NS-PPO: A Two-Stage Data Resampling Framework for the Initial Phase of Software Defect Prediction

Xiaowei Zhao
School of Computer Science and Engineering
Digital Research Department, South China University of Technology, China Southern Power Grid, China
zhaoxw@csg.cn

Xuanye Wang
School of Computer Science and Engineering
South China University of Technology, China
202310188991@mail.scut.edu.cn

Siliang Suo
Electric Power Research Institute, China Southern Power Grid, China
suosl@csg.cn

Lu Lu
School of Computer Science and Engineering
South China University of Technology, China
lul@scut.edu.cn

Abstract: *Software Defect Prediction (SDP) is one of the most reliability assurance methods before the delivery of software projects. However, class imbalance is a common issue in software projects, significantly hindering the ability of SDP methods to distinguish between defective and non-defective instances. Recently, although several SDP imbalance-handling methods have achieved certain success, they still exhibit limitations in terms of reliability and applicability. To address this, this paper proposes Neighborhood cleaning rule and Synthetic minority oversampling technique with Proximal Policy Optimization-based adaptive sampling (NS-PPO), a two stage-based data resampling framework aimed at mitigating the impact of class imbalance in software projects. NS-PPO operates in two phases. In the first phase, a hybrid sampler that combines Neighborhood CLeaning rule (NCL) and Synthetic Minority Oversampling TEchnique (SMOTE) is employed to generate a large number of synthetic samples for minority instances. In the second phase, a Deep Reinforcement Learning (DRL)-based undersampler is designed to filter high-quality synthetic samples. These selected samples are then combined with real samples to form the training set for the SDP methods. Extensive experiments are conducted on 18 software projects from the PRedictOr Models In Software Engineering (PROMISE) and National Aeronautics and Space Administration (NASA) datasets, with Matthews Correlation Coefficient (MCC), Area Under the Curve (AUC), and F-measure used as evaluation metrics. The findings demonstrate that, regardless of whether expert metrics or semantic metrics are used as inputs for SDP methods, NS-PPO exhibits significant advantages over the state-of-the-art SDP imbalance-handling methods, including Learning-To-Rank UnderSampling (LTRUS).*

Keywords: *Software defect prediction, class imbalance, deep reinforcement learning.*

Received January 18, 2025; accepted June 30, 2025
https://doi.org/10.34028/iajit/22/5/3

1. Introduction

After a software project is released, even a minor defect can result in serious consequences, including system crashes, data loss, and reduced user satisfaction [39]. Therefore, identifying software defects early in the development cycle is essential for ensuring the software reliability. Existing Software Defect Prediction (SDP) methods utilize historical data to build machine learning-based classifiers, aiming to predict the presence of defects in future datasets [36]. However, software projects often suffer from significant class imbalance, where the number of non-defective instances may be much larger than that of defective ones. This issue results in an SDP classifier that prioritizes the majority class while neglecting the minority class, thereby leading to biased prediction performance [12]. Software engineering practitioners employ data resampling techniques to mitigate the class imbalance issue. Data resampling techniques can be broadly categorized into two types: Undersampling and oversampling [1]. Undersampling reduces class imbalance by removing a portion of samples from the majority class, while oversampling enhances the representation of the minority class by generating synthetic samples. Together, these techniques help achieve a balanced class distribution in the training set.

Common undersampling techniques used in SDP include Random Under Sampling (RUS) and Edited Nearest Neighbor (ENN). RUS reduces class imbalance by randomly removing samples from the majority class, though this approach may risk discarding valuable information [4]. ENN leverages the K-Nearest Neighbors (KNN) algorithm to eliminate noisy or borderline samples [26]. Typical oversampling techniques include the Synthetic Minority Oversampling TEchnique (SMOTE) and cluster centroids. SMOTE generates synthetic samples by interpolating between existing minority class instances, resulting in a more balanced class distribution [4]. Cluster centroids employs the K-means clustering algorithm to compute cluster centers, which are then used to generate new samples [11]. Previous SDP methods have typically tackled the class imbalance issue in software projects by using common data resampling techniques. Khleel and Nehez [20] combined SMOTE with Bidirectional Long Short-Term Memory (Bi-LSTM) and successfully demonstrated its effectiveness. Feng *et al*. [11] proposed a learning-based termination condition for RUS, treating the undersampling process of software instances as a ranking task, where instances with lower ranks are prioritized for removal. Yang *et al*. [40] conducted a comparative analysis of data resampling techniques in SDP methods, and the results suggest that, oversampling outperform undersampling. Despite the significant progress made in previous studies, several shortcomings remain evident:

1. Common data resampling methods used in SDP studies rely on predefined sampling rules, lacking the ability to dynamically adjust based on the feature distribution of software instances or the training performance [19, 20, 26].
2. Previous SDP imbalance-handling studies have mainly focused on the expert metrics-based features of software instances, neglecting the semantic features [11, 12].

Extensive SDP studies indicate that semantic features, compared to expert features, contain richer contextual information, which can achieve superior performance [31, 36, 42]. Furthermore, even within the same project, there can be significant differences in the feature distributions of expert metrics and semantic features. Therefore, while some SDP imbalance-handling methods perform well with expert metrics, they often cannot be directly transferred to semantic features.

To address the above issues, this paper proposes a two-stage data sampling framework, Neighborhood cleaning rule and Synthetic minority oversampling technique with Proximal Policy Optimization-based adaptive sampling (NS-PPO), to mitigate the class imbalance in SDP methods. Specifically, NS-PPO consists of two stages. In the first stage, the framework leverages prior knowledge from previous SDP studies to develop a hybrid sampler that combines the Neighborhood CLeaning rule (NCL) and SMOTE [6]. This sampler first applies NCL to remove overlapping class samples. Then, based on the cleaned dataset, it uses SMOTE to generate a large number of synthetic samples for the minority class. In the second stage, the framework introduces a Deep Reinforcement Learning (DRL)-based undersampler, framing the undersampling process of synthetic samples as a Markov Decision Process (MDP) to dynamically optimize the sampling strategies. By constructing a simple SDP classifier, the sampler iteratively refines the sampling strategy based on the classifier's performance on the validation set, allowing for more effective sample selection. To validate the effectiveness of NS-PPO, experiments are conducted on 18 open-source software projects from the Predictor Models in Software Engineering (PROMISE) and National Aeronautics and Space Administration (NASA) datasets, using F-measure, Area Under the Curve (AUC), and Matthews Correlation Coefficient (MCC) as evaluation metrics and employing Scott-Knott Effect Size Difference (ESD) tests. The experimental results demonstrate that the method using NS-PPO for software instance sampling outperforms baseline methods, including Learning-To-Rank Under-Sampling (LTRUS).

In summary, the main contributions of this paper are as follows:

- By introducing DRL theory, this paper models the undersampling process of synthetic software instances as a MDP, enabling the sampler to dynamically adjust based on the characteristics of instances and the training performance.
- A two-stage sampling framework, NS-PPO, is proposed to mitigate the imbalance issue in SDP methods. NS-PPO first generates a large number of synthetic samples using NCL and SMOTE, then optimizes the undersampling process using the Proximal Policy Optimization (PPO) algorithm. Ablation experiments highlight the essential role of each step.
- Experimental results on 18 software projects show that NS-PPO achieves significant performance over baseline methods in terms of both expert metrics-based features and semantic features.

The structure of this paper is as follows: Section 2 presents the related work; section 3 provides a detailed description of the proposed method; section 4 outlines the experimental setup; section 5 discusses the experimental results; and section 6 concludes the paper and offers suggestions for future work.

2. Related Work

In this section, we introduce the previous SDP research and the background on data resampling techniques.

2.1. The SDP Research

The goal of SDP is to support software development teams in ensuring software reliability throughout the development lifecycle [27]. By leveraging historical data, SDP methods construct defect prediction models that assist developers in identifying potential defects in pre-release software [8, 36].

In early SDP research, the community primarily focused on designing effective expert metrics to construct machine learning-based classifiers. Okutan and Yildiz [30] proposed two additional metrics, Number of Developers (NoD) and Source Code Quality

(LOCQ), and used a Bayesian network to assess the impact of metrics on defect prediction. Nam and Kim [28] focused on the magnitude of expert metrics to determine the presence of defects in software instances.

With the widespread adoption of deep learning techniques across various fields, the community has found that deep representation learning can effectively capture potential defects from semantic features [24]. Pan *et al*. [31] designed an Improved Convolutional Neural Networks (ICNN) model to perform representation learning for Abstract Syntax Trees (ASTs) of source code. Zhou *et al*. [42] introduced a Two-Stage Encoding (TSE) method to identify defect information within the code context. Liu *et al*. [25] employed the pre-trained UniXcoder to extract defect-prone features from source code. Yang *et al*. [38] extracted Program Dependency Graphs (PDGs) from source code and constructed a graph convolutional network for fine-grained defect prediction. Jiang *et al*. [17] proposed an enhanced AST-based defect prediction method that integrates semantic and syntactic information. Although semantic features provide rich contextual information from source code, the deep learning techniques need to extract these features often entail substantial computational resources. In contrast, expert metrics-based SDP methods continue to be a popular focus in current SDP research, owing to their simple model architecture and rapid training efficiency.

2.2. The Data Resampling Technique

Software projects face the challenge of class imbalance, which often causes classifiers to prioritize the majority class while neglecting the minority class. Consequently, migrating the class imbalance issue has remained a central focus in SDP research.

At the data level, the community employs data resampling techniques to migrate the class imbalance issue. Data resampling techniques can be divided into two main categories: Undersampling and oversampling methods. Chen *et al*. [6] proposed an SDP method that achieves balanced subsets by performing multiple rounds of RUS and using the AdaBoost mechanism to construct an SDP classifier. Goyal [14] introduced a neighborhood-based undersampling technique, which outperformed traditional undersampling methods in SDP. Feng *et al*. [11] highlighted that RUS could lead to the loss of critical information. To mitigate this problem, they proposed an optimized termination condition based on the differential evolution algorithm, which adapts to the characteristics of the training set to minimize the information loss caused by RUS. Liu *et al*. [26] employed a random search approach to optimize the hyperparameters of SMOTE and neural networks. Khleel and Nehez [19] employed the SMOTE-Tomek method to generate synthetic samples for the minority class while removing overlapping instances. Kim and Chung [22] evaluated the performance of four oversampling techniques on both machine learning-based and deep learning-based SDP methods. Their results indicated that oversampling techniques are more suitable for machine learning-based SDP methods. However, in deep learning tasks, they still relied on expert-based metrics as inputs, overlooking the importance of semantic metrics. Gupta *et al*. [15] utilized the SMOTE-tomek method to mitigate class imbalance and employed a genetic algorithm-optimized SDP classifier.

3. Methodology

This section offers a comprehensive explanation of each step within the NS-PPO framework. Specifically, NS-PPO consists of two key stages: in the first stage, a hybrid sampler is constructed to generate a large number of synthetic samples for the minority class instances. In the second stage, a DRL-based undersampler is designed to filter synthetic samples based on the feature distribution and the training performance. The filtered samples are then combined with the original training set to serve as the training data for SDP methods. The overall framework of NS-PPO is shown in Figure 1.

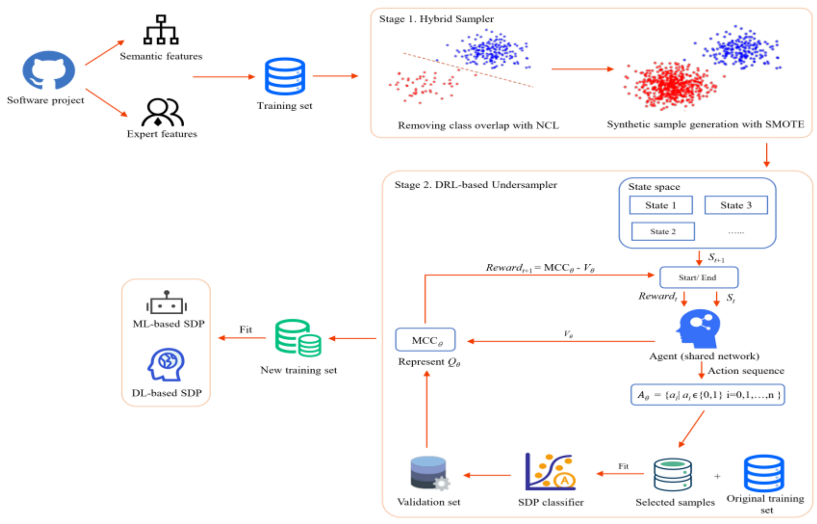


Figure 1. The overall framework of NS-PPO.

3.1. Data Preprocessing

As described in section 2, SDP methods rely on expert metrics or semantic features extracted from instances. Expert metrics are derived through the manual analysis to assess various aspects, including code quality, complexity, and readability. For example, cyclomatic complexity, which quantifies the number of branches and loops in the code, aids software engineers in evaluating the logical structure of instances. Lower cyclomatic complexity indicates simpler logic, enhancing the code's readability and ease of understanding. However, extracting these metrics relies heavily on expert knowledge, making the process both time-consuming and costly. Additionally, these metrics mainly capture structural information and offer limited representation of the richer contextual semantics [25, 42]. Both the PROMISE and NASA datasets provide expert metrics for each instance, eliminating the need for additional preprocessing and allowing for direct use. Moreover, a detailed description of the expert metrics used in this study can also be found in [18].

Semantic features represent the logical meaning and functional information embedded within the instance. In SDP methods, static analysis of the source code is employed to extract its structured representations, such as the ASTs. AST is a tree-like structure where each node represents a specific syntactic element of the source code. An example of an AST is shown in Figure 2. Compared to handcrafted metrics, AST-based semantic features not only capture the structural details of the code but also provide richer contextual information, leading to superior prediction performance. Building on the preprocessing methods described in ICNN and TSE, this study utilizes the javalang tool to extract the ASTs as semantic features. Specific node types are retained and transformed into vectorized sequences [31, 42].

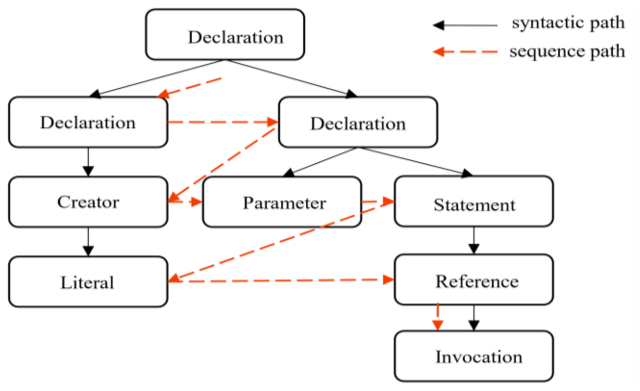
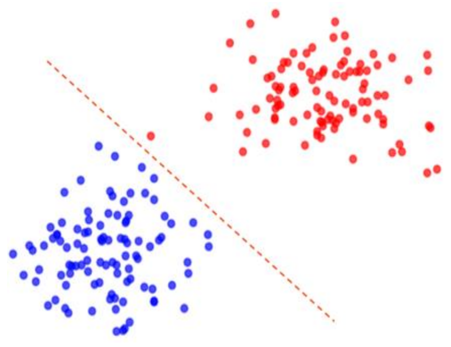


Figure 2. An example of an AST Structure. Black arrows represent the data flow between AST nodes, while red arrows indicate the extraction order of nodes in SDP methods.

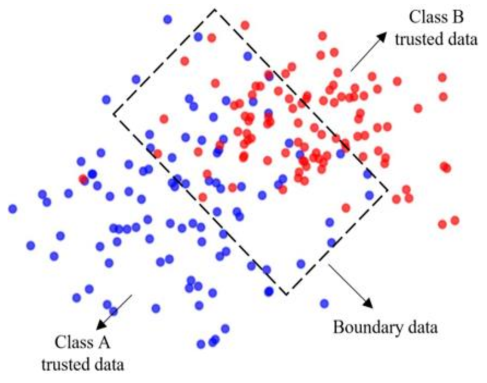
3.2. Hybrid Sampler for Synthetic Sample Generation

The bias caused by differences in sample sizes across classes is not the only factor that makes the challenge of model learning in the class imbalance issue [9]. In fact, even with balanced datasets, the presence of class overlap can still lead to a significant drop in classifier performance. In previous SDP studies, data resampling techniques have primarily focused on balancing sample quantities, while the issue of class overlap has often been overlooked [2, 10].

The class overlap issue occurs when samples from different classes share the same region within the feature space. In this region, samples originate from distinct classes but exhibit similar feature distributions. Due to the high similarity, SDP classifiers struggle to distinguish between them during the training process. Figure 3 contrasts an ideal balanced data distribution with one that exhibits the class overlap phenomenon.



a) Imbalanced dataset.



b) Overlap between classes.

Figure 3. A comparative example of an ideal balanced dataset versus a dataset with the class overlap issue.

To mitigate the issue of feature overlap between software instances, the first phase of hybrid sampler integrates a domain-cleaning approach based on the KNN algorithm, specifically the NCL [23]. The cleaning process of NCL includes the following steps:

- Neighborhood construction: the neighborhood of each sample under evaluation is determined by arranging other known samples in ascending order based on the Euclidean distance. The neighborhood consists of k-nearest samples, which may include samples from both majority and minority class. The Euclidean distance is calculated as follows:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2} \quad (1)$$

where x_i and x_j are the feature vectors of the samples, and *n* is the dimensionality of the feature space.

- Classification assessment: for the sample under evaluation, if the KNN classifier identifies it as misclassified or finds that its presence causes neighboring minority class samples to be misclassified as majority class samples, the sample is deemed detrimental to classification performance.
- Sample cleaning: all samples labeled as boundary samples are removed from the training set to reduce redundant features.

SMOTE generates synthetic samples for the minority class by interpolating within its feature space [29]. Following the resolution of inter-class overlap, the hybrid sampler employs the SMOTE to produce a substantial number of synthetic samples. The detailed steps are as follows:

- Neighborhood construction: for each minority class sample, the KNN algorithm is used to find its neighbors in the feature space.
- Synthetic sample generation: a synthetic sample is generated by randomly selecting one neighbor from the nearest neighbors of each minority class sample and performing linear interpolation based on the distance between the sample and the selected neighbor. The process is illustrated as follows:

$$x_{new} = x_{original} + \lambda \times \left(x_{neighbor} - x_{original}\right) \tag{2}$$

Wher λ e a value randomly generated in the range [0, 1], $x_{oriainal}$ is the current sample, and $x_{neighor}$ is the selected neighbor.

In the first stage of NS-PPO, the hybrid sampler generates synthetic samples until the number of minority class samples reaches five times that of the majority class. The purpose of generating a large number of synthetic samples is to enable the undersampler in the second satge to thoroughly learn the feature distribution of the software instances.

3.3. DRL-Based Undersampler

Reinforcement Learning (RL) focuses on training an agent to discover the optimal action policy for a specific task through trial-and-error interactions and feedback from its environment. The principle of RL lies in enabling an agent to interact with its environment by observing its state, taking actions, and receiving rewards or penalties, thereby continuously optimizing its decision-making process [37]. Notably, RL consists of the following key components:

- Agent: the agent is the decision-maker that interacts with the environment, performs actions, and learns from the feedback it receives.
- State (s_t): s_t represents the agent's perception of the environment at a specific time step.
- Action (a_t): a_t represents the decision the agent makes in a given state.
- Reward (*rt*): *rt* is the feedback signal provided by the environment after the agent takes an action.
- Policy (π): π defines the agent's action-selection strategy, represented as a probability distribution $\pi(a|s)$.

The goal of the agent is to learn an optimal policy through continuous interaction with the environment, maximizing the long-term accumulated rewards. RL introduces state value functions and action value functions to evaluate the effectiveness of a policy. The state value $V^{\pi}(s)$ evaluates the expected cumulative reward that the agent can obtain by following policy π from state s, as shown in the following Equation (3):

$$V^{\pi}(s) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_t = s\right] \tag{3}$$

Where γ is the discount factor, which is used to measure the importance of future rewards. The action value function $Q^{\pi}(s, a)$ evaluates the expected cumulative reward of an agent starting from state s, taking action a, and then following policy π. This can be expressed as follows:

$$Q^{\pi}(s,a) = E_{\pi}[r_{t+1} + \gamma \cdot Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a] \tag{4}$$

DRL combines the decision-making capability of RL with the representational power of deep neural networks, leveraging neural networks to approximate value functions or policies. In the second stage of NS-PPO, the undersampling process of synthetic samples is formulated as a MDP, and the undersampler is optimized using the PPO algorithm. PPO [34] is a classic RL algorithm designed for policy optimization. It employs the concept of importance sampling to evaluate the performance of a new policy based on experiences gathered from the old policy. At each time step, the importance sampling ratio is calculated as the probability ratio between the new policy and the old policy, as shown below:

$$r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{5}$$

where $\pi_{\theta}(a_t \backslash s_t)$ represents the probability of selecting action at in state *st* under the new policy π_{θ}. In policy optimization, PPO introduces a "clipped" objective function to constrain the magnitude of policy updates during each iteration, ensuring that the updated policy stays close to the current policy. This function enhances the stability of the agent throughout the training process and is defined as follows:

$$L^{CLIP}(\theta) = E_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, \quad 1+\epsilon)A_t] \tag{6}$$

Where ϵ is a clipping hyperparameter that constrains the range of policy updates to ensure stability. In NS-PPO, ϵ is set to 0.2.

In this framework, the synthetic samples generated in the first phase constitute the state space for undersampler. These samples are divided into five batches, which are sequentially fed into the agent as input states. The input states are first passed through a shared network, whose feature extraction layers consist of two fully connected layers and two ReLU functions, resulting in a 4-dimensional feature vector. Subsequently, two independent linear layers are used to generate the state value V_θ and the action probability A_θ. The structure of the shared network is shown in Figure 4. The action probability A_θ guides the agent in sample selection: when A_θ=0, the sample is discarded; when A_θ=1, the sample is retained. Additionally, the agent's reward mechanism is based on the MCC values, calculated on the validation set using a lightweight SDP method. A Random Forest (RF) classifier is employed for expert metric-based features, while the ICNN method is used for semantic features. The MCC value is treated as the action value Q_θ while the difference between Q_θ and V_θ is regarded as the reward for the action, as expressed in the following Equation (7):

$$Reward_{NS-PPO} = Q_{\pi_\theta} - V_{\pi_\theta} \tag{7}$$

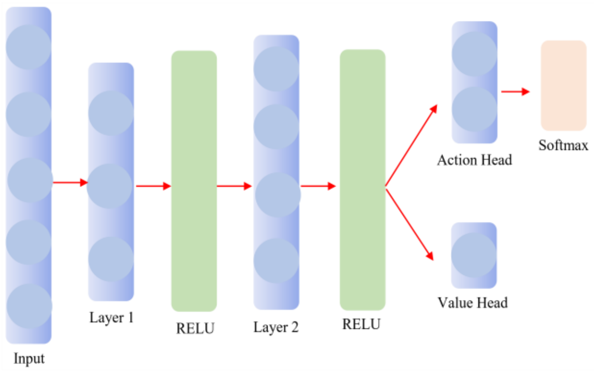


Figure 4. The structure of the shared network.

After iterative optimization, all the synthetic samples generated from the hybrid sampler are provided to the agent, and the number of selected samples is determined by the agent based on its learned policy. Finally, the high-quality samples selected by the agent are combined with the original training data to construct the final training set for SDP methods.

4. Experimental Setup

This section provides an overview of the experimental setup for this study, including the datasets used, baseline methods, and evaluation metrics applied.

4.1. Research Questions

This paper seeks to address on the following Research Questions (RQs) to fill the gaps left by previous SDP imbalance-handling studies:

- **RQ1:** how does the performance of NS-PPO compare to baseline methods when handling expert metrics?

Motivation: previous SDP imbalance-handling studies have often relied on static environments [19, 20, 26], making it challenging to adjust based training performance. To address this limitation, this paper introduces NS-PPO. RQ1 conducts a comparative analysis against baseline methods, including LTRUS, to demonstrate the effectiveness of NS-PPO.

- **RQ2:** how does the performance of NS-PPO compare to baseline methods when handling semantic features?

Motivation: previous SDP imbalance-handling studies have predominantly focused on expert metrics, overlooking the potential of semantic features [11, 12, 19, 20, 26]. However, semantic features-based SDP methods tend to achieve superior performance. RQ2 aims to evaluate the applicability of NS-PPO in handling semantic features by comparing it with baseline methods.

- **RQ3:** does each component of NS-PPO contribute positively to its overall performance?

Motivation: RQ3 designs an ablation study to evaluate the effectiveness of each component within NS-PPO. By systematically removing the NCL and the DRL-based undersampler from the framework, the study analyzes their impact on prediction performance, providing valuable insights for practitioners in the SDP field.

4.2. Dataset

The experimental datasets are sourced from 15 open-source software projects from PROMISE and 3 open-source projects from NASA. As two of the most widely used SDP datasets, PROMISE and NASA frequently appear in previous SDP studies [12, 36, 39]. Detailed information about the selected projects is provided in Table 1. It is important to note that the NASA dataset does not include semantic features. Therefore, RQ2 is conducted exclusively using the PROMISE dataset.

Table 1. Description of 18 projects.

Dataset	Project	Instance	Defect ratio
PROMISE	xerces-1.2	439	16.17%
PROMISE	lucene-2.4	330	61.52%
PROMISE	velocity-1.4	195	75.38%
PROMISE	ant-1.4	177	22.60%
PROMISE	poi-3.0	438	61.16%
PROMISE	camel-1.4	848	17.10%
PROMISE	synapse-1.1	222	27.03%
PROMISE	lucene-2.0	186	48.92%
PROMISE	xalan-2.5	762	50.79%
PROMISE	velocity-1.6.1	229	34.06%
PROMISE	ivy-2.0	352	11.36%
PROMISE	xerces-1.3	452	15.27%
PROMISE	xerces-1.4.4	331	64.35%
PROMISE	camel-1.6	935	20.11%
PROMISE	jedit-4.3	487	2.26%
NASA	JM1	9591	18.34
NASA	KC1	2095	15.51%
NASA	PC3	1099	12.56%

4.3. Baseline Methods

To evaluate the performance of NS-PPO in SDP methods, this paper compares it with six baseline methods: RUS, borderline, cluster, SMOTE-ENN, LTRUS, and LTRUS-ratio. RUS, borderline, cluster, and SMOTE-ENN are widely used data resampling techniques in previous SDP studies [11, 12], while LTRUS and its variant LTRUS-ratio represent State-Of-The-Art (SOTA) data resampling methods specifically designed to address the class imbalance issue in SDP. A brief description of each method is provided below.

- **RUS**: is a data undersampling technique that achieves class balance by randomly removing samples from the majority class [21].
- **Borderline**: is a boundary-based oversampling technique that generates minority class samples near the decision boundary, improving the classifier's ability to discriminate between classes [16].
- **Cluster**: is a data undersampling technique that employs a clustering algorithm, such as K-means, to group majority class samples. It then substitutes each cluster with its centroid, effectively reducing the number of majority class instances [11].
- **Smote-ENN**: generates synthetic samples using SMOTE and then removes noisy instances through ENN, thereby enhancing the quality of the training data [3].
- **LTRUS**: treats the undersampling process of software instances as a learning-to-rank task. It optimizes a linear model to rank the majority class instances and removes those ranked at the bottom [11].
- **LTRUS-ratio**: is a variant of LTRUS that optimizes the final defect rate, rather than simply balancing the number of instances between classes [11].

4.4. SDP Methods

In RQ1, we employ three commonly used machine learning classifiers from previous SDP studies [13, 41]: RF [5], KNN [7], and Decision Tree (DT) [33]. In RQ2, we use two deep learning-based SDP methods: ICNN [31] and TSE [42].

4.5. Evaluation Metrics

The prediction results of an SDP method can be categorized into four cases:

1) True Positive (TP): the actual class is positive, and the model correctly predicts it as positive.
2) True Negative (TN): the actual class is negative, and the model correctly predicts it as negative.
3) False Positive (FP): the actual class is negative, but the model incorrectly predicts it as positive.
4) False Negative (FN): the actual class is positive, but the model incorrectly predicts it as negative.

In this study, we use three widely adopted evaluation metrics in SDP research: F-measure, MCC, and AUC. F-measure is derived from Recall and Precision. The specific formulas for these metrics are as follows:

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{10}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{11}$$

$$AUC = \int_0^1 \left(\frac{TP}{TP + FN}\right) d\left(\frac{FP}{FP + TN}\right) \tag{12}$$

The values of F-measure and AUC range from 0 to 1, while MCC ranges from -1 to 1. For all metrics, higher values indicate superior performance.

After obtaining the evaluation metrics, this experiment employs the Scott-Knott ESD test to perform statistical analysis on the impact of NS-PPO and baseline methods for SDP methods. This test is widely used in software engineering research for its ability to partition a set of methods into statistically distinct, non-overlapping groups based on their performance. Specifically, the test utilizes hierarchical clustering combined with effect size analysis to ensure that groupings are both statistically significant and practically meaningful. Further details on the Scott-Knott ESD test can be found in [35].

4.6. Development Environment

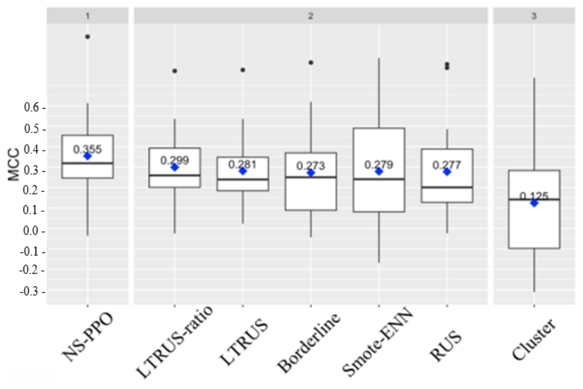
The experiment is conducted on a Linux machine equipped with an AMD MI210 GPU. All experiment procedures are executed within a Python 3.8 environment, utilizing three primary Python libraries: pytorch (version 2.1.2+rocm5.6), sklearn (version 1.5.2) [32], and imblearn (version 0.12.4).

In the second stage of the NS-PPO, the hidden layer dimension of the shared network is set to 4. The Adam optimizer is used with a learning rate of 1e-5, and the model is trained for 20 epochs.

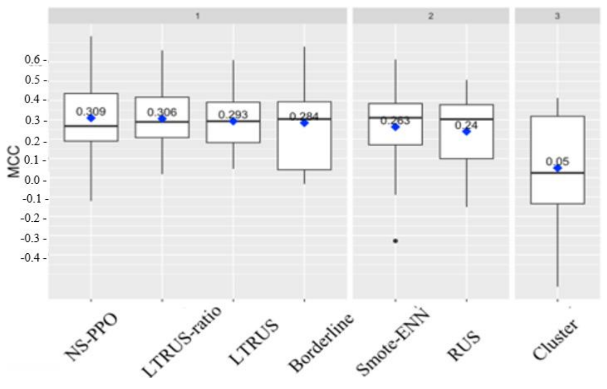
5. Results and Discussion

5.1. Answer to RQ1

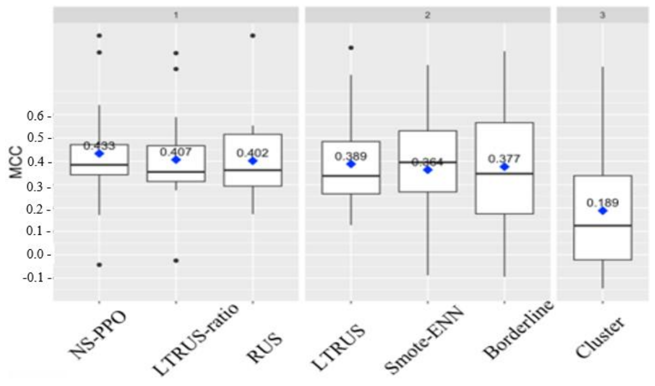
RQ1 evaluates the impact of NS-PPO and baseline methods on expert metrics-based SDP methods. Figure 5 shows the average F-measure performance and Scott-Knott ESD test results of NS-PPO and six baseline methods across 18 software projects. The boxplots illustrate the performance distribution, with mean values marked in blue. To evaluate the statistical significance of performance differences, the Scott-Knott ESD test was employed to group methods based on distinguishable differences. The group labels above each boxplot indicate relative rankings: methods within the same group are not significantly different, while those in different groups are. Figures 6 and 7 present the same comparison using AUC and F-measure as evaluation metrics, respectively. Moreover, to provide a more comprehensive view, Tables 2, 3, and 4 present the MCC values of NS-PPO and the six baseline methods on each software project using DT, KNN, and RF classifiers, respectively. Within each table, the "W/D/L" row statistics summarize the number of wins, draws, and losses for NS-PPO when compared against the baseline methods on each respective project.



a) MCC values on DT.

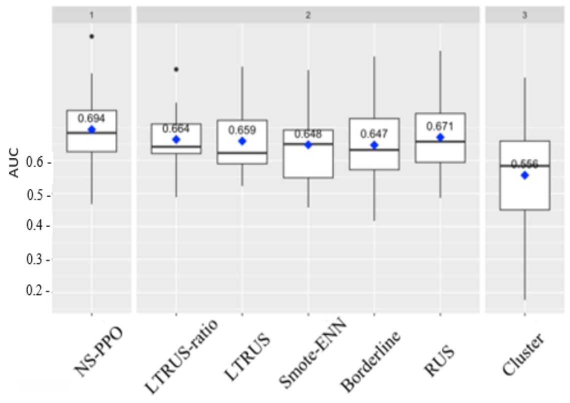


b) MCC values on KNN.

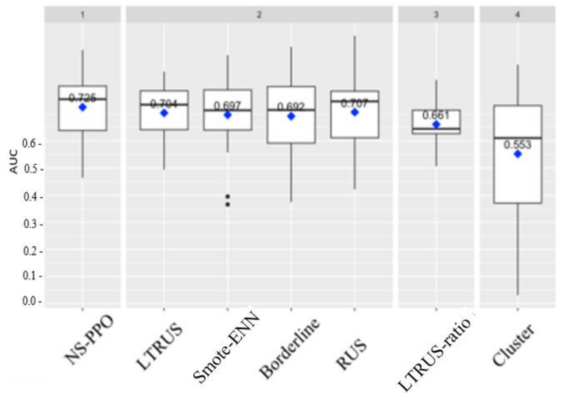


c) MCC values on RF.

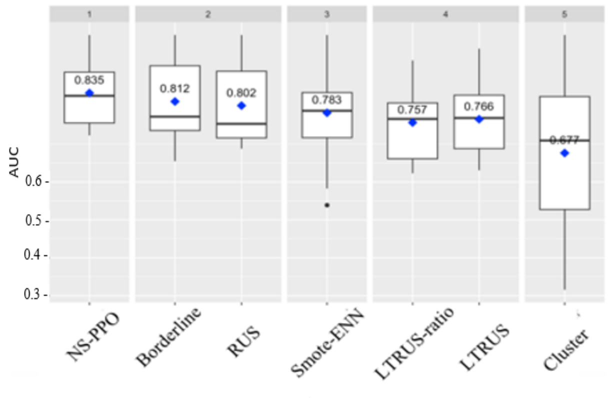
Figure 5. MCC values and ESD test results for NS-PPO and baseline methods across 18 software projects.



a) AUC values on DT.

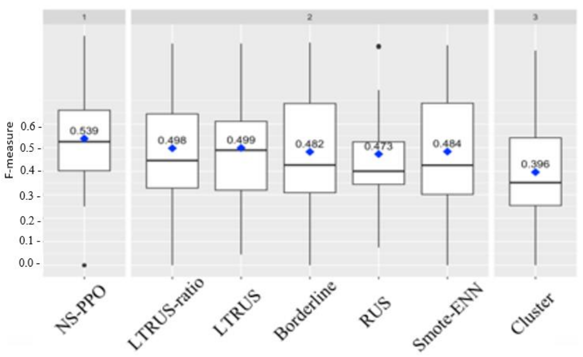


b) AUC values on KNN.

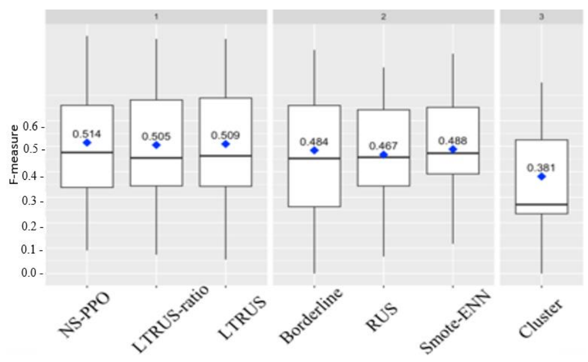


c) AUC values on RF.

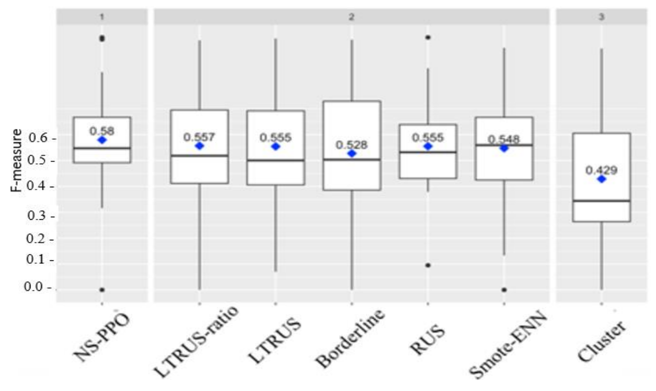
Figure 6. AUC values and ESD test results for NS-PPO and baseline methods across 18 software projects.



a) F-measure values on DT.



b) F-measure values on KNN.



c) F-measure values on RF.

Figure 7. F-measure values and ESD test results for NS-PPO and baseline methods across 18 software projects.

Table 2. MCC values of NS-PPO and baseline methods on DT.

	RUS	Borderline	Cluster	Smote-ENN	LTRUS	LTRUS-ratio	NS-PPO
xerces-1.2	0.080	0.078	0.226	0.061	0.145	0.165	0.087
lucene-2.4	0.486	0.562	0.190	0.334	0.231	0.418	0.385
velocity-1.4	0.786	0.629	0.289	0.236	0.515	0.467	0.404
ant-1.4	0.152	0.035	-0.121	0.000	0.371	0.217	0.473
poi-3.0	0.412	0.312	0.311	0.620	0.536	0.536	0.581
camel-1.4	0.197	0.127	0.018	0.027	0.107	0.123	0.317
synapse-1.1	0.335	0.647	0.321	0.467	0.248	0.255	0.321
lucene-2.0	0.036	-0.168	-0.069	0.382	0.203	0.389	0.15
xalan-2.5	0.123	0.091	0.145	0.325	0.188	0.264	0.279
velocity-1.6.1	-0.024	0.280	0.516	0.189	0.265	0.394	0.399
ivy-2.0	0.119	0.533	0.138	0.204	0.280	0.271	0.553
xerces-1.3	0.406	0.363	0.005	0.528	0.481	0.371	0.614
xerces-1.4.4	0.804	0.834	0.738	0.812	0.776	0.771	0.939
camel-1.6	0.305	0.191	-0.125	0.027	0.037	0.136	0.106
jedit-4.3	0.141	-0.043	-0.189	-0.043	0.023	-0.025	-0.036
JM1	0.205	0.219	-0.107	0.265	0.183	0.229	0.236
KC1	0.183	0.263	-0.311	0.299	0.269	0.208	0.285
PC3	0.246	0.072	0.268	0.173	0.201	0.199	0.289
W/D/L	12/0/6	13/0/5	14/1/3	12/0/6	13/0/5	12/0/6	

Table 3. MCC values of NS-PPO and baseline methods on KNN.

	RUS	Borderline	Cluster	Smote-ENN	LTRUS	LTRUS-ratio	NS-PPO
xerces-1.2	-0.089	0.087	0.019	-0.089	0.109	0.129	0.080
lucene-2.4	0.385	0.679	0.385	0.385	0.307	0.283	0.437
velocity-1.4	0.346	0.603	0.3333	0.29	0.610	0.609	0.733
ant-1.4	-0.152	-0.033	-0.265	-0.328	0.046	0.018	-0.121
poi-3.0	0.354	0.354	0.412	0.485	0.489	0.497	0.467
camel-1.4	0.227	0.026	-0.095	0.111	0.157	0.169	0.134
synapse-1.1	-0.071	-0.018	-0.172	0.324	0.228	0.251	0.273
lucene-2.0	0.368	0.368	0.262	0.382	0.285	0.295	0.231
xalan-2.5	0.299	0.305	0.273	0.169	0.444	0.442	0.198
velocity-1.6.1	0.088	0.280	-0.042	0.032	0.300	0.314	0.434
ivy-2.0	0.408	0.348	0.373	0.341	0.214	0.238	0.262
xerces-1.3	0.505	0.604	0.055	0.58	0.412	0.443	0.528
xerces-1.4.4	0.507	0.558	0.413	0.613	0.588	0.660	0.614
camel-1.6	0.092	0.019	-0.257	0.174	0.171	0.230	0.249
jedit-4.3	0.119	-0.029	-0.565	0.207	0.046	0.081	0.167
JM1	0.307	0.261	-0.110	0.297	0.233	0.200	0.187
KC1	0.240	0.402	-0.144	0.438	0.305	0.305	0.300
PC3	0.382	0.303	0.030	0.328	0.327	0.342	0.394
W/D/L	13/0/5	9/0/9	15/0/3	10/0/8	10/0/8	9/0/9	

Table 4. MCC values of NS-PPO and baseline methods on RF.

	RUS	Borderline	Cluster	Smote-ENN	LTRUS	LTRUS-ratio	NS-PPO
xerces-1.2	0.293	-0.094	0.163	-0.043	0.280	0.331	0.357
lucene-2.4	0.550	0.613	0.583	0.619	0.509	0.478	0.634
velocity-1.4	0.544	0.866	0.333	0.545	0.770	0.795	0.866
ant-1.4	0.478	0.119	-0.121	0.473	0.194	0.299	0.396
poi-3.0	0.553	0.428	0.412	0.486	0.594	0.504	0.473
camel-1.4	0.296	0.156	0.086	0.174	0.251	0.327	0.285
synapse-1.1	0.271	0.449	-0.037	0.549	0.339	0.355	0.321
lucene-2.0	0.308	0.149	0.263	0.267	0.377	0.354	0.367
xalan-2.5	0.354	0.354	0.302	0.275	0.411	0.405	0.344
velocity-1.6.1	0.280	0.233	0.464	-0.088	0.269	0.304	0.342
ivy-2.0	0.373	0.718	0.000	0.449	0.369	0.390	0.449
xerces-1.3	0.528	0.604	0.055	0.640	0.509	0.589	0.641
xerces-1.4.4	0.938	0.871	0.805	0.812	0.886	0.863	0.938
camel-1.6	0.370	0.340	0.340	0.353	0.324	0.339	0.421
jedit-4.3	0.174	-0.029	-0.029	-0.029	0.127	-0.025	-0.043
JM1	0.307	0.280	-0.110	0.297	0.192	0.309	0.374
KC1	0.239	0.422	-0.144	0.438	0.258	0.276	0.170
PC3	0.382	0.303	0.030	0.328	0.336	0.433	0.466
W/D/L	11/1/6	12/1/5	15/0/3	12/1/5	12/0/6	12/0/6	

In terms of prediction performance, Figure 5 illustrates that NS-PPO achieves the highest average MCC values across the DT, KNN, and RF classifiers. The performance is most significant with the DT classifier, where NS-PPO outperforms LTRUS-ratio by 5.6%, LTRUS by 7.4%, SMOTE-ENN by 8.2%, borderline by 7.6%, RUS by 7.8%, and cluster by 23%. As shown in Table 2, it outperforms LTRUS-ratio in 12 projects, LTRUS in 13 projects, SMOTE-ENN in 12 projects, Borderline in 13 projects, RUS in 12 projects, and Cluster in 14 projects. Moreover, Tables 3 and 4 further demonstrate that NS-PPO consistently surpasses the advanced baseline LTRUS-ratio in 9 projects under the KNN classifier and in 12 projects under the RF classifier. Figure 6 highlights that NS-PPO achieves the highest average AUC values across all classifiers. Specifically, on the DT classifier, NS-PPO outperforms the second-best LTRUS by 3%; on the KNN classifier, it surpasses the second-best LTRUS by 2.2%; and on the RF classifier, it exceeds the second-best Borderline by 2.3%. Figure 7 shows that NS-PPO achieves the optimal F-measure values across all SDP classifiers. Specifically, on the DT classifier, NS-PPO outperforms the second-best LTRUS by 4.1%; on the KNN classifier, it surpasses the second-best LTRUS by 0.9%; and on the RF classifier, it exceeds the second-best LTRUS by 2.3%.

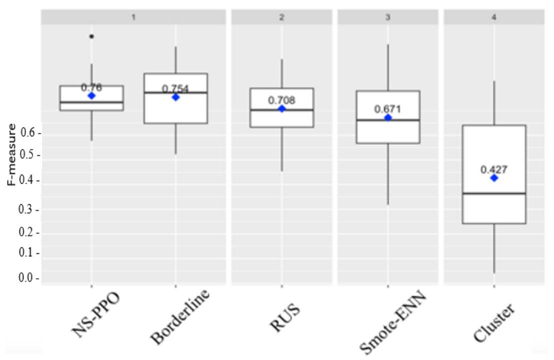
In terms of the Scott-Knott ESD test results, NS-PPO demonstrates statistically significant advantages over the baseline methods across all evaluation metrics on the DT classifier. For the KNN classifier, NS-PPO shows a significant advantage in AUC compared to the second-ranked LTRUS-ratio, but it only demonstrates a slight lead in F-measure and MCC values. For the RF classifier, NS-PPO exhibits significant advantages in AUC and F-measure compared to the baseline methods. However, its advantage in MCC over LTRUS is marginal and does not exhibit statistical significance.

Based on the results, the performance of cluster is limited, primarily due to its reliance on the K-means algorithm to generate cluster centers. However, K-means often struggles to accurately capture the feature distribution of software instances, leading to suboptimal clustering results. The RUS is also suboptimal, as random sampling may discard important features, resulting in a loss of data representativeness. In contrast, the Borderline and Smote-ENN methods exhibit improved effectiveness, as they are capable of generating high-quality samples. However, they may still introduce additional noise into the synthesized samples. While Smote-ENN uses the ENN algorithm to clean noise, its effectiveness remains constrained. LTRUS and LTRUS-ratio achieve the best performance compared to other baseline methods. While both methods incorporate objective functions to optimize undersampling strategies, their focus is limited to the undersampling process. Recent research by Yang *et al.* [40] suggest that effective oversampling techniques outperform undersampling techniques when addressing software instances. NS-PPO builds on this by utilizing a DRL-based undersampler in its second phase to filter out noisy samples, leading to superior performance compared to LTRUS and LTRUS-ratio.

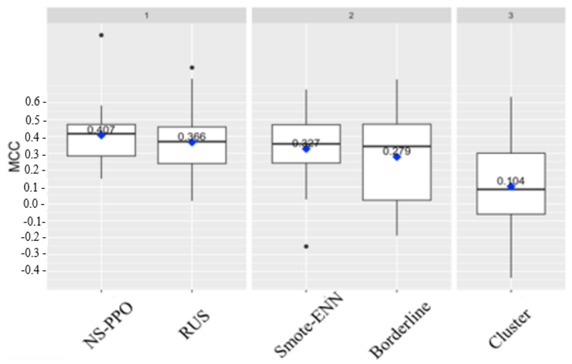
- **Finding 1**: when handling expert metrics-based software instances, NS-PPO demonstrates a significant advantage over baseline methods on DT and RF, while showing a slight advantage on KNN.

5.2. Answer to RQ2

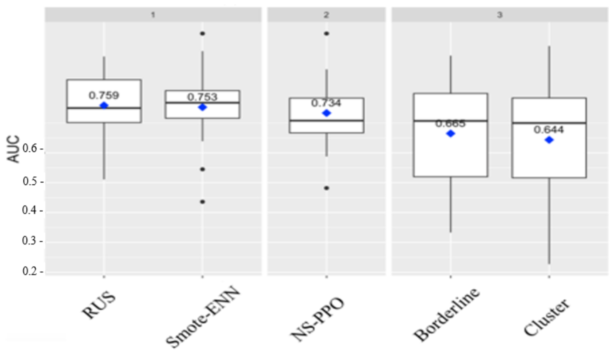
RQ2 evaluates the impact of NS-PPO and baseline methods on semantic features-based SDP methods. Since LTRUS and LTRUS-ratio depend on expert metrics for ranking tasks, they are excluded from the comparison in RQ2. Furthermore, due to the lack of semantic content in the NASA dataset, this evaluation is conducted on 15 software projects from the PROMISE dataset. Figure 8 presents the comparative performance of NS-PPO and baseline methods integrated into the ICNN, accompanied by the Scott-Knott ESD test results. Figure 9 reports the corresponding results based on the TSE.



a) F-measure values on ICNN.

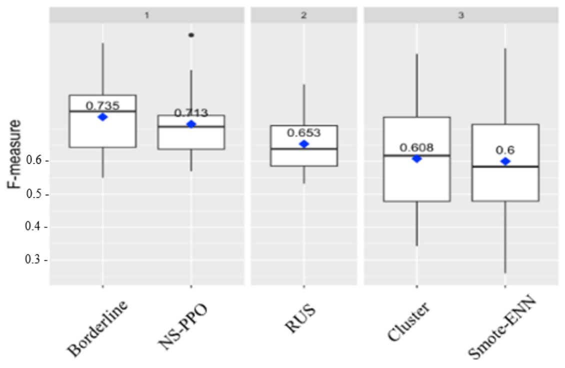


b) MCC values on KNN.

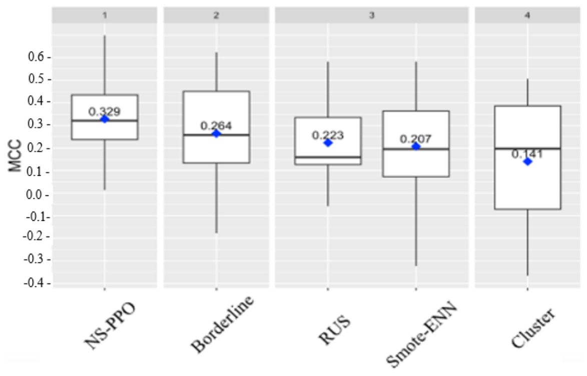


c) AUC values on ICNN.

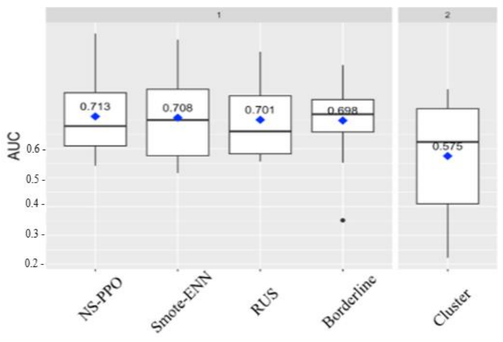
Figure 8. Performance and ESD test results of the NS-PPO and baseline methods on ICNN.



a) F-measure values on TSE.



b) MCC values on TSE.



c) AUC values on TSE.

Figure 9. Performance and ESD test results of the NS-PPO and baseline methods on TSE.

As shown in Figure 8, NS-PPO achieves the highest average F-measure value, surpassing Borderline by 0.6%, RUS by 5.2%, SMOTE-ENN by 8.9%, and Cluster by 33.3%. In terms of MCC values, NS-PPO outperforms RUS by 4.1%, SMOTE-ENN by 8%, borderline by 12.8%, and Cluster by 30.3%. However, NS-PPO shows a slight disadvantage in the AUC, where it fails behind RUS and SMOTE-ENN. As shown in Figure 9, NS-PPO achieves the highest average MCC value with a significant advantage, outperforming borderline by 6%, RUS by 10.2%, SMOTE-ENN by 11.4%, and cluster by 18.5%. In terms of AUC values, NS-PPO surpasses SMOTE-ENN by 0.8%, RUS by 1.4%, borderline by 2%, and cluster by 13.2%. However, NS-PPO lags behind borderline by 2.3% in the F-measure value but exhibits a significant advantage over other methods.

It is noteworthy that NS-PPO achieves the highest MCC values on both ICNN and TSE. A plausible explanation is that the undersampler in the second stage constructs the reward function based on the MCC values of the SDP classifier on the validation set. As a result, the final performance shows strong performance in terms of MCC, although this may come at the expense of other evaluation metrics. Furthermore, recent SDP studies have indicated that AUC and F-measure are biased, whereas MCC is unbiased [11, 13]. Therefore, this paper places greater emphasis on the impact of NS-PPO on the MCC values of the SDP methods.

- **Finding 2**: compared to the baseline methods, NS-PPO demonstrates a clear advantage in handling semantic features-based software instances.

5.3. Answer to RQ3

NS-PPO consists of two stages. In the first stage, a hybrid sampler based on NCL and SMOTE is employed to generate a large number of samples for the minority class. In the second stage, a DRL-based undersampler is designed to select high-quality samples. RQ3 aims to evaluate the effectiveness of each step within the NS-PPO, determining their individual contributions to the overall performance. Specifically, RQ3 explores the impact of NS-PPO$_{\text{-DRL}}$ (NS-PPO without the second stage) and NS-PPO$_{\text{-NCL}}$ (NS-PPO without the NCL in the first stage) on the performance of DT and RF, which excelled in RQ1, as well as the ICNN and TSE in RQ2.

Table 5 presents the impact of different components within NS-PPO on prediction performance. Bolded values indicate the optimal strategy for each SDP method. For expert metrics-based SDP methods, NS-PPO achieve the highest MCC and AUC values on the RF classifier, while falling slightly behind NS-PPO-DRL by 0.2% in F-measure. On the DT classifier, NS-PPO demonstrates the superior performance across all evaluation metrics. For semantic metrics-based SDP methods, NS-PPO achieves the optimal performance across all evaluation metrics on the ICNN. On the TSE, NS-PPO exhibits the optimal performance in F-measure and MCC but lags behind NS-PPO$_{\text{-NCL}}$ in AUC.

Table 5. Impact of different components in NS-PPO on prediction performance.

SDP methods	Imbalance handling methods	F-measure	MCC	AUC
RF	NS-PPO	0.580	0.433	0.835
	NS-PPO$_{\text{-DRL}}$	0.582	0.426	0.813
	NS-PPO$_{\text{-NCL}}$	0.507	0.364	0.820
DT	NS-PPO	0.539	0.355	0.694
	NS-PPO$_{\text{-DRL}}$	0.534	0.352	0.686
	NS-PPO$_{\text{-NCL}}$	0.462	0.274	0.648
ICNN	NS-PPO	0.760	0.407	0.759
	NS-PPO$_{\text{-DRL}}$	0.730	0.271	0.660
	NS-PPO$_{\text{-NCL}}$	0.742	0.253	0.742
TSE	NS-PPO	0.713	0.329	0.713
	NS-PPO$_{\text{-DRL}}$	0.700	0.304	0.694
	NS-PPO$_{\text{-NCL}}$	0.707	0.318	0.734

The experimental results clearly demonstrate that NS-PPO achieves the highest MCC values across all SDP methods. However, it falls slightly behind the compared methods in F-measure for RF and AUC for TSE.

The main reason for this phenomenon is likely consistent with the explanation in RQ2: the second stage of NS-PPO focuses on optimizing the MCC value, which may come at the expense of other evaluation metrics. Additionally, the results show that NS-PPO$_{\text{-DRL}}$ outperforms NS-PPO$_{\text{-NCL}}$. This can be attributed to the class overlap issue in the software instances. Superior performance is achieved by first mitigating the class overlap and then applying oversampling techniques. In contrast, several SDP studies have overlooked this and applied oversampling techniques without mitigating the overlap issue [2, 10].

- **Finding 3**: each component of NS-PPO is essential and has a positive impact on prediction performance.

6. Conclusions

Previous SDP methods depend on static data resampling techniques, which may produce synthetic samples that deviate from the original feature distribution. Moreover, serval SDP imbalance-handling methods focus exclusively on expert metric-based features, restricting their applicability to semantic features. To address these limitations, this paper proposes NS-PPO, a two-stage data resampling method specifically designed for SDP tasks. In the first stage, NS-PPO constructs a hybrid sampler based on NCL and SMOTE to generate a large number of synthetic samples for the minority class. In the second stage, NS-PPO treats the undersampling process for synthetic samples as a MDP, building a DRL-based undersampler that selects high-quality samples based on the training performance. The experimental results show that NS-PPO has a significant advantage over baseline methods, including LTRUS, across three key evaluation metrics: MCC, F-measure, and AUC. Additionally, this paper demonstrates the effectiveness of NS-PPO in handling semantic features.

In the future, we plan to extend NS-PPO to C/C++-based datasets. Additionally, we aim to explore dynamic cleaning techniques for the class overlap issue in software instances and investigate generative adversarial network-based oversampling methods to improve the quality of synthetic samples.

Data Availability Statement

The supplementary materials associated with this paper can be obtained from the corresponding author upon reasonable request.

Acknowledgment

This work is supported by Guangdong Natural Science Fund Project (Grant No. 2024A1515010204), and China

Southern Power Grid Company Limited (Grant No. ZBKJXM20232483).

References

[1] Agrawal A. and Menzies T., ""Better Data" is Better than "Better Data Miners" (Benefits of Tuning SMOTE for Defect Prediction)," *in Proceedings of the 40th International Conference on Software Engineering*, Gothenburg, pp. 1050-1061, 2018. https://doi.org/10.1145/3180155.3180197

[2] Bahaweres R., Agustian F., Hermadi I., Suroso A., and Arkeman Y., "Software Defect Prediction Using Neural Network Based SMOTE," *in Proceedings of the 7th International Conference on Electrical Engineering, Computer Sciences and Informatics*, Yogyakarta, pp. 71-76, 2020. DOI:10.23919/EECSI50503.2020.9251874

[3] Batista G., Prati R., and Monard M., "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20-29, 2004. https://doi.org/10.1145/1007730.1007735

[4] Bennin K., Keung J., and Monden A., "On the Relative Value of Data Resampling Approaches for Software Defect Prediction," *Empirical Software Engineering*, vol. 24, pp. 602-636, 2019. https://doi.org/10.1007/s10664-018-9633-6

[5] Breiman L., "Random Forests," *Machine Learning*, vol. 45, pp. 5-32, 2001. https://doi.org/10.1023/A:1010933404324

[6] Chen L., Fang B., Shang Z., and Tang Y., "Tackling Class Overlap and Imbalance Problems in Software Defect Prediction," *Software Quality Journal*, vol. 26, pp. 97-125, 2018. https://doi.org/10.1007/s11219-016-9342-6

[7] Cover T. and Hart P., "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, 1967. DOI:10.1109/TIT.1967.1053964

[8] Czibula G., Marian Z., and Czibula I., "Software Defect Prediction Using Relational Association Rule Mining," *Information Sciences*, vol. 264, pp. 260-278, 2014. https://doi.org/10.1016/j.ins.2013.12.031

[9] Ding H., Chen L., Dong L., Fu Z., and Cui X., "Imbalanced Data Classification: A KNN and Generative Adversarial Networks-Based Hybrid Approach for Intrusion Detection," *Future Generation Computer Systems*, vol. 131, pp. 240-254, 2022. https://doi.org/10.1016/j.future.2022.01.026

[10] Dipa W. and Sunindyo W., "Software Defect Prediction Using SMOTE and Artificial Neural Network," *in Proceedings of the International Conference on Data and Software Engineering*, Bandung, pp. 1-4, 2021. DOI:10.1109/ICoDSE53690.2021.9648476

[11] Feng S., Keung J., Xiao Y., Zhang P., Yu X., and Cao X., "Improving the Undersampling Technique by Optimizing the Termination Condition for Software Defect Prediction," *Expert Systems with Applications*, vol. 235, pp. 121084, 2024. https://doi.org/10.1016/j.eswa.2023.121084

[12] Feng S., Keung J., Yu X., Xiao Y., Bennin K., Kabir M., and Zhang M., "COSTE: Complexity-Based Oversampling Technique to Alleviate the Class Imbalance Problem in Software Defect Prediction," *Information and Software Technology*, vol. 129, pp. 106432, 2021. https://doi.org/10.1016/j.infsof.2020.106432

[13] Gong L., Jiang S., Wang R., and Jiang L., "Empirical Evaluation of the Impact of Class Overlap on Software Defect Prediction," *in Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, San Diego, pp. 698-709, 2019. DOI:10.1109/ASE.2019.00071

[14] Goyal S., "Handling Class-Imbalance with KNN (Neighbourhood) Under-Sampling for Software Defect Prediction," *Artificial Intelligence Review*, vol. 55, pp. 2023-2064, 2022. https://doi.org/10.1007/s10462-021-10044-w

[15] Gupta M., Rajnish K., and Bhattacharjee V., "Software Fault Prediction with Imbalanced Datasets Using SMOTE-Tomek Sampling Technique and Genetic Algorithm Models," *Multimedia Tools and Applications Journal*, vol. 83, pp. 47627-47648, 2024. https://doi.org/10.1007/s11042-023-16788-7

[16] Han H., Wang W., and Mao B., "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning," *in Proceedings of the Advances in Intelligent Computing Conference*, Hefei, pp. 878-887, 2005. https://doi.org/10.1007/11538059_91

[17] Jiang S., Chen Y., He Z., Shang Y., and Ma L., "Cross-Project Defect Prediction Via Semantic and Syntactic Encoding," *Empirical Software Engineering*, vol. 29, no. 4, pp. 80, 2024. https://doi.org/10.1007/s10664-024-10495-z

[18] Jureczko M. and Madeyski L., "Towards Identifying Software Project Clusters with Regard to Defect Prediction," *in Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, Timisoara, pp. 1-10, 2010. https://doi.org/10.1145/1868328.1868342

[19] Khleel N. and Nehez K., "A Novel Approach for Software Defect Prediction Using CNN and GRU Based on SMOTE Tomek Method," *Journal of Intelligent Information Systems*, vol. 60, no. 3, pp. 673-707, 2023. https://doi.org/10.1007/s10844-023-00793-1

[20] Khleel N. and Nehez K., "Software Defect Prediction Using a Bidirectional LSTM Network

Combined with Oversampling Techniques," *Cluster Computing*, vol. 27, pp. 3615-3638, 2024. https://doi.org/10.1007/s10586-023-04170-z

[21] Khoshgoftaar T. and Gao K., "Feature Selection with Imbalanced Data for Software Defect Prediction," *in Proceedings of the International Conference on Machine Learning and Applications*, Miami, pp. 235-240, 2009. DOI:10.1109/ICMLA.2009.18

[22] Kim D. and Chung Y., "Addressing Class Imbalances in Software Defect Detection," *Journal of Computer Information Systems*, vol. 64, no. 2, pp. 219-231, 2024. https://doi.org/10.1080/08874417.2023.2187483

[23] Laurikkala J., "Improving Identification of Difficult Small Classes by Balancing Class Distribution," *in Proceedings of the 8th Conference on Artificial Intelligence in Medicine in Europe*, Cascais, pp. 63-66, 2001. https://doi.org/10.1007/3-540-48229-6_9

[24] Li J., He P., Zhu J., and Lyu M., "Software Defect Prediction via Convolutional Neural Network," *in Proceedings of the IEEE International Conference on Software Quality, Reliability and Security*, Prague, pp. 318-328, 2017. DOI:10.1109/QRS.2017.42

[25] Liu J., Ai J., Lu M., Wang J., and Shi H., "Semantic Feature Learning for Software Defect Prediction from Source Code and External Knowledge," *Journal of Systems and Software*, vol. 204, pp. 111753, 2023. https://doi.org/10.1016/j.jss.2023.111753

[26] Liu Y., Sun F., Yang J., and Zhou D., "Software Defect Prediction Model Based on Improved BP Neural Network," *in Proceedings of the 6th International Conference on Dependable Systems and their Applications*, Harbin, pp. 521-522, 2020. DOI:10.1109/DSA.2019.00095

[27] Ma Y., Luo G., Zeng X., and Chen A., "Transfer Learning for Cross-Company Software Defect Prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248-256, 2012. https://doi.org/10.1016/j.infsof.2011.09.007

[28] Nam J. and Kim S., "Heterogeneous Defect Prediction," *in Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, Bergamo, pp. 508-519, 2015. https://doi.org/10.1145/2786805.2786814

[29] Nazarudin N., Ariffin N., and Maskat R., "Leveraging on Synthetic Data Generation Techniques to Train Machine Learning Models for Tenaga Nasional Berhad Stock Price Movement Prediction," *The International Arab Journal of Information Technology*, vol. 21, no. 3, pp. 483-494, 2024. https://doi.org/10.34028/iajit/21/3/11

[30] Okutan A. and Yildiz O., "Software Defect Prediction Using Bayesian Networks," *Empirical Software Engineering*, vol. 19, pp. 154-181, 2014. https://doi.org/10.1007/s10664-012-9218-8

[31] Pan C., Lu M., Xu B., and Gao H., "An Improved CNN Model for Within-Project Software Defect Prediction," *Applied Sciences*, vol. 9, no. 10, pp. 1-28, 2019. https://doi.org/10.3390/app9102138

[32] Pedregosa F., Varoquaux G., Gramfort A., Michel V., and et al., "Scikit-Learn: Machine Learning in Python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011. https://dl.acm.org/doi/10.5555/1953048.2078195

[33] Quinlan J., "Induction of Decision Trees," *Machine Learning*, vol. 1, pp. 81-106, 1986. https://doi.org/10.1007/BF00116251

[34] Schulman J., Wolski F., Dhariwal P., Radford A., and Klimov O., "Proximal Policy Optimization Algorithms," *arXiv Preprint*, vol. arXiv:1707.06347, pp. 1-12, 2017. https://doi.org/10.48550/arXiv.1707.06347

[35] Tantithamthavorn C., Mclntosh S., Hassan A., and Matsumoto K., "An Empirical Comparison of Model Validation Techniques for Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1-18, 2017. DOI:10.1109/TSE.2016.2584050

[36] Wang X., Lu L., Tian Q., and Lin H., "IC-GraF: An Improved Clustering with Graph-Embedding-Based Features for Software Defect Prediction," *IET Software*, vol. 2024, pp. 1-22, 2024. https://doi.org/10.1049/2024/8027037

[37] Yan K., Lu C., Ma X., Ji Z., and Huang J., "Intelligent Fault Diagnosis for Air Handing Units Based on Improved Generative Adversarial Network and Deep Reinforcement Learning," *Expert Systems with Applications*, vol. 240, pp. 122545, 2024. https://doi.org/10.1016/j.eswa.2023.122545

[38] Yang F., Zhong F., Zeng G., Xiao P., and Zheng W., "LineFlowDP: A Deep Learning-Based Two-Phase Approach for Line-Level Defect Prediction," *Empirical Software Engineering*, vol. 29, no. 2, pp. 50, 2024. https://doi.org/10.1007/s10664-023-10439-z

[39] Yang P., Zhu L., Zhang Y., Ma C., Liu L., Yu X., and Hu W., "On the Relative Value of Clustering Techniques for Unsupervised Effort-Aware Defect Prediction," *Expert Systems with Applications*, vol. 245, pp. 123041, 2024. https://doi.org/10.1016/j.eswa.2023.123041

[40] Yang X., Wang S., Li Y., and Wang S., "Does Data Sampling Improve Deep Learning-Based Vulnerability Detection? Yeas! and Nays!," *in Proceedings of the IEEE/ACM 45th International Conference on Software Engineering*, Melbourne, pp. 2287-2298, 2023. DOI:10.1109/ICSE48619.2023.00192

[41] Yu X., Liu L., Zhu L., Keung J., Wang Z., and Li F., "A Multi-Objective Effort-Aware Defect Prediction Approach Based on NSGA-II," *Applied*

Soft Computing, vol. 149, pp. 110941, 2023. https://doi.org/10.1016/j.asoc.2023.110941

[42] Zhou Y., Lu L., Zou Q., and Li C., "Two-Stage AST Encoding for Software Defect Prediction," *in Proceedings of the 34th International Conference on Software Engineering and Knowledge Engineering*, Pittsburgh, pp. 1-4, 2022. DOI:10.18293/SEKE2022-039

Xiaowei Zhao he is currently working on his Ph.D. dissertation in the School of Computer Science and Engineering, South China University of Technology. His research interests include Digital Transformation Development, Software Defect Detection, and High-Performance Computing Optimization.

Xuanye Wang he is currently working on his Ph.D. dissertation in the School of Computer Science and Engineering, South China University of Technology. His research interests include Intelligent Software Engineering and Software Defect Prediction.

Siliang Suo received the master degree from Huazhong University of Science and Technology, Wuhan, China, in 2007. His research interests include Power Artificial Intelligence Technology, Power Communication, and Security Protection of New Power Systems.

Lu Lu he received the Ph.D. degree from Xi'an Jiaotong University, Xian, China, in 1999. He is currently a Professor with the School of Computer Science and Engineering, South China University of Technology. His research interests include Software Engineering, Software Testing, and Software Architecture Design.