] M.R. Lyu, et al., "Handbook of Software Reliability Engineering," *IEEE computer society press Los Alamitos,* vol. 222.

[6] M.H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., (1977).

[7] T.J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, (4), 308-320 (1976).

[8] S.R. Chidamber, C.F. Kemerer, "A metrics suite for object-oriented design," *IEEE Transactions on software engineering* 20(6), 476-493 (1994).

[9] K.E. Bennin, et al., "Empirical evaluation of cross-release effort-aware defect prediction models," in *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 214-221 (2016).

[10] S. Lessmann, et al., "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE transactions on software engineering*, 34(4), 485-496 (2008).

[11] T. Menzies, et al., "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, 33(1), 2-13 (2006).

[12] S.K. Pandey, et al., "Machine learning based methods for software fault prediction: A survey," *Expert Systems with Applications*, 172, 114595 (2021).

[13] S. Huda, et al., "An ensemble oversampling model for class imbalance problem in software defect prediction," *IEEE access* 6, 24184-24195 (2018).

[14] A. Balaram, et al., "Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm," *Automated Software Engineering*, 29(1), 6 (2022).

[15] S.S. Rathore, et al., "Generative oversampling methods for handling imbalanced data in software fault prediction," *IEEE Transactions on Reliability*, 71(2), 747-762 (2022).

[16] Z. Cui, et al., "Improving software fault localization by combining spectrum and mutation," *IEEE Access* 8, 172296-172307 (2020).

[17] G. Chandrashekar, F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, 40(1), 16-28 (2014).

[18] http://promise.site.uottawa.ca/SERepository/datasets-page.html

[19] S.C. Rathi, et al., "Empirical evaluation of the performance of data sampling and feature selection techniques for software fault prediction," *Expert Systems with Applications*, 223, 119806 (2023).

[20] M. Nevendra, P. Singh, "Software defect prediction by strong machine learning classifier," *Intelligent Computing and Communication Systems*, 321-329 (2021).

[21] M. Mangla, N. Sharma, "A sequential ensemble model for software fault prediction," *Innovations in Systems and Software Engineering*, 1-8 (2021).

[22] M. Jureczko, "Significance of different software metrics in defect prediction," *Software Engineering: An International Journal,* 1(1), 86-95 (2011).

[23] S.S. Rathore, A. Gupta, "A comparative study of feature-ranking and featuresubset selection techniques for improved fault prediction," in *Proceedings of the 7th India Software Engineering Conference*, pp. 1-10 (2014).

[24] A. Joon, R.K. Tyagi, K. Kumar, "Noise filtering and imbalance class distribution removal for optimizing software fault prediction using best software metrics suite," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pp. 1381-1389 (2020). IEEE.

[25] C. Manjula, L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Computing,* 22(Suppl 4), 9847-9863 (2019).

[26] S. Mehta, K.S. Patnaik, "Improved prediction of software defects using ensemble machine learning techniques," *Neural Computing and Applications*, 33, 10551- 10562 (2021).

[27] P.N. Tan, et al., *Introduction to Data Mining*, (First Edition). Addison-Wesley Longman Publishing Co., Inc., USA (2005).

KẾT HỢP CÁC KỸ THUẬT TRÍCH CHỌN ĐẶC TRƯNG VÀ LẤY MẪU DỮ LIỆU TRONG DỰ ĐOÁN LỖI PHẦN MỀM

Hà Thị Minh Phương, Nguyễn Thanh Long, Nguyễn Thanh Bình

TÓM TẮT: *Dự đoán lỗi phần mềm là quy trình xây dựng các mô hình để dự đoán lỗi trong giai đoạn đầu phát triển phần mềm. Dự đoán các môđun dễ bị lỗi phần mềm có thể giúp các nhà phát triển phân bổ nguồn lực kiểm thử hiệu quả hơn và tối ưu hóa chi phí bảo trì phần mềm. Tuy nhiên, hiệu suất của các mô hình dự đoán lỗi bị ảnh hưởng bởi chất lượng của bộ dữ liệu lỗi phần mềm. Các đặc trưng không liên quan và dư thừa của bộ dữ liệu có thể dẫn đến tác động tiêu cực đến tốc độ và độ chính xác của các mô hình máy học được áp dụng để huấn luyện. Ngoài ra, sự hiện diện của sự mất cân bằng dữ liệu khiến số lượng môđun bị lỗi ít hơn đáng kể so với số lượng môđun không bị lỗi là một thách thức trong dự đoán lỗi. Nghiên cứu này đã áp dụng 3 mô hình Generative Adversarial Network (GAN) gồm VanillaGAN, CTGAN và WGANGP cùng với 4 phương pháp lựa chọn đặc trưng như Chi-Squared, Information Gain, Fisher và Relief trên 4 bộ dữ liệu lỗi phần mềm. Phân tích so sánh được thực hiện bằng cách sử dụng 4 bộ phân loại khác nhau để dự đoán lỗi phần mềm. Chúng tôi lựa chọn các độ đo precision, recall, F1-score and Area Under the ROC (receiver operating characteristic curve) Curve (AUC) để đánh giá hiệu suất. Kết quả thử nghiệm cho thấy rằng sự kết hợp của CTGAN, VanillaGAN và các phương pháp lựa chọn đặc trưng vượt trội so với các mô hình dự đoán lỗi mà không cần áp dụng các phương pháp lấy mẫu dữ liệu và lựa chọn đặc trưng. Cặp kết hợp CTGAN và Relief thể hiện hiệu suất tốt nhất so với các kết hợp khác với giá trị trung bình của precision, recall, F1-score AUC cao nhất lần lượt là 0,857, 0,873, 0,856 và 0,767 trên Extra Tree.*