Hindawi
IET Software
Volume 2023, Article ID 6293074, 30 pages
https://doi.org/10.1049/2023/6293074

Research Article

Evaluating the Impact of Data Transformation Techniques on the Performance and Interpretability of Software Defect Prediction Models

Yu Zhao,[1] **Zhiqiu Huang**,[1] **Lina Gong**,[1] **Yi Zhu**,[2] **Qiao Yu**,[2] **and Yuxiang Gao**[2]

[1]*School of Computer Science and Technology, Key Laboratory of Safety-Critical Software, Nanjing University of Aeronautics and Astronautics, Nanjing 210000, China*
[2]*School of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221116, China*

Correspondence should be addressed to Zhiqiu Huang; zqhuang@nuaa.edu.cn

Received 2 June 2023; Revised 7 October 2023; Accepted 20 October 2023; Published 14 November 2023

Academic Editor: Nadeem Sarwar

The performance of software defect prediction (SDP) models determines the priority of test resource allocation. Researchers also use interpretability techniques to gain empirical knowledge about software quality from SDP models. However, SDP methods designed in the past research rarely consider the impact of data transformation methods, simple but commonly used preprocessing techniques, on the performance and interpretability of SDP models. Therefore, in this paper, we investigate the impact of three data transformation methods (Log, Minmax, and Z-score) on the performance and interpretability of SDP models. Through empirical research on (i) six classification techniques (random forest, decision tree, logistic regression, Naive Bayes, K-nearest neighbors, and multilayer perceptron), (ii) six performance evaluation indicators (Accuracy, Precision, Recall, F1, MCC, and AUC), (iii) two interpretable methods (permutation and SHAP), (iv) two feature importance measures (Top-k feature rank overlap and difference), and (v) three datasets (Promise, Relink, and AEEEM), our results show that the data transformation methods can significantly improve the performance of the SDP models and greatly affect the variation of the most important features. Specifically, the impact of data transformation methods on the performance and interpretability of SDP models depends on the classification techniques and evaluation indicators. We observe that log transformation improves NB model performance by 7%–61% on the other five indicators with a 5% drop in Precision. Minmax and Z-score transformation improves NB model performance by 2%–9% across all indicators. However, all three transformation methods lead to substantial changes in the Top-5 important feature ranks, with differences exceeding 2 in 40%–80% of cases (detailed results available in the main content). Based on our findings, we recommend that (1) considering the impact of data transformation methods on model performance and interpretability when designing SDP approaches as transformations can improve model accuracy, and potentially obscure important features, which lead to challenges in interpretation, (2) conducting comparative experiments with and without the transformations to validate the effectiveness of proposed methods which are designed to improve the prediction performance, and (3) tracking changes in the most important features before and after applying data transformation methods to ensure precise and traceable interpretability conclusions to gain insights. Our study reminds researchers and practitioners of the need for comprehensive considerations even when using other similar simple data processing methods.

1. Introduction

Recently, intelligent software development methods have received extensive attention from researchers in the software engineering community. Software defect prediction (SDP) research is an important part of intelligent software development methods and an important application scenario of Artificial Intelligence for Software Engineering (AI4SE) [1, 2]. Software defect prediction combines machine learning with defect datasets to train models, predicting defects in the target software modules [3–5]. In the past, much research has been dedicated to enhancing the performance of SDP models. This aims to assist testers in effectively allocating their limited testing resources and crafting efficient, resource-saving software quality assurance (SQA) plans [6, 7]. However, recent researches [8–10] pointed out that improving the prediction performance of SDP models to formulate SQA is only one of the multiple goals of software defect prediction. The SDP model should also have the capability to offer practitioners profound insights. This includes comprehending the features, especially the most crucial ones, associated with past software defects. Additionally, it should provide an explanation for the rationale behind a specific prediction made by the SDP model. Then that can help the software development team to design software development activities scientifically.

The performance of the SDP model is directly contingent on the quality of the training dataset. Consequently, in order to enhance the model's performance, for the phenomenon of non-normal data distribution and dimensional differences in the training datasets, researchers use various simple data transformation methods to eliminate the dimensional differences of the data or incline the data to normal distribution to eliminate the distribution difference of data [11, 12]. Recent research on defect prediction has examined the influence of data transformation methods on the performance of SDP models. Menzies et al. [13] observed an enhancement in the performance of the Naive Bayes (NB) model when applying the log transformation method to NASA datasets. However, Adnan et al. [14] experiments on Just-In-Time (JIT) datasets and Hao et al. [15] on Eclipse datasets yielded contrasting results, showing that the log transformation method not only failed to improve the AUC value of the NB model but even led to a slight decrease. These conflicting findings in the previous studies make it challenging to establish definitive guidelines regarding the application of data transformation methods in preprocessing training datasets for model construction. In particular, it is unknown, which indicators of which SDP models will be affected by the data transformation method.

When we review the previous studies on building interpretable SDP models, we find that most of the studies did not consider the impact of data transformation methods on model interpretability [16–18]. Recent studies point out that the performance and interpretability of defect prediction models are the inverse effects of each other [19]. The improvement of model performance may cause a decline in interpretation, and the improvement of interpretation will cause a decline in performance. Currently, the data transformation method has become a conventional processing step in the field of defect prediction research. However, it remains uncertain whether certain existing research endeavors, focused on developing interpretable defect prediction model methods and enhancing the performance of SDP models [20–22], have incorporated data transformation methods or directly analyzed the original datasets. Consider that previous empirical studies on the impact of data transformation methods on SDP model were conducted on problematic datasets (e.g., NASA dataset) and insufficient classifiers and evaluation metrics. This also threatens the validity and reproducibility of existing work. Therefore, it is necessary to explore the impact of data transformation methods on the performance and interpretability of SDP models.

In this paper, we set out to investigate the impact of three commonly used data transformation methods (i.e., log transformation, Minmax transformation, and Z-score transformation) on the performance and interpretability of SDP models. We use six commonly used classification techniques to train SDP models, including Random Forest (RF), Logistic Regression (LR), Naive Bayes (NB), Decision Tree (DT), K-Nearest Neighbors (KNN), and Multilayer Perceptron (MLP). We use six commonly used performance evaluation indicators, including Accuracy, Precision, Recall, F1, MCC, and AUC, to evaluate the performance of the SDP model built on the datasets before and after applying the data transformation method. We use Top-k feature rank overlap and Top-k feature rank difference to evaluate the change of the most important features of SDP models built on the datasets before and after applying the data transformation method. Through an empirical study of 16 software projects in three open-source software datasets, we draw the following conclusions.

Compared to the SDP model constructed using the original datasets, SDP models constructed using datasets transformed by log transformation, Minmax transformation, and Z-score transformation can yield significant improvements in the performance evaluation metrics for most models. Even in cases where certain classification models or evaluation metrics do not show significant improvement, it will hardly cause a decline in model performance (details in Section 4.1).

Compared to the SDP model constructed using the original datasets, the SDP models constructed using datasets transformed by log transformation, Minmax transformation, and Z-score transformation exhibit notable changes in feature importance for most models. This shift in feature importance directly impacts the interpretability of the SDP models. In other words, whether the application of data transformation methods affects the interpretability of SDP models depends on the classification techniques utilized in the study (details in Section 4.2).

There is indeed a tradeoff between the performance and interpretability of SDP models. The decision to employ data transformation methods for preprocessing the datasets before constructing the SDP model should be based on a careful analysis of the specific goals (details in Section 5).

Our contributions include:

(i) We revisit whether data transformation methods can improve the performance of SDP models. Using six classification techniques on a more extensive and reasonable dataset to conduct qualitative analysis, quantitative analysis, and significance analysis on whether the three data transformation methods can improve the six performance evaluation indicators of the SDP model.

(ii) We conduct an empirical study on whether and how the use of three data transformation methods affects the importance rank of the most important features of SDP models.

(iii) We discuss the relationship between defect prediction model performance and interpretability and give suggestions on whether data transformation methods should be used for data preprocessing in defect datasets according to the different goals of defect prediction.

(iv) We release a repository containing experimental data, experimental code, and experimental processes to enhance the persuasiveness and generalizability of experimental conclusions and promote replicable and sustainable research in the community (https://github.com/OpenSELab/IET-Software-2023).

The rest of this paper is organized as follows.

Section 2 introduces related work and research motivations. Section 3 presents the detailed experimental design that addresses the questions posed. Section 4 analyzes the experimental results from a holistic and partial perspective and answers the two research questions posed. In Section 5, we give suggestions based on the conclusions of experimental results according to the different objectives of defect prediction. Section 6 analyzes the threats to the validity of the research conclusions. Finally, we conclude our study in Section 7.

2. Related Work and Research Motivation

Agrawal and Menzies [23] emphasized the significance of data preprocessing as a crucial step in the experiments. In the design of defect prediction methods, data preprocessing is more important than classifier selection. Data transformation is a commonly employed technique for data preprocessing. Numerous prior studies have introduced diverse methods aimed at enhancing the performance of SDP models, with some studies explicitly outlining the steps involved in the data transformation. For example, Liu et al. [24] proposed a two-phase transfer learning model for cross-project defect prediction, which clearly pointed out that they used the log transformation method to process training datasets in the data preprocessing stage. Zhou et al. [25] and Li et al. [26] also clearly stated that they used the Z-score transformation method to standardize the feature values in the datasets before building the SDP model.

However, much more research on defect prediction do not specify whether data transformation methods are used or not and what kind of data transformation methods are used for data preprocessing in detail. For example, Wei et al. [27] proposed the SDP model with effective dimension reduction. Wu et al. [28] proposed a cost-sensitive kernelized semi-supervised dictionary learning defect prediction method. The above methods, including some other similar SDP methods [29–31], do not indicate whether they used data transformation methods to preprocess the datasets and which method they used to preprocess the dataset, or whether their methods directly experimented on the original datasets. Since, it is not clear how effective the data transformation method is in improving the performance of the model. Therefore, the repeatability of the previous experiments and the validity of experimental conclusions may pose threats. That is to say, if the data transformation method can significantly improve the performance of the SDP model, it is difficult to know what factors are responsible for the improvement of the performance of the SDP model by some existing methods.

In past research on SDP models using data transformation methods, Menzies et al. [13] found that building NB models on NASA datasets transformed by the log transformation can significantly improve performance. However, Jiang et al. [32] found that the log transformation rarely improved the performance of the SDP model on NASA datasets. The experimental results of Amin et al. [14] on JIT datasets and Jia et al. [15] on Eclipse datasets are also contrary to the conclusions of Menzies et al. [13]. They found that the log transformation method not only failed to improve the AUC of the NB model but even decreased slightly. These studies produced inconsistent and even conflicting conclusions, which made it difficult to derive practical guidelines on whether data transformation methods should be applied to dataset preprocessing and building SDP models. Furthermore, previous empirical studies on the impact of data transformation methods on the performance of SDP models were analyzed on the problematic datasets (e.g., NASA datasets were proved to have noise data [33], and other datasets were also unbalanced) and on fewer than three performance evaluation indicators (e.g., some only selected one single AUC indicator). This threatens the validity of previous research conclusions. Because the conclusions may not generalize to the other datasets and other indicators. If we want to build an SDP model on a new dataset and hope to get better AUC and F1, we cannot get effective guidance from the conclusions of existing works. Therefore, in order to solve the inconsistency and effectiveness threats of the previous studies, we propose the first research question:

RQ1: How do the data transformation methods affect the performance of the defect prediction model?

Recently, many research focused on the interpretability of SDP models. Jiarpakdee et al. [34] pointed out that the three goals of defect prediction technology are equally important. That is, it is equally important for developers to understand the features related to the past software defects and explain why a specific file is predicted to be defective as well as to predict whether a file will be defective in the future. Wan et al. [35] conducted a quantitative and qualitative study on the value of defect prediction in practice. They conducted interviews and surveys on practitioners' thoughts, behaviors, and expectations, and found that more than 90% of respondents were willing to adopt defect prediction techniques.

The interpretability of the defect prediction model makes the prediction results of the model easier to understand and trust. When decision-makers or stakeholders can understand the basis for the model's judgments, they are more likely to accept and trust the model's predictions. Additionally, interpretability can provide important insights into why a particular prediction was made. This helps developers quickly locate and fix defects in software, rather than relying

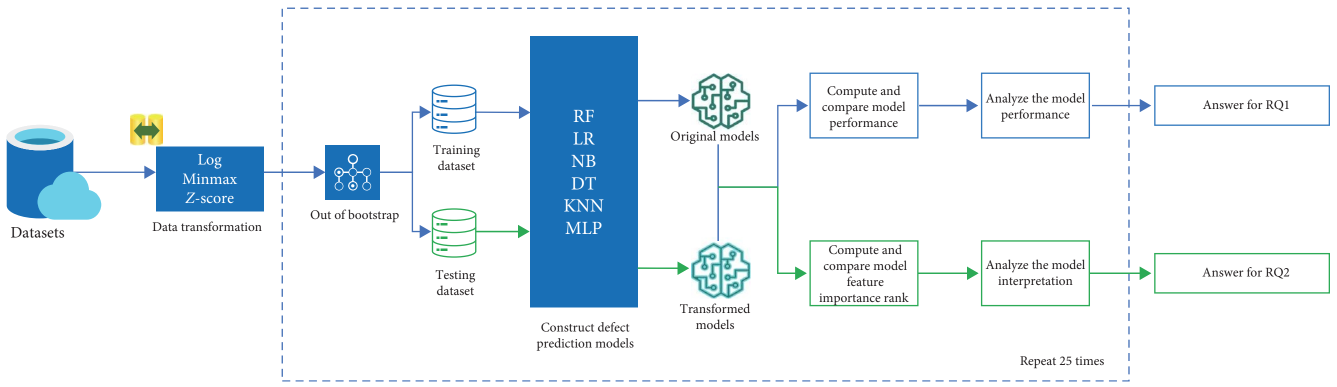

Datasets
Log
Minmax
Z-score
Data transformation
Out of bootstrap
Training dataset
Testing dataset
RF
LR
NB
DT
KNN
MLP
Construct defect prediction models
Original models
Transformed models
Compute and compare model performance
Compute and compare model feature importance rank
Analyze the model performance
Analyze the model interpretation
Answer for RQ1
Answer for RQ2
Repeat 25 times

Figure 1: Overview of case study design. (Transformed models mean the models built using the datasets transformed by the data transformation methods, original models mean the models built using the datasets without transformed).

solely on the model predictions. However, the existing defect prediction technology does not give an explanation for making a certain prediction, which hinders practitioners from applying defect prediction technology in practice.

Many researchers start from feature importance to study the interpretability of SDP models. They use model-specific feature importance methods to gain insights from SDP models. For example, the analysis of variance (ANOVA) method of the LR model is used to analyze the defect prediction model [36, 37], and the feature importance method of the RF model [38, 39] and DT model [40, 41] is used to acquire useful knowledge for the development process. Similarly, researchers also use model-agnostic methods to study the most important features that contribute to the classification of SDP models. For example, Qiu et al. [22] use the partial dependence plots (PDP) method to calculate the feature importance of the RF model for the experimental comparison. Esteves et al. [42] use the SHapley Additive exPlanations (SHAP) method to interpret the predictions of their designed defect model. Jiarpakdee et al. [43] use local interpretable model-agnostic explanations (LIME), breakdown, and their proposed method to analyze the most contributing features of prediction for a given instance.

Similar to the research on defect prediction performance, it is also unclear whether some existing works investigating the interpretability of SDP models applied data transformation methods or directly performed on the original datasets. For example, Tantithamthavorn et al. [44] pointed out clearly that the log transformation is used to process the original datasets when studying the impact of mislabeling on the interpretability of the SDP model. However, some of the above research on the interpretability of SDP models did not indicate whether they were empirical studies on the datasets after data transformation. Data transformation appears to be a conventional data preprocessing step but its impact on interpretability is understudied. Therefore, this phenomenon may pose threats to the validity and repeatability of the conclusions of the existing interpretability work, because it is not yet clear whether the data transformation method will affect the interpretability of the SDP model. So when repeating the previous work, whether to use data transformation to process the dataset may lead to conclusions that are inconsistent with the previous work. If the existing research uses the data transformation method to deal with the datasets, it is more necessary to further explore what is the reason for the conclusion. In addition, recent work pointed out that there is a certain balance between the performance and interpretability of the SDP model [22]. That is, the improvement of performance may cause a decrease in interpretability, and the enhancement of interpretability will reduce the performance of the model. Therefore, if the data transformation method affects the performance of the SDP model, it may also affect the interpretability of the model. So, we propose the second research question:

RQ2: How do data transformation methods affect the interpretability of defect prediction models?

To address the above research questions, we use six comprehensive performance evaluation metrics and two feature importance metrics to study the impact of three data transformation methods on the performance and interpretability of six commonly used SDP classification models on 16 software projects. The next section will introduce the technology involved in detail.

3. Case Study Design

In this section, we describe the case study designed to address the above two research questions, and the framework of the entire case design is shown in Figure 1. For each part of the framework, we will introduce in following subsections in detail, including data transformation methods, classification techniques, datasets, feature importance methods, evaluation indicators, and experimental implementation, etc.

3.1. Data Transformation Methods. Appropriate transformation of data has become one of the basic steps before the construction of defect prediction model. This paper studies three data transformation methods commonly used by researchers in the past. We first choose the log transformation, which is a method used by the previous researchers that led to inconsistent conclusions as mentioned earlier [13, 14, 16, 32, 45]. Then we choose Minmax transformation and *Z*-score transformation, which are two data transformation methods frequently used by researchers in the field of defect prediction [6, 46–48]. Below, we briefly introduce these three data transformation methods.

Log transformation [49] is a mathematical technique that converts the original data values of software features into their corresponding logarithmic values. This method is widely employed in the field of software defect prediction. Before building the SDP model, the dataset is preprocessed to make the data distribution closer to the normal distribution. Log transformation is essentially a mathematical function transformation. Due to the rules of mathematical functions, it can only process values greater than 0. However, in practical applications, it is common to add a constant to the value being transformed to handle 0 values [50]. Therefore, Formula (1) is generally used for log transformation in practice.

$$X_{\text{new}} = \text{In}(X+1). \tag{1}$$

Minmax transformation [48] is a linear data transformation method that scales the original data to a range between 0 and 1. Max and Min are the maximum and minimum values of a feature column in the dataset, respectively. The objective is to convert the original feature values into dimensionless pure values, enabling comparison and weighting of features with the different units or magnitudes. This helps eliminate the influence of dimensional differences on the final results. We use Formula (2) to perform Minmax transformation on the dataset.

$$X_{\text{new}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}. \tag{2}$$

Z-score transformation [51], also known as standard score transformation, is a technique that performs standardization based on the mean and standard deviation of the original data. It ensures that the processed feature values have a mean of 0 and a standard deviation of 1. The transformation calculates the standard deviation between each original feature value and the mean value. This results in relative values that remove dimensional effects while preserving the relationship between the data points. We use Formula (3) to perform *Z*-score transformation.

$$\frac{X_{\text{new}} = X - \overline{X}}{\sigma}. \tag{3}$$

In order to feel the difference in data transformation more intuitively, we draw a diagram to display the transformed data and the original data. It can be clearly seen from Figure 2 that Minmax transformation and *Z*-score transformation can retain the distribution information of the original data and only transform the value of the data; log transformation not only changes the value of the data but also transforms the original data into a normal distribution. These three transformation methods can be automatically called and executed in the scikit-learn machine learning package and the Weka tool platform.

3.2. Datasets. In all publicly available datasets, NASA datasets have been commonly used by the researchers in the field of defect prediction over the past few years [52–54]. However, recent studies have shown that there are noisy data in NASA datasets [33, 55], and the cleared NASA datasets still have quality problems [56, 57]. Therefore, we did not use the NASA datasets, because datasets with quality problems may threaten the validity of the experimental conclusions. We selected some software projects of three datasets, Promise, Relink, and AEEEM. The Promise dataset is provided by Jureckzo and Madeyski [58], each project in the Promise is characterized by 20 different metrics. These 20 metrics all focus on the code complexity. These metrics take into account the inherent characteristics of object-oriented programs such as encapsulation, inheritance, and polymorphism. The Relink dataset is provided by Wu et al. [59], which contains 26 features, which can be divided into two categories: complexity indicators and count indicators. These indicators are usually defined from the perspective of code complexity such as line of codes, cycloramic complexity, number of classes and abstract syntax trees such as number of blocks, number of statements, and method references. And the AEEEM dataset is provided by D'Ambros et al. [60], each project in AEEEM contains 61 features. Among them, 17 features are related to source code, 5 features are related to previous predictions, 5 features are related to code change entropy, 17 features are related to source code entropy, and 17 features are related to source code decay. To the best of our knowledge, these three datasets do not expose quality issues, and they contain a large number of software projects of the different types and domains and are widely used by the researchers in the field of defect prediction. At the same time, these three datasets are also available as open source. From the selected datasets, we excluded certain software projects that exhibited highly imbalanced classes. This decision was based on a previous research, which has demonstrated that building a defect prediction model on imbalanced datasets can introduce biases [61, 62]. Additionally, using class rebalancing techniques to address the imbalance can impact the performance and interpretability of the final model [63, 64]. Therefore, these projects were excluded to ensure the integrity and fairness of the analysis. The summary information of the 16 software projects used in the final experiment is shown in Table 1.

3.3. Out-of-Sample Bootstrap Validation. Tantithamthavorn et al. [65] conducted an empirical study of the various model validation techniques. They found that the holdout family validation method is consistently the most biased and least stable model validation technique and cross-validation techniques can only produce stable performance on part models and datasets. Irrespective of the type of classifier and datasets, the out-of-sample bootstrap validation produced the best balance between estimated bias and variance compared to holdout validation and cross-validation techniques and even if the validation experiment is repeated, this conclusion remains unchanged. Therefore, in order to ensure that the conclusions we draw about the defect prediction model are reliable, we also use the out-of-sample bootstrap validation [66, 67].

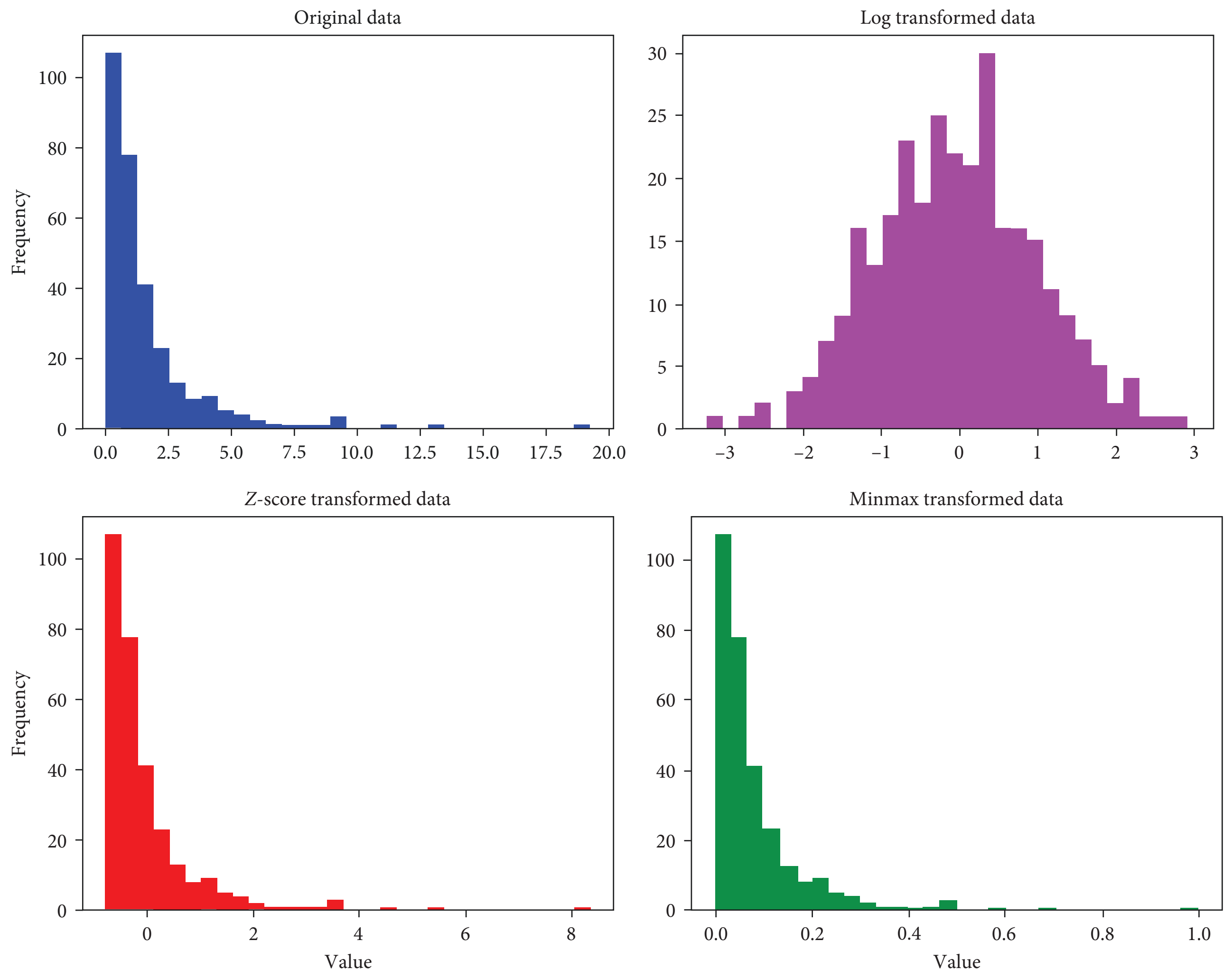


FIGURE 2: Illustration of transformed data vs. original data.

TABLE 1: Experimental datasets.

Project	Features	Modules	Defects	Ratio (%)
Camel-1.2	20	608	216	36
Ivy-1.1	20	111	63	57
Jedit-3.2	20	272	90	33
Log4j-1.1	20	109	37	34
Lucene-2.0	20	195	91	47
Lucene-2.2	20	247	144	58
Lucene-2.4	20	340	203	60
Poi-1.5	20	237	141	59
Poi-2.5	20	385	248	64
Poi-3.0	20	442	281	64
Xalan-2.5	20	803	387	48
Xalan-2.6	20	885	411	46
Apache	26	194	98	51
Safe	26	56	22	39
ZXing	26	399	118	30
EQ	61	325	129	40

The basic principles and steps are as follows. If dataset D contains d instances, then the goal is to sample D to produce dataset D^* with the same number of instances and the same distribution of data. Then we select an instance from dataset D each time and put the instance back into dataset D so that the instance may still be selected in the next selection. This process is repeated for d times, and finally, dataset D^* with the same size as dataset D is generated. In the whole instance selection process, 36.8% of the instances in dataset D are actually not selected into D^*. Therefore, we use the sampled dataset D^* as the training set, and 36.8% of the instances in dataset D that do not appear in D^* as the testing set. The above process is repeated 25 times.

3.4. Classification Techniques. In the field of software defect prediction, there are many machine learning classifiers used to build SDP models. It is difficult for us to study all classification techniques because there are too many types of classifiers. Therefore, we selected six classifiers of different types that are frequently used by researchers, including NB, LR, DT, RF, KNN, and MLP.

NB is a simple and efficient classification model based on Bayes' theorem and feature condition independence assumption. Although the characteristics of the SDP dataset do not satisfy the conditional independence assumption, it still achieves good defect prediction performance.

KNN is a classification model based on the distance measure of instances. The basic idea is that an instance should belong to the same category as most of the instances of its nearest neighbors in the feature space.

LR introduces sigmoid transformation on the basis of linear regression to map continuous values into probability values. New instances are classified by training the parameter matrix and bias values of the features based on instances of known categories.

The essence of DT is to summarize a set of classification rules from the training dataset by calculating the information gain of features and using the rules to classify a new instance.

RF is an ensemble learning model based on DTs, and its final classification of a new instance is obtained by voting from all DTs.

MLP is a feedforward artificial neural network model, which usually consists of three parts: input layer, hidden layer, and output layer, and has good nonlinear fitting ability. New instances are classified by training the parameters of the hidden layer on the training set.

The above six models are widely used in the field of defect prediction due to their simplicity, convenience, and good performance. However, it is worth noting that the RF and MLP require more time to train. These six classifiers take into account both quantity and type and can draw general conclusions from the perspective of the model. Similar to the data transformation method, these six models can be automatically called and executed in the scikit-learn machine learning package and the Weka platform.

3.5. Feature Importance

3.5.1. Model-Agnostic Feature Importance Methods. Model-agnostic feature importance methods possess the ability to calculate the importance of features without requiring knowledge about the specific model's details, such as the neural network structure. These methods treat the model to be explained as a black-box and determine feature importance accordingly. The key advantage of model-agnostic feature importance methods lies in their flexibility, as they can provide explanations for any type of model. This characteristic makes them particularly well-suited for the six different model types selected in this paper. While some of the classifiers chosen in this paper may have their own model-specific methods for calculating feature importance, previous research [68] has shown inconsistencies in determining the most important features when using such model-specific methods. Hence, we decided not to adopt the model-specific feature importance method. Additionally, Rajbahadur et al. [68] highlighted the risk of relying solely on one particular feature importance method, which could compromise the validity of the results. Consequently, we opt for two model-agnostic methods to calculate the feature importance of the SDP model. Since, it is uncertain whether removing feature interactions affects the impact of data transformation methods on model performance. Therefore, we do not remove the correlation and interaction between features in the datasets [10].

We choose permutation [69, 70] and SHAP [71, 72] to calculate the characteristic importance of the SDP model. First, permutation is one of the oldest and most commonly utilized methods in the software engineering communities [18, 73]. Second, SHAP is a recently developed global feature importance method, which is theoretically proven to generate optimal feature importance rankings, whose adoption is growing within the software engineering community [41, 71, 74]. Finally, previous research [68] has found that both of these two methods do not require additional steps for hyperparameter optimization and demonstrate high consistency in computing feature importance. However, the current popular LIME [43, 75] benefits from a complex hyperparameter optimization process, which will affect the validity of the experimental results. Moreover, both permutation and SHAP are not affected by feature interactions and correlations.

3.5.2. Generate Feature Importance Scores and Ranks. The permutation feature importance method [70] measures feature importance by calculating the increase in model prediction error after randomly permuting a column of feature values. A feature is considered important if permuting its values increases the model error. Because in this case, the model relies on this feature to make predictions. A feature is considered unimportant if the model error remains unchanged after permuting its values. Because in this case, the model will ignore the feature. For this reason, we randomly arrange each feature column on the test data of each out-of-sample bootstrap validation and set the attenuation value of the AUC predicted by the model before and after the arrangement as the importance score of the feature. Since the arrangement of the feature column values is random, in order to reduce the randomness, we perform 25 random processes and take the average value of the attenuation value to obtain the importance score of each feature.

The SHAP feature importance method [71] is a method to explain individual prediction. The goal is to explain the prediction of an instance by calculating the contribution of each feature to the prediction. This contribution is described as the Shapley value in SHAP, and features with large absolute Shapley values are more important. We can use this value as the importance score of each feature for each instance prediction. To calculate the global feature importance score, we average the absolute Shapley values of each feature on the test data for each out-of-sample bootstrap validation to obtain the global feature importance score for each feature.

After calculating the importance scores of each feature using permutation and SHAP, we rank these feature importance scores and corresponding features. Since the value of the feature importance score calculated by the two methods is larger, the feature is more important, so we can sort it in descending order. If there are multiple features with the same importance score, then those features are ranked the same. For example, if the importance scores of the three features are ($f1 = 0.2, f2 = 0.2, f3 = 0.1$), then their ranks are ($f1 = 1, f2 = 1, f3 = 2$). Note that when using permutation and SHAP, there is a problem of computational efficiency. The advantage of SHAP is that, it allows for direct calculation of feature importance scores. However, it is important to notice that the computational complexity of SHAP increases exponentially with the number of features. Although permutation has a relatively low-calculation complexity when compared with SHAP, it depends on the evaluation indicators to get the feature importance score, which requires increased computational time to repeat the experiment multiple times thus avoiding randomness in the importance scores.

3.6. Evaluation Indicators

3.6.1. Performance Evaluation Indicators. Software defect prediction is a classification problem, and there are many binary performance evaluation indicators available in the machine learning field. Although we hope to obtain a more comprehensive and objective evaluation of the model, it is time-consuming and laborious to use all classification indicators to evaluate the performance of the SDP model. Agrawal and Menzies [23] pointed out that there are many evaluation criteria. Therefore, it is not necessary to use all evaluation indicators when evaluating the performance of the SDP model, but more than one evaluation indicator should be used. And the indicators used should be those widely used by researchers in the field, rather than other evaluation indicators that have not been widely accepted. Following this recommendation, we utilize six evaluation indicators that are widely used by researchers, including Accuracy, Precision, Recall, F1, MCC, and AUC.

There are four situations when making decisions in the defect prediction model, namely TP, TN, FN, and FP. Among them, TP and TN are the cases where the prediction is correct, indicating that defective instances are predicted to be defective and non-defective instances are predicted to be nondefective, respectively. FP and FN are two cases of the prediction errors, which, respectively, indicate that defect-free instances are predicted to be defective and defective instances are predicted to be defect-free. The evaluation indicators of this paper are derived from these four situations. Accuracy, Precision, and Recall are three commonly used evaluation indicators. Accuracy is the simplest and most intuitive evaluation indicator, which measures the proportion of correctly classified instances to the total number of instances. Precision and Recall correspond to the capabilities of model precision and recall, respectively, but these two indicators are conflicting. F1 is calculated by the harmonic mean of Precision and Recall. It comprehensively considers the overall performance of the model on Recall and Precision, which can fully reflect the actual performance of the model. AUC is the coverage area of the ROC curve, which is a threshold-independent performance indicator used to measure the ability of a classifier to distinguish between defective modules and clean modules and is widely used in the field of software defect prediction. Recently, MCC has also been widely used by the researchers [76–78]. MCC also considers the four situations that appear in the actual prediction, i.e., TP, TN, FN, and FP, which is a more balanced indicator. The closer these evaluation indicators are to 1, the better the performance of the model. Among them, Accuracy, Precision, Recall, F1, and MCC can be expressed by the following Formulas (4)–(8).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{4}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{5}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{6}$$

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{7}$$

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}. \tag{8}$$

3.6.2. Top-k Feature Rank Overlap. Top-*k* feature rank overlap [68, 79] is a straightforward indicator that calculates the ratio of the number of Top-*k* feature rank overlaps to the total number in all feature rank lists. This indicator does not take into account the order of features in the Top-*k* feature rank list. Instead, it simply checks whether a given feature appears in the Top-*k* feature rank of all feature rank lists. In this paper, we calculate the feature rank overlap of Top-1, Top-3, and Top-5. The following example illustrates the calculation method of this indicator. After getting two feature rank lists l1 and l2, if the Top-3 feature of the l1 is ($f1 = 1$, $f2 = 2$, $f3 = 3$), and the Top-3 feature of the l2 is ($f4 = 1$, $f1 = 2$, $f5 = 3$), then the Top-3 feature rank overlap is 1/5, and the Top-1 feature rank overlap is 0.

In order to make the Top-*k* feature rank overlap easier to understand, the result of the Top-*k* feature rank overlap is defined as consistency, and we adopt a consistency interpretation scheme defined in previous research [68], which is shown in Formulas 9 and 10. For Top-1 feature rank overlap, if the Top-1 feature rank overlap value is less than or equal to 0.5, the consistency is considered as “low”; otherwise, the consistency is considered as “high”. For Top-3 and Top-5 feature rank overlap, if the value of the Top-3 or Top-5 feature rank overlap is greater than or equal to 0 and less than or equal to 0.25, the consistency is considered to be “negligible”. If the value of the Top-3 or Top-5 feature rank overlap is greater than 0.25 and less than or equal to 0.5, the consistency is considered to be “small”. If the value of the Top-3 or Top-5 feature rank overlap is greater than 0.5 and less than or equal to 0.75, the consistency is considered to be “moderate”. If the value of the Top-3 or Top-5 feature rank overlap is greater than 0.75 and less than or equal to 1, the consistency is considered to be “large”.

$$\text{Top}-1 \text{ feature rank overlap consistency} = \begin{cases} \text{low}, & 0.00 \leq \text{consistency} \leq 0.50 \\ & \\ \text{high}, & 0.50 < \text{consistency} \leq 1.00 \end{cases}, \quad (9)$$

$$\text{Top}-3,5 \text{ feature rank overlap consistency} = \begin{cases} \text{negligible}, & 0.00 \leq \text{consistency} \leq 0.25 \\ \text{small}, & 0.25 < \text{consistency} \leq 0.50 \\ \text{moderate}, & 0.50 < \text{consistency} \leq 0.75 \\ \text{large}, & 0.75 < \text{consistency} \leq 1.00 \end{cases}. \quad (10)$$

3.6.3. Top-k Feature Rank Difference. The consistency of the most important features of the SDP model built before and after applying the data transformation method can be obtained through the results of the Top-k feature rank overlap, but the specific changes in the rank of the most important features cannot be displayed in detail. Therefore, we also compute the Top-k feature rank difference. The Top-k feature rank difference [63, 64] is defined as the absolute value of the rank offset Top-k feature in the different feature rank lists. In this paper, we calculate the rank difference of the most important top five features, i.e., $k = 1, 2, 3, 4$, and 5. The following example illustrates the calculation method of this indicator. After getting two feature rank lists l1 and l2, if the Top-3 feature of the l1 is ($f1 = 1, f2 = 2, f3 = 3$), and the Top-3 feature of the l2 is ($f4 = 1, f1 = 2, f5 = 3$), then the Top-1 feature rank difference is $|2–1| = 1$.

The larger the value of the Top-k feature rank difference, the greater the change in the most important feature. For example, in the above case, rfc was originally the Top-1 feature. But after the transformation method, rfc became the Top-2 feature on the SDP model constructed by the transformed dataset. The rfc has a rank difference of 1. Top-1 to Top-2 does not particularly affect the importance of rfc, but if the original rfc rank is 1, and then the rank becomes 7, then it means that rfc is not important to the latter. Therefore, for the Top-k feature rank difference, we believe that if the difference value exceeds 2, indicating that the importance of the feature changes significantly.

3.7. Experimental Implementation. All experiments involved in this paper were conducted on the Jupyter notebook running on a machine with the following configuration: Intel(R) Xeon(R) CPU E5-2620 v4 @2.10GHz RAM 64G. We only need the CPU to perform experiments, which means repeated experiments are computationally cheap. All experimental codes rely on Python language implementation. Among all the steps, steps such as data processing, model training, performance collection, and feature importance calculation rely on machine-learning packages such as scikit-learn, pandas, numpy, ELI5, and SHAP.

As explained before, when performing out-of-sample bootstrap validation, 36.8% of the instances in a whole dataset are not sampled, therefore the remaining 36.8% serve as the test dataset and the sampled instances constitute the training dataset. The ratio of an entire dataset divided into training and testing is roughly close to 6 : 4. It is clear that the experimental scenario actually belongs to within-project defect prediction. In order to ensure the replicability of data segmentation, we set the random seed to 0–24 and save the corresponding data. Then, we apply three data transformation methods on the saved training data and test data to obtain the transformed dataset and save it to the hard disk.

To ensure consistency during model training and replicability of training, we set the random state of the model to 0 and set other parameters to default values when we train the models. In this way, training the same model on the same dataset will get the same internal parameters, and thus we will get consistent performance when testing.

When collecting performance, we rely on the scikit-learn package, and when collecting feature importance scores, we rely on the ELI5 and SHAP packages. The calculation of performance values and feature importance scores starts on the test dataset after the model is trained on each training set. This operation ensures that changes in performance values and feature importance scores are caused by transformation operations on the dataset rather than being caused by the model inconsistencies.

4. Case Study Results

This section introduces the details and results of the experimental research. Through the following experimental results, we answer the above two research questions.

4.1. How Do the Data Transformation Methods Affect the Performance of the Defect Prediction Model?

4.1.1. Approach. We first use six classification techniques (i.e., RF, LR, NB, DT, KNN, and MLP) to build SDP models (i.e., original model) on the 16 software projects such as Camel, Ivy, and Jedit, in the three datasets introduced in Section 3.2 (i.e., Promise, AEEEM, and Relink). We then apply the three data transformation methods of Log, Minmax, and Z-score to these datasets and use the same classification techniques to build SDP models (i.e., transformed model). Finally, we calculate the values of six performance evaluation indicators (i.e., Accuracy, Precision, Recall, F1, MCC, and AUC) of the original model and transformed model. Note that in the experiment, we utilize out-of-sample bootstrap validation. This involves generating 25 groups of performance values for each classification model and evaluation indicator across all datasets. Subsequently, we calculated the average performance values from these groups.

We experimentally investigate the impact of data transformation methods on model performance from two perspectives. The first is to calculate the absolute value of the differences between the transformed models and the original models in all six performance evaluation indicators from the overall point of view. Due to a large number of performance indicators, the overall perspective can provide the overall impact of data transformation methods on each performance evaluation indicator. The second is to calculate the

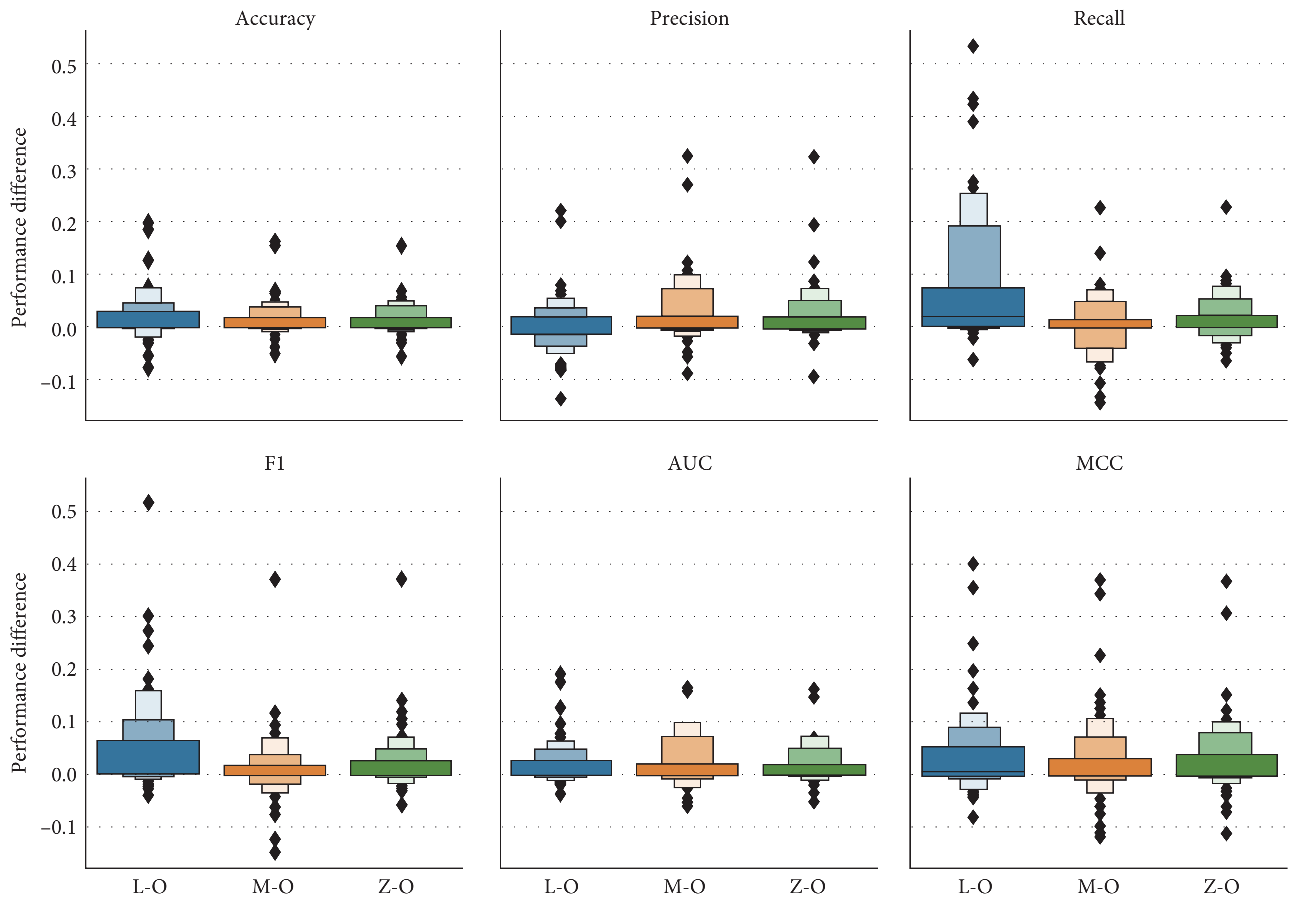


Figure 3: The overall performance differences on all six performance evaluation indicators across all datasets between all transformed models and original models. E.g., Abscissa L-O represents the difference between the performance indicators of the SDP model constructed by the transformed datasets using the log transformation method minus the performance indicators of the SDP model constructed by the original datasets.

comparison of the performance evaluation indicators of each transformed model and original model from the partial point of view. From the perspective of comparing the performance indicators of each classification model, it can be clarified which transformation method has an impact on a certain classification model on which performance evaluation indicators.

Result Overall, the Log transformation method can significantly improve Accuracy, Recall, F1, AUC, and MCC. The Minmax transformation method can significantly improve Accuracy, Precision, AUC, and MCC. The Z-score transformation method can significantly improve all six performance evaluation indicators. Therefore, the data transformation methods can generally improve the performance of SDP models.

From Figure 3, we can see that the log transformation method improves the Recall, F1, and MCC of the SDP model at a relatively high level. In these indicators, the performance difference is generally between 0.1 and 0.2, and there are still many performance differences exceeding 0.3. It can also improve the performance of Accuracy and AUC since the performance difference is generally within 0.1, and some are around 0.2. However, the log transformation method cannot improve the Precision of the SDP mode because the performance difference is basically evenly distributed between positive and negative values. Therefore, using log transformation to process datasets in the SDP field can significantly improve the performance of multiple indicators of the SDP model.

The Minmax transformation method can improve the Precision and MCC of the SDP model effectively. Since the performance difference is generally within 0.1, and some differences are around 0.3. However, its improvement in the Accuracy and AUC is relatively weak, and no improvement in Recall and F1 because the performance difference in Accuracy and AUC is generally within 0.05 and the performance difference in Recall and F1 is basically distributed between positive and negative values. Therefore, using Minmax transformation to process datasets in the SDP field can improve the performance of the SDP model, but it is not stable enough.

The performance differences between the *Z*-score transformed model and the original model are within 0.1 on each indicator. This shows that *Z*-score transformation can slightly improve the performance of the SDP model.

Table 2: The p-values of the Wilcoxon signed-rank test on all six performance evaluation indicators across all datasets between the transformed models and original models.

Wilcoxon p-values	L vs. O	M vs. O	Z vs. O
Accuracy	$p<0.01$	$p<0.01$	$p<0.01$
Precision	$p>0.05$	$p<0.01$	$p<0.01$
Recall	$p<0.01$	$p>0.05$	$p<0.05$
F1	$p<0.01$	$p>0.05$	$p<0.01$
AUC	$p<0.01$	$p<0.05$	$p<0.01$
MCC	$p<0.01$	$p<0.01$	$p<0.01$

In order to verify whether there is a significant difference in the distribution of the performance evaluation indicators obtained by the original model and transformed model. We performed the Wilcoxcon signed rank test ($\alpha=0.05$) on six evaluation indicators for all the classification models. The p-values of the Wilcoxcon signed rank test are shown in Table 2. The results of the significance test align with the findings of the performance analysis, supporting the validity, and accuracy of the previous analysis. Moreover, the significance test reveals that the log transformation significantly enhances the Accuracy, Recall, F1, AUC, and MCC, while the Minmax transformation improves Accuracy, Precision, AUC, and MCC, and the Z-score transformation shows significant improvement across all six indicators. These improvements are not random fluctuations, but rather meaningful enhancements validated by the statistical significance.

From the overall perspective, we may miss some detailed information about the impact of the transformation method on the performance of the SDP model. For example, we conclude that the data transformation method can significantly improve the performance of the SDP model. However, we cannot get useful information about the classification model when designing the experiment of defect prediction methods. That is to say, whether the data transformation method can play the role of improving most performance indicators on all classification models. For this reason, we calculate the comparison of the performance evaluation indicators of each transformed model and the original model from a partial perspective. We want to know which classification model and which indicator these data transformation methods can improve.

The Log transformation method significantly enhances the performance of the SDP model, particularly in LR, NB, and KNN models.

From Figure 4 and Table 3, we found that for the SDP model built by the log transformed datasets, compared with the SDP model built by the original datasets, the RF model only has a slight improvement by 0.6% in Recall but without significance, a weak significant decline in Precision by 0.3% and remain unchanged in other four indicators. Therefore, log transformation cannot effectively improve the performance of the SDP model constructed by RF. The situation is similar to the DT model and the MLP model. The log transformed DT model and log transformed MLP model stay unchanged in Precision and improve the performance on the other five indicators, with the average performance increased by 1%, 5%, 3%, 2%, 8%, and 2%, 7%, 5%, 2%, 8%, respectively. However, the test results in Table 3 show that the DT model only has a significant increase of 2% on AUC, the MLP model only has a significant increase of 5% on Recall and 2% on F1, and the other indicators have no significant increase. The log transformed LR model and log transformed KNN model improve the performance on all six indicators, with the average performance increased by 5%, 4%, 7%, 7%, 5%, 22%, and 3%, 3%, 7%, 5%, 4%, 17%, respectively. However, the test results in Table 3 show that the 4% improvement of the LR model in Precision is not significant. Therefore, log transformed LR model and log transformed KNN model can significantly improve various performance evaluation indicators, except for Precision. Finally, Figure 4 shows that the performance of the log transformed NB model has decreased (5%) in Precision, while the performance of the other five indicators has improved by 7%, 61%, 31%, 7%, and 26%, respectively. But we can see that the 7% improvement of the NB model on Accuracy is not significant. Therefore, the log transformed NB model can significantly increase Recall by 61%, F1 by 31%, AUC by 7%, and MCC by 26%, but will cause a decrease of Precision by 5%.

For the Minmax transformation, in Figure 4 and Table 3, we can see that the performance values of the Minmax transformed RF model on all six indicators remain unchanged. The same situation occurs in the LR model, DT model and the KNN model. The Minmax transformed LR model only has a significant improvement of 3% in Accuracy. The Minmax transformed DT model only significantly improves 0.9% of AUC and the Minmax transformed KNN model only significantly improves in Precision. Therefore, Minmax transformation cannot effectively improve the performance of the SDP model constructed by RF, LR, DT, and KNN. For the Minmax transformation, in Figure 4 and Table 3, we can see that the performance values of the Minmax transformed RF model on all six indicators remain unchanged. The same situation occurs in the LR model, DT model, and the KNN model. The Minmax transformed LR model only has a significant improvement of 3% in Accuracy. The Minmax transformed DT model only significantly improves 0.9% of AUC and the Minmax transformed KNN model only significantly improves in Precision. Therefore, Minmax transformation cannot effectively improve the performance of the SDP model constructed by RF, LR, DT, and KNN. In contrast, both the Minmax transformed NB model and the Minmax transformed MLP model get performance improvements on all six indicators. Although the results of the significance test show that the improvement of the Minmax transformed NB model in Precision and the Minmax transformed MLP model in Recall is not significant. But the Minmax transformed NB model and Minmax transformed MLP model are still the two models with the most significant improvement effects, improving the remaining five indicators by 2%, 4%, 5%, 2%, 9%, and 4%, 4%, 4%, 3%, 13%, respectively.

The Minmax transformation method significantly improves the performance of the NB model

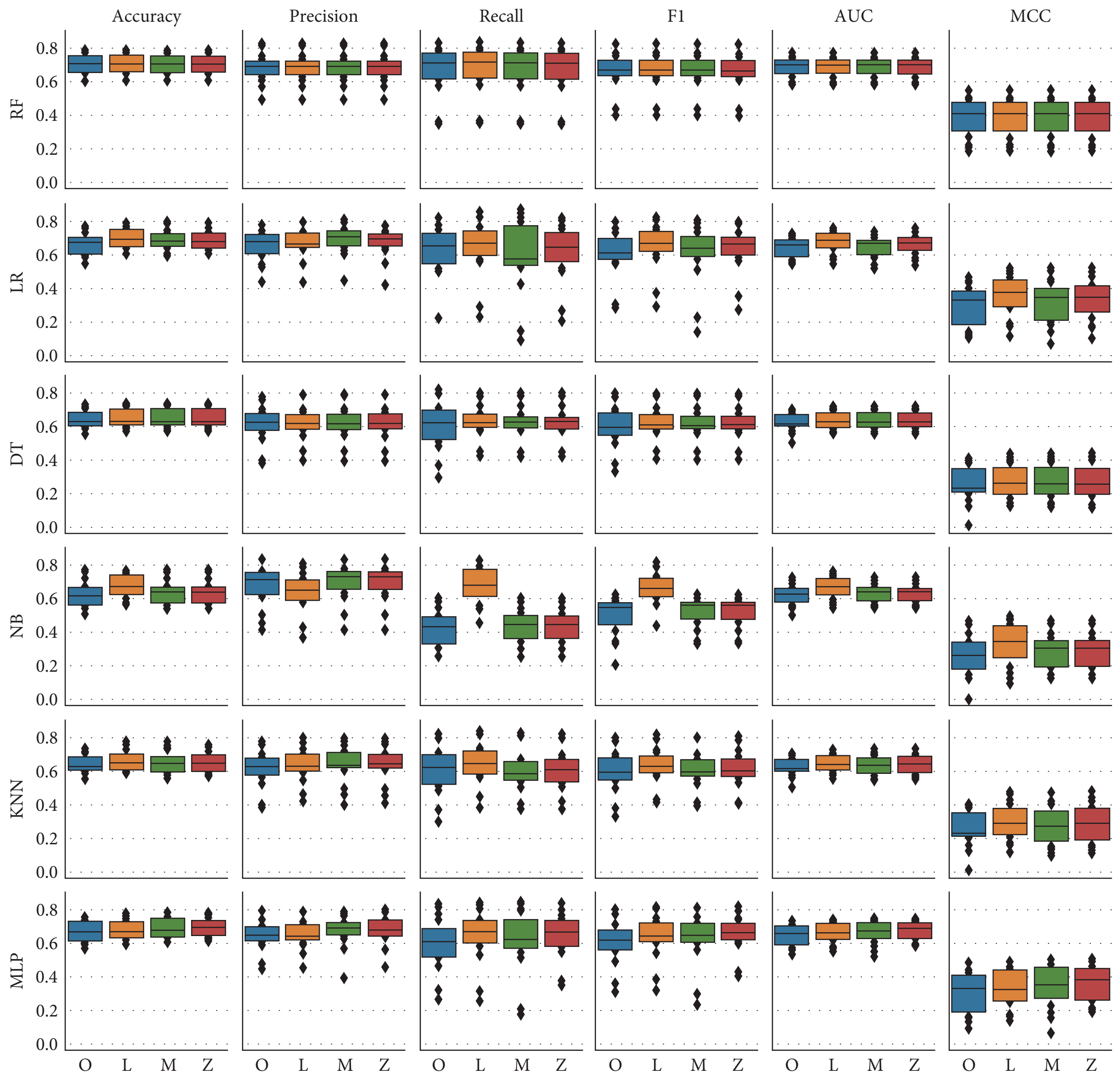


Figure 4: The classifier performance comparison on all six performance evaluation indicators between transformed models and original models. E.g., Abscissa O and L, respectively, represent the performance of the defect prediction model built on the original dataset and the datasets transformed using the log transformation method.

and MLP model in the transformed dataset, while having little impact on other models.

Finally, the SDP model built on the dataset transformed by the *Z*-score transformation method in Figure 4 is similar to the Minmax transformation method. Compared with the original RF model, the performance values of the *Z*-score transformed RF model on all six indicators remain unchanged. The *Z*-score transformed DT model only has a significant improvement of 3% on AUC. The *Z*-score transformed KNN model only has a significant improvement of 4% in Precision and has no significant improvement on the other indicators. *Z*-score transformed LR model, *Z*-score transformed NB model, and *Z*-score transformed MLP model have significant improvements on many indicators. *Z*-score transformed LR model has significantly improved by 3%, 4%, 4%, 3%, and 15% on the other five indicators except for Recall. *Z*-score transformed NB model has significantly improved by 2%, 4%, 5%, 2%, and 9% on the other five indicators except for Precision. *Z*-score transformed MLP model has significantly improved by 4%, 4%, 7%, 7%, 4%, and 17% on all six indicators.

The Z-score transformation method brings significant and consistent improvements to the performance of SDP models created with the LR, NB, *and MLP algorithms while having minimal impact on other models.*

4.2. *How Do Data Transformation Methods Affect the Interpretability of Defect Prediction Models?*

4.2.1. Approach. We continue experiments on original model and transformed model obtained in Section 4.1. For each model, we use the permutation and SHAP importance methods to calculate the model's feature importance score and rank. Then we compute the Top-k feature rank overlap ($k = 1, 3, 5$) and the Top-k feature rank difference ($k = 1, 2, 3, 4, 5$). Finally, we still analyze the changes in the importance of model features from the perspectives of the whole and parts to measure the changes in the interpretability of the model.

Result Overall, the utilization of any data transformation method significantly affects the feature importance in the SDP model. This variation can make it challenging to interpret the model accurately.

Ideally, all peaks of the density distribution of the Top-k feature overlap should be between 0.75 and 1, and the values of the highest peak and the median are all around 1.

For log transformation, the highest peak distribution of Top-1 overlap is concentrated around 0, and the median of the distribution of Top-1 overlap is 0. This shows that the consistency of the most important feature overlap after applying the log transformation method is negligible, which means that the most important features have changed basically. The distribution of the highest peak of Top-3 overlap is concentrated around 0.5. Two of the three peaks are less than or equal to 0.5, and the median of the distribution is 0.5. This shows that after the log transformation method is applied, the consistency of the Top-3 feature rank overlap is small, which also indicates that the most important first three features have basically changed. The highest peak distribution of Top-5 overlap is concentrated around 0.25, the highest two peaks of the four peaks are concentrated below 0.5, and the median of the distribution is less than 0.5, which indicates that Top-5 important features are inconsistent in at least half of the experiments.

For Minmax transformation, similar to the above, the distribution of the highest peak of Top-1 overlap is concentrated around 0, and the median of the distribution is 0, which indicates that the most important features have basically changed after the Minmax transformation method was applied. The distribution of Top-3 overlap and Top-5 overlap is similar. Although the highest peak value is distributed around 1, the distribution between 0 and 0.5 is the main body of the distribution. The median of the Top-3 overlap is 0.5, while the median of the Top-5 overlap is less than 0.5. This shows that the consistency of the Top-3 and Top-5 important features is small.

For Z-score transformation, the distribution of Top-3 overlap and Top-5 overlap is similar. It is also similar to the distribution of the Top-3 overlap and Top-5 overlap of Minmax transformation, and the median is the same. Therefore, for Z-score transformation, the consistency of the Top-3 and Top-5 important features overlap is also small. However, the distribution of Top-1 overlap is different from the above two transformation methods. The highest peak distribution of Top-1 overlap is concentrated around 1, and the median of the distribution is 1, so the most important feature overlap of Z-score transformation has a high degree of consistency. However, since the value of Top-1 overlap is basically 0 or 1, the peak value at the value of 0 is also not low. By investigating the raw data we found that the most important features were inconsistent in more than 40% of the experiments.

Applying Log transformation and Minmax transformation decreases the consistency of the Top-1, Top-3, and Top-5 features between the Original Models and Transformed Models. Additionally, applying Z-score transformation reduces the consistency of the Top-3 and Top-5 features between the Original Models and Transformed Models. This has a significant impact on changes in feature importance.

The Top-k features rank overlap can only measure the strength of consistency. In order to observe the changes in the Top-k feature ranks more clearly, we display the differences in Top-k feature ranks through bar graphs.

From Figure 5(a), we observe that only about 49%, 49%, and 54% of the top features remain the most important features and about 34%, 44%, and 38% of the top features differ by more than 2 if the data transformation method is used on the dataset. This means that for the three data transformation methods, at least half of the top features of the Transformed SDP Model have changed, and more than 30% of the original Top-1 features are no longer important features of Top-3. Similarly, for features with ranks of 2, 3, and 4, only about 35%–45% of them remain unchanged, and about 30%–40% of feature importance ranks differ by more than 2. For features with a rank of 5, only about 40% of them remain in the same rank at most, while about 40% of them have a difference of more than 2 in importance rank. This means that in about 40% of the experiments, the rank of Top-5 important features has a large difference, such as slipping from the first rank to the fourth rank and slipping from the second rank to the fifth rank.

In order to make the conclusion more generalizable, we also use SHAP to calculate the feature importance rank overlap and feature importance rank difference. We can find that the density distribution of feature overlap shown in Figure 6(b) is similar to Figure 6(a), and the bar chart distribution of feature differences shown in Figure 5(b) is also similar to Figure 5(a). This shows that the Top-k features calculated by using the SHAP on the Transformed SDP Model also have low feature rank overlap consistency and large feature rank differences.

Applying Log transformation, Minmax transformation, and Z-score transformation causes the difference in the Top-5 feature importance ranks

Table 3: The p-values of the Wilcoxon signed-rank test and average percentage performance gains on all six performance evaluation indicators between the transformed models and original models built with different classifiers.

p-Values and percentage	O vs. L		O vs. M		O vs. Z	
RF-Accuracy	p > 0.05	0	p > 0.05	0	p > 0.05	0
RF-Precision	p > 0.05	−0.3%	p > 0.05	0	p > 0.05	0
RF-Recall	p < 0.01	0.7%	p > 0.05	0	p > 0.05	0
RF-F1	p > 0.05	0.2%	p > 0.05	0	p > 0.05	0
RF-AUC	p > 0.05	0	p > 0.05	0	p > 0.05	0
RF-MCC	p > 0.05	0	p > 0.05	0	p > 0.05	0
LR-Accuracy	p < 0.01	5%	p < 0.05	3%	p < 0.01	3%
LR-Precision	p > 0.05	4%	p > 0.05	6%	p < 0.05	4%
LR-Recall	p < 0.01	7%	p > 0.05	−2%	p > 0.05	2%
LR-F1	p < 0.01	7%	p > 0.05	0	p < 0.01	4%
LR-AUC	p < 0.01	5%	p > 0.05	2%	p < 0.05	3%
LR-MCC	p < 0.01	2%	p > 0.05	12%	p < 0.01	15%
DT-Accuracy	p > 0.05	1%	p > 0.05	0.9%	p > 0.05	0.9%
DT-Precision	p > 0.05	0.9%	p > 0.05	0.9%	p > 0.05	0.9%
DT-Recall	p > 0.05	5%	p > 0.05	4%	p > 0.05	4%
DT-F1	p > 0.05	3%	p > 0.05	3%	p > 0.05	3%
DT-AUC	p < 0.05	2%	p < 0.05	2%	p < 0.05	2%
DT-MCC	p > 0.05	8%	p > 0.05	8%	p > 0.05	7%
NB-Accuracy	p > 0.05	7%	p < 0.05	2%	p < 0.05	2%
NB-Precision	p < 0.01	−5%	p > 0.05	3%	p > 0.05	3%
NB-Recall	p < 0.01	61%	p < 0.05	4%	p < 0.05	4%
NB-F1	p < 0.01	31%	p < 0.05	5%	p < 0.05	5%
NB-AUC	p < 0.01	7%	p < 0.05	2%	p < 0.05	2%
NB-MCC	p < 0.05	26%	p < 0.05	9%	p < 0.05	9%
KNN-Accuracy	p < 0.01	3%	p > 0.05	1%	p > 0.05	2%
KNN-Precision	p < 0.05	3%	p < 0.05	4%	p < 0.05	4%
KNN-Recall	p < 0.01	7%	p > 0.05	−0.8%	p > 0.05	0
KNN-F1	p < 0.01	5%	p > 0.05	1%	p > 0.05	2%
KNN-AUC	p < 0.01	4%	p > 0.05	2%	p > 0.05	2%
KNN-MCC	p < 0.01	17%	p > 0.05	9%	p > 0.05	12%
MLP-Accuracy	p > 0.05	2%	p < 0.01	4%	p < 0.01	4%
MLP-Precision	p > 0.05	1%	p < 0.05	4%	p < 0.01	4%
MLP-Recall	p < 0.01	7%	p > 0.05	3%	p < 0.01	7%
MLP-F1	p < 0.05	5%	p < 0.05	4%	p < 0.01	7%
MLP-AUC	p > 0.05	2%	p < 0.05	3%	p < 0.01	4%
MLP-MCC	p > 0.05	8%	p < 0.05	13%	p < 0.01	17%

between Original Models and Transformed Models to exceed 2 in 30%–50% of cases. This indicates a significant change in feature importance that would threaten correct explanatory models.

Similar to the performance research, we also hope to obtain more detailed information when designing defect prediction methods, that is, which data transformation method affects the variation of the most important features on which models. Therefore, in Figures 7–13, we show the density distribution of the Top-k feature rank overlap and differences of the Top-k feature rank across all six classification models constructed from the transformed datasets and the original datasets. We can observe the following results.

When using RF and DT techniques to build the SDP models, there is large consistency in the Top-5 feature importance rank overlap and small differences in the Top-5 feature importance rank between Original Models and Transformed Models no matter which data transformation method is applied.

From Figure 7, we can find that no matter which data transformation method is used, both the highest peak and the median of the density plot of the Top-1 feature rank overlap, and the Top-3 feature rank overlap calculated by permutation or SHAP are around 1. About the Top-5 feature rank overlap, no matter which data transformation method

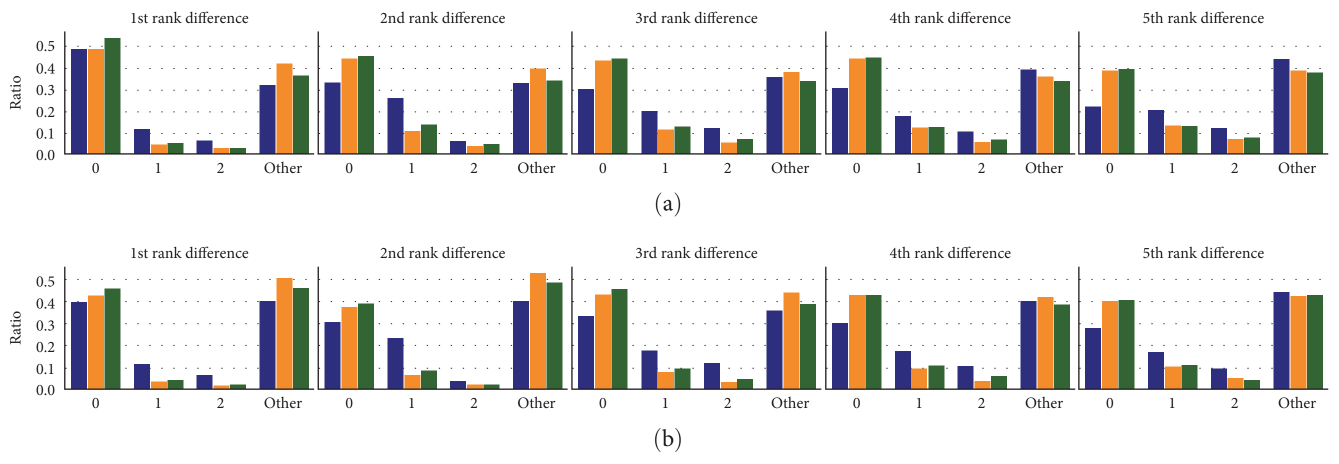


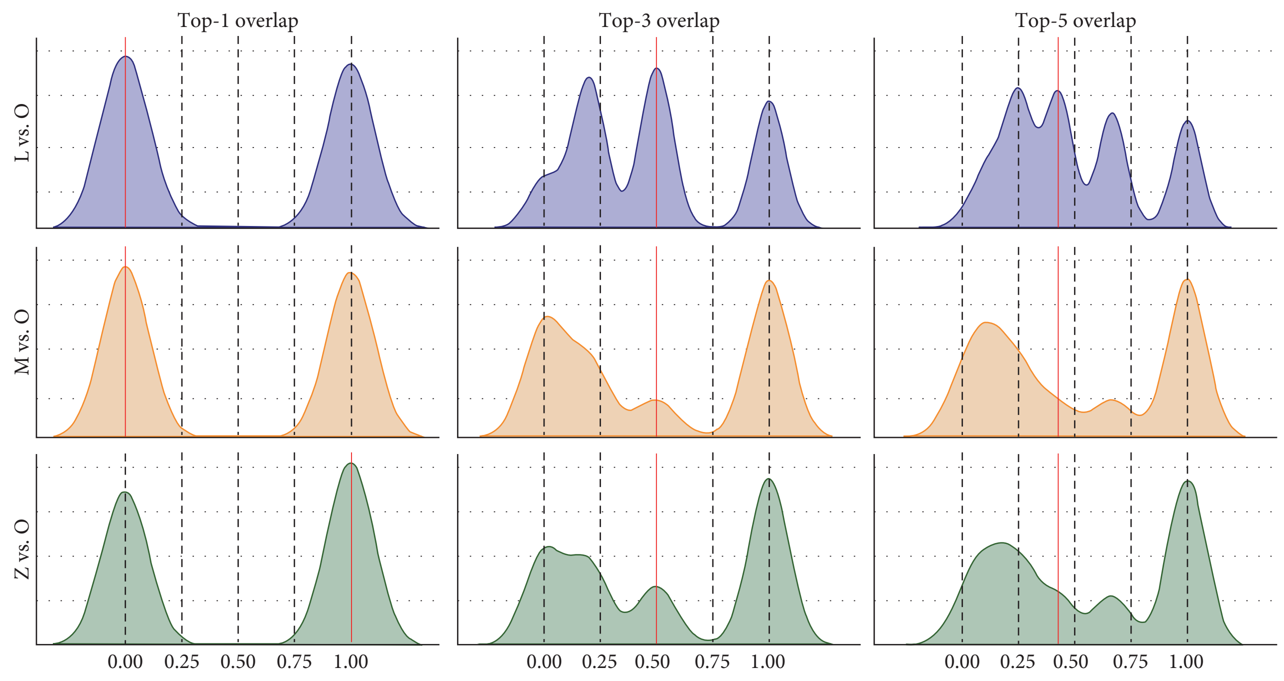
Figure 5: Bar plot of the overall Top-k feature importance rank difference computed by permutation and SHAP between the (a) transformed models and (b) original models.

except for log transformation is used, the highest peak and the median of the density plot are also basically around 1. And when log transformation is used, the median of the density plot is around 0.75. Similar to the results of the RF models, Figure 8 shows that no matter which data transformation method is used, the highest peak and the median of the density plot of the Top-k feature rank overlap ($k = 1, 3, 5$) are all around 1. Therefore, the RF and DT models constructed on the dataset transformed by the three data transformation methods have a large consistency in feature importance rank overlap. The first and fourth rows of Figure 13 show the feature rank difference of the RF model and the DT model calculated by permutation and SHAP, respectively. We can find that no matter which data transformation method is used, the proportion of Top-k feature rank difference values greater than 2 calculated by permutation or the SHAP is generally 0, and very few exceed 5%. The values of the Top-k feature rank difference are concentrated in 0, 1, 2. This means that the differences in feature importance rank of SDP models built on the dataset transformed by the three data transformation methods using RF and DT techniques are small.

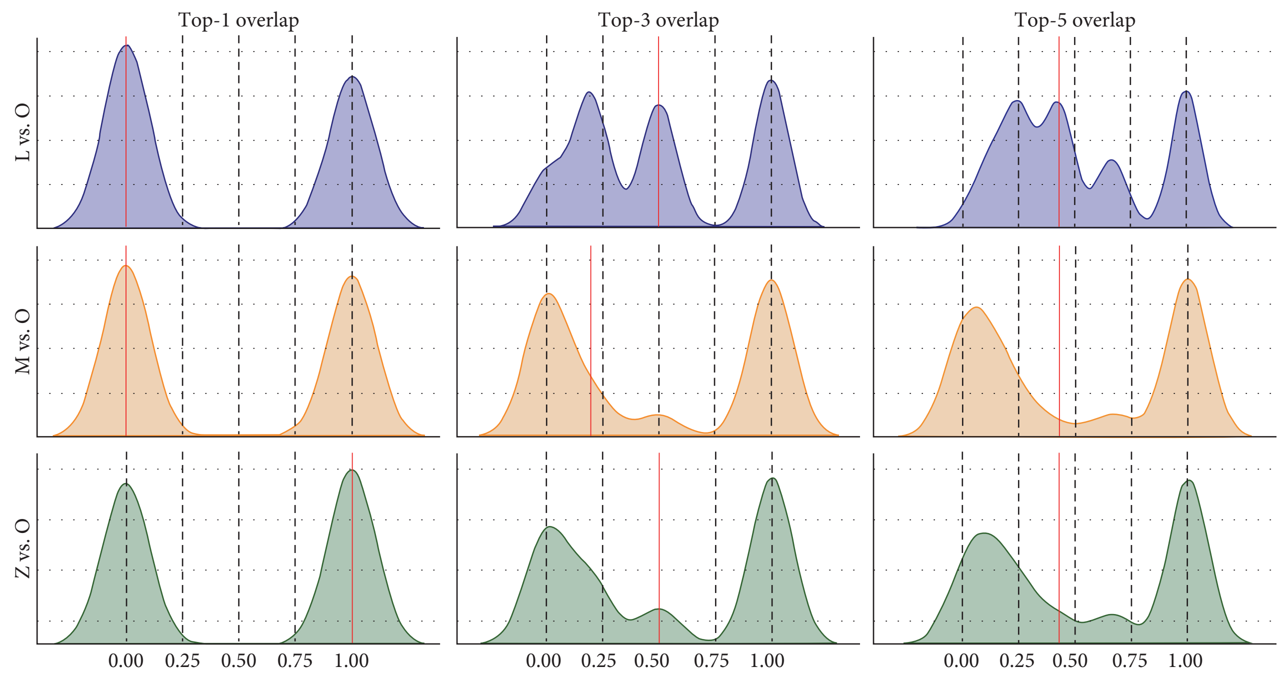
When using LR, KNN, and MLP techniques to build SDP models, there is small consistency in the Top-5 feature importance rank overlap and large differences in the Top-5 feature importance rank between Original Models and Transformed Models no matter which data transformation method is applied.

From Figure 9, we can find that no matter which data transformation method is used, both the highest peak and the median of the density plot of the Top-1 feature rank overlap calculated by permutation or SHAP are around 0. And the difference between the peak value at 0 and the peak value at 1 is extremely large. This means that no matter which data transformation method is used, the degree of consistency of the LR model on the most important features is extremely low. On the density plots of the Top-3 feature rank overlap and Top-5 feature rank overlap, we can find that there are much more peaks between 0 and 0.25 and between 0.25 and 0.5. Except that the median of the Top-5 feature rank overlap of the Transformed LR model calculated by permutation is between 0.25 and 0.5, the rest of the median is between 0 and 0.25. This means that no matter which data transformation method is used, the consistency degree of the LR model on Top-3 feature rank overlap and Top-5 feature rank overlap is negligible, atmost small.

We can observe the same result in Figures 10 and 11. Figure 10 shows the consistency of feature rank overlap of the KNN model calculated by permutation and SHAP, respectively. Figure 11 shows the consistency of feature rank overlap of the MLP model calculated by permutation and SHAP, respectively. Therefore, the LR, KNN, and MLP models constructed on the dataset transformed by the three data transformation methods have a small consistency in feature importance rank overlap. The second, fifth, and sixth rows of Figure 13 show the feature rank difference of the LR model, KNN model, and DT model calculated by permutation and SHAP, respectively. From the figures, we can find that on the LR model, the proportion of the Top-1 feature difference of the log transformation and the Z-score transformation calculated by permutation is greater than 2 is close to 40%. The proportion of Top-2 feature differences greater than 2 exceeds 40%. The proportion of the Top-3 feature difference greater than 2 is close to 60%. The proportion of Top-4 and Top-5 feature differences greater than 2 exceeds 60%. The proportion of the Top-k feature difference of the Minmax transformation method greater than 2 has always exceeded 60%. About the MLP model, the proportion of the Top-1 feature difference of the log transformation calculated by permutation is greater than 2 is close to 60%, the proportion of Top-2 and Top-3 feature difference greater than 2 is close to 50%, and the proportion of Top-4 and Top-5 feature difference greater than 2 is around 60%. The proportion of the Top-k feature difference of the Minmax transformation and Z-score transformation greater than 2 has always



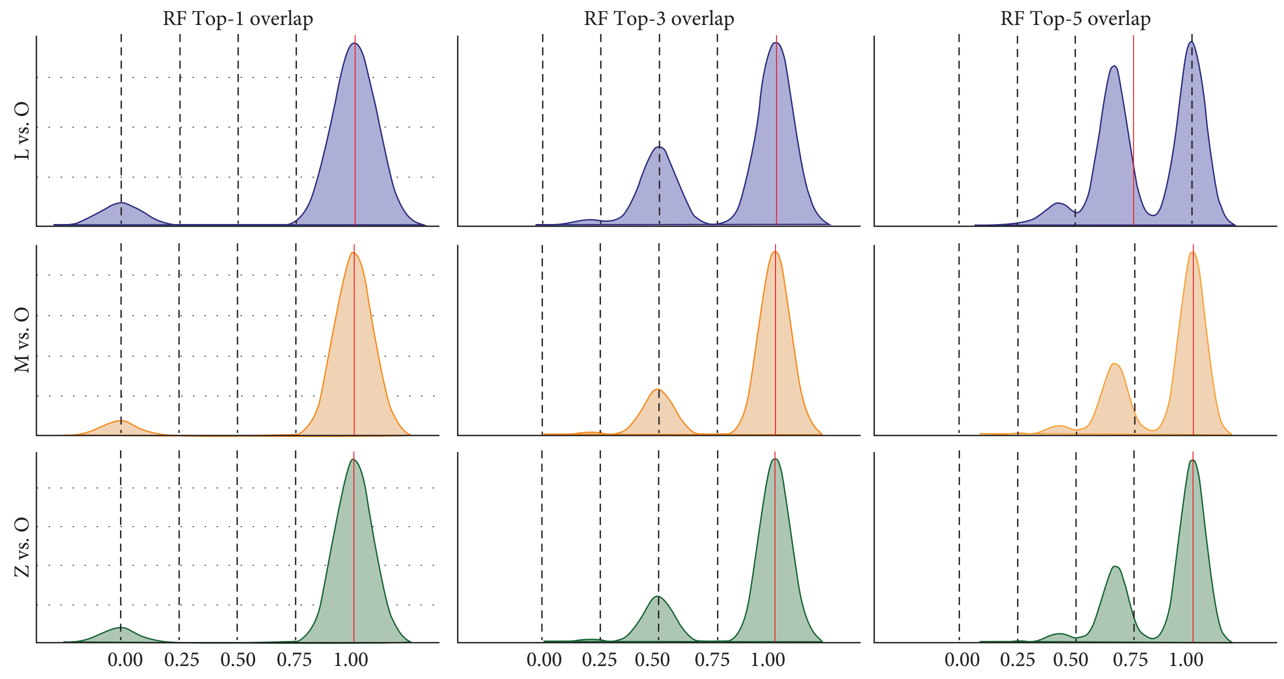
(a)



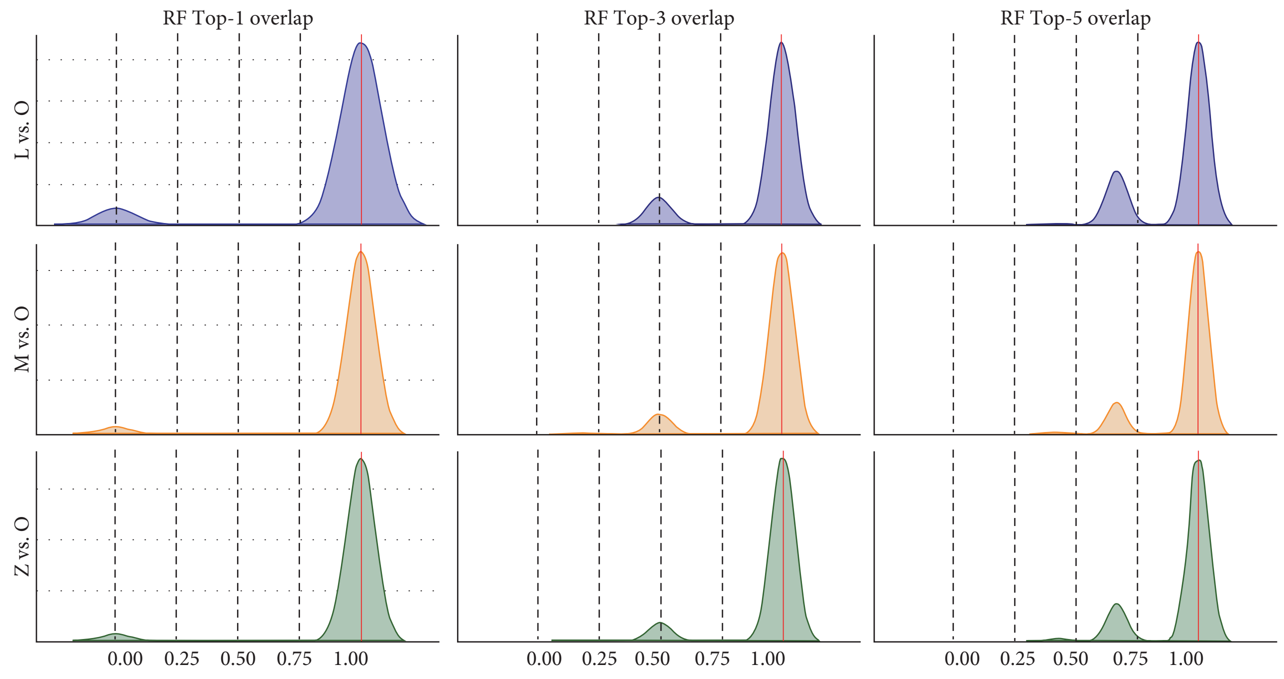
(b)

Figure 6: Density plot of the overall Top-*k* feature importance rank overlap computed by permutation and SHAP between the (a) transformed models and (b) original models.

exceeded 60%. The results of the SHAP are more pronounced. Regardless of the transformation method, the proportion of the Top-*k* feature difference of the LR model and MLP model greater than 2 is more than 60%. This means that the proportion of SDP models built on the datasets transformed by the three data transformation methods using LR and MLP techniques whose feature importance rank remains unchanged is very small, and most of them have a rank difference of more than 2, indicating a large difference in feature rank. The KNN model using the Minmax transformation and *Z*-score transformation method is similar to MLP in terms of feature difference, and the proportion of the Top-*k* feature difference greater than 2 is mostly more than 60%. However, in log transformation, the proportion of



(a)

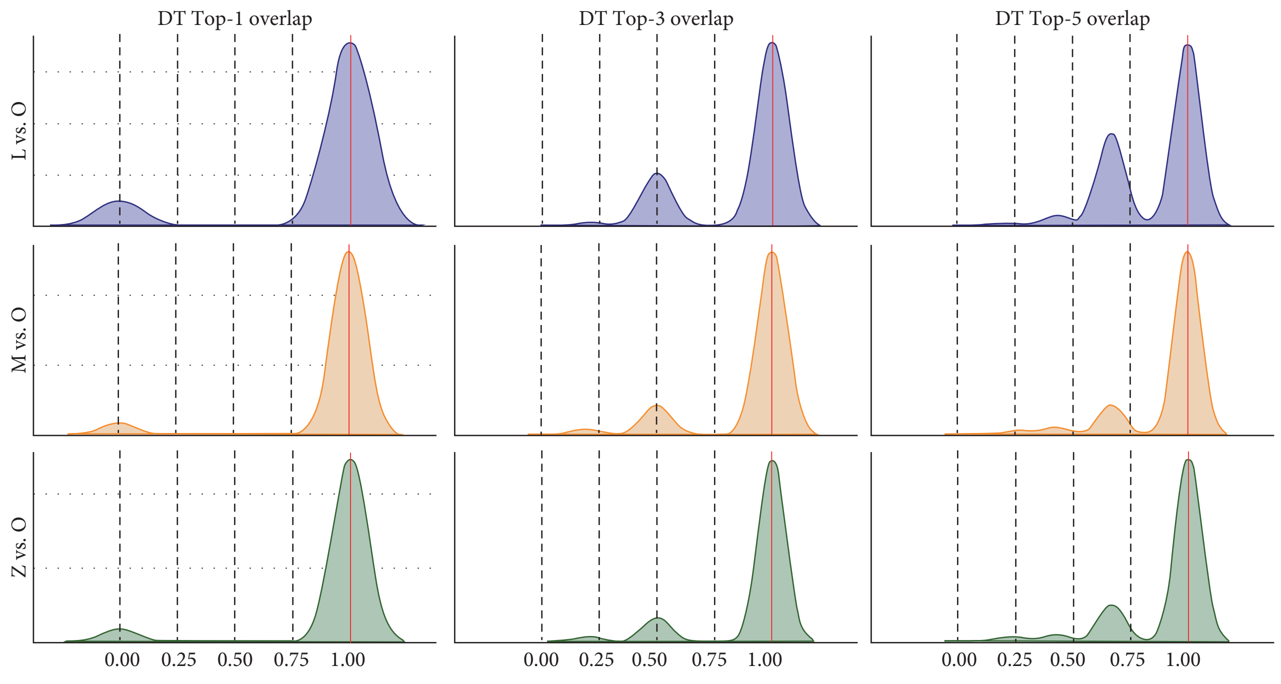


(b)

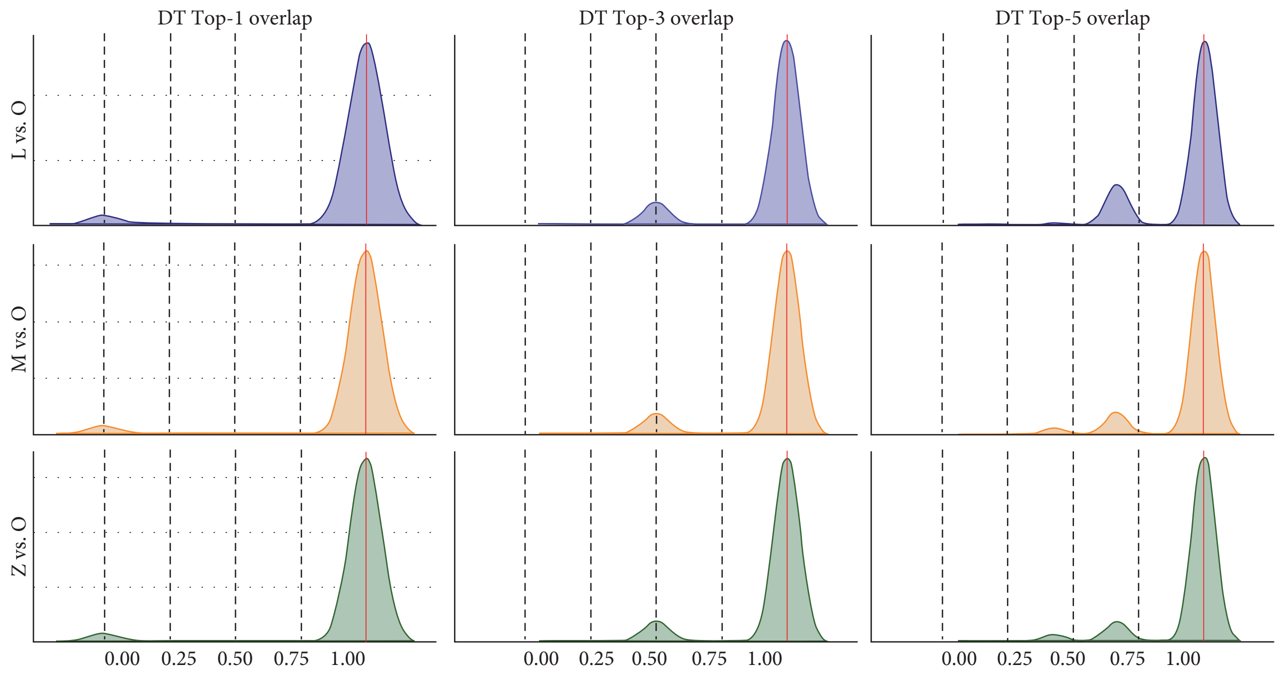
Figure 7: Density plot of the Top-*k* feature importance rank overlap computed by permutation and SHAP between the (a) RF transformed models and (b) original models.

KNN model rank difference of more than 2 on the Top-1 and Top-2 feature is less than 30%, and the proportion of KNN model rank difference of more than 2 on the Top-3, Top-4, and Top-5 feature is between 40% and 60%. This shows that the log transformation can keep the rank of the KNN model unchanged on the first two most important features, while the other features still show a high-rank difference.

In order to more intuitively see the changes in feature importance caused by the impact of data transformation methods on the dataset, we use the weighted *t*-SNE visualization method mentioned by Grisci et al. [80] as an additional experiment to identify the differences between the feature importance obtained with each data transformation method. We randomly selected an experiment of the LR model, and



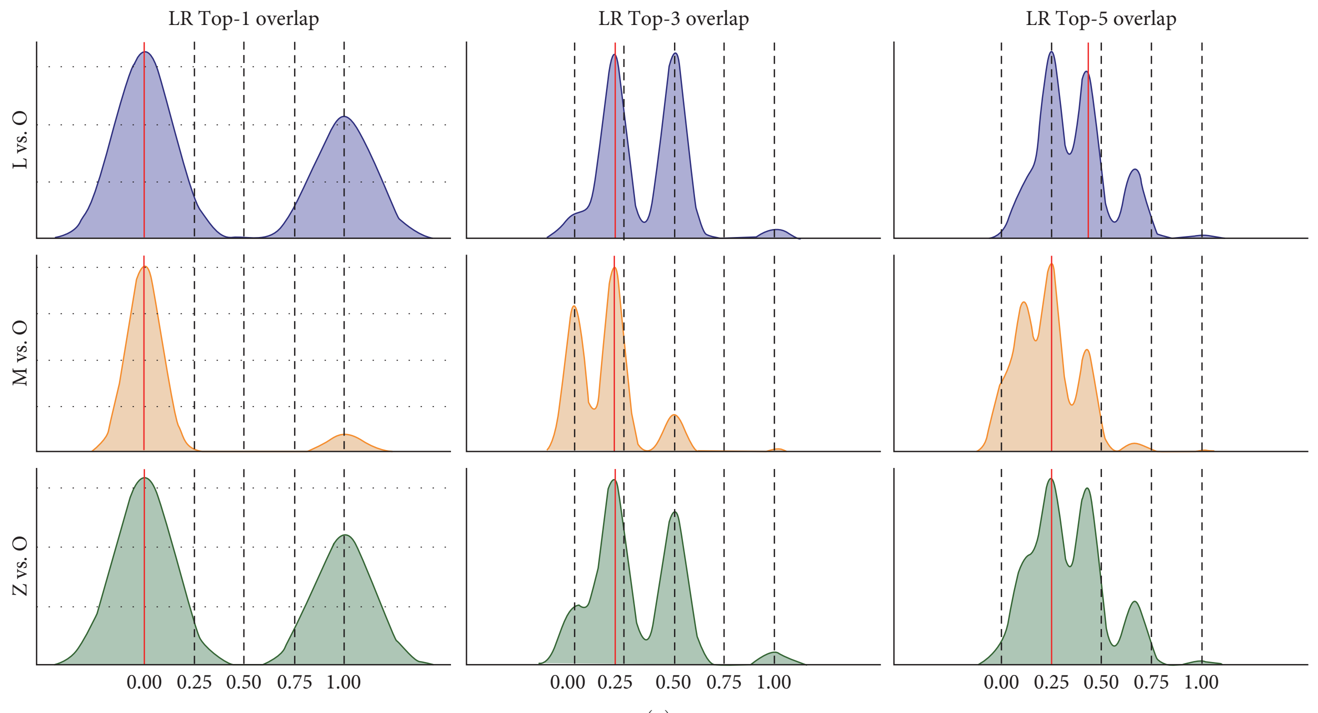
(a)



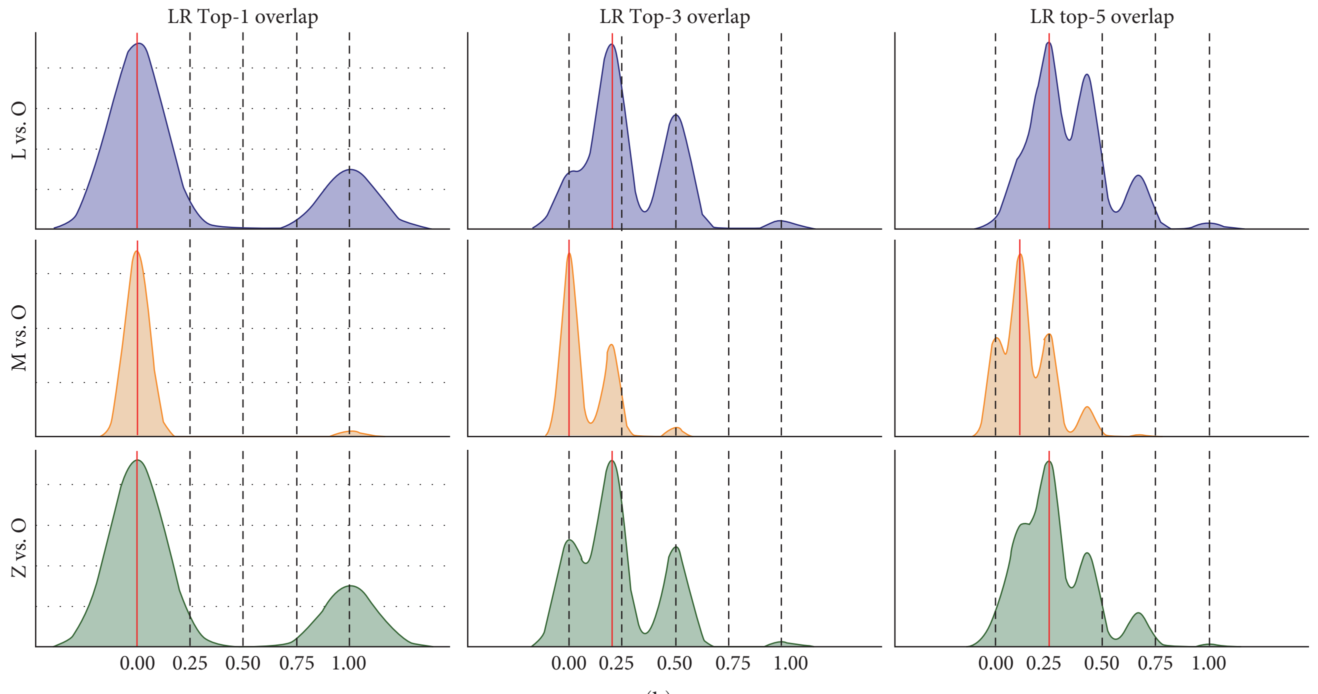
(b)

FIGURE 8: Density plot of the Top-k feature importance rank overlap computed by permutation and SHAP between the (a) DT transformed models and (b) original models.

plotted on the original dataset and the dataset transformed by the three methods, combining the feature importance scores as weights. The results are shown in Figure 11(c). It can be seen that the original dataset and the dataset transformed by the three transformation methods combined with the feature weights show different two-dimensional projections. This illustrates the significant difference in feature importance caused by different data transformation methods. In addition, we can also find that the difference in feature importance between the Minmax transformed dataset and the Original dataset is more significant. This corresponds to the above analysis, the proportion of the Top-k feature difference of the Minmax transformation method greater than 2 has always exceeded 60%.



(a)

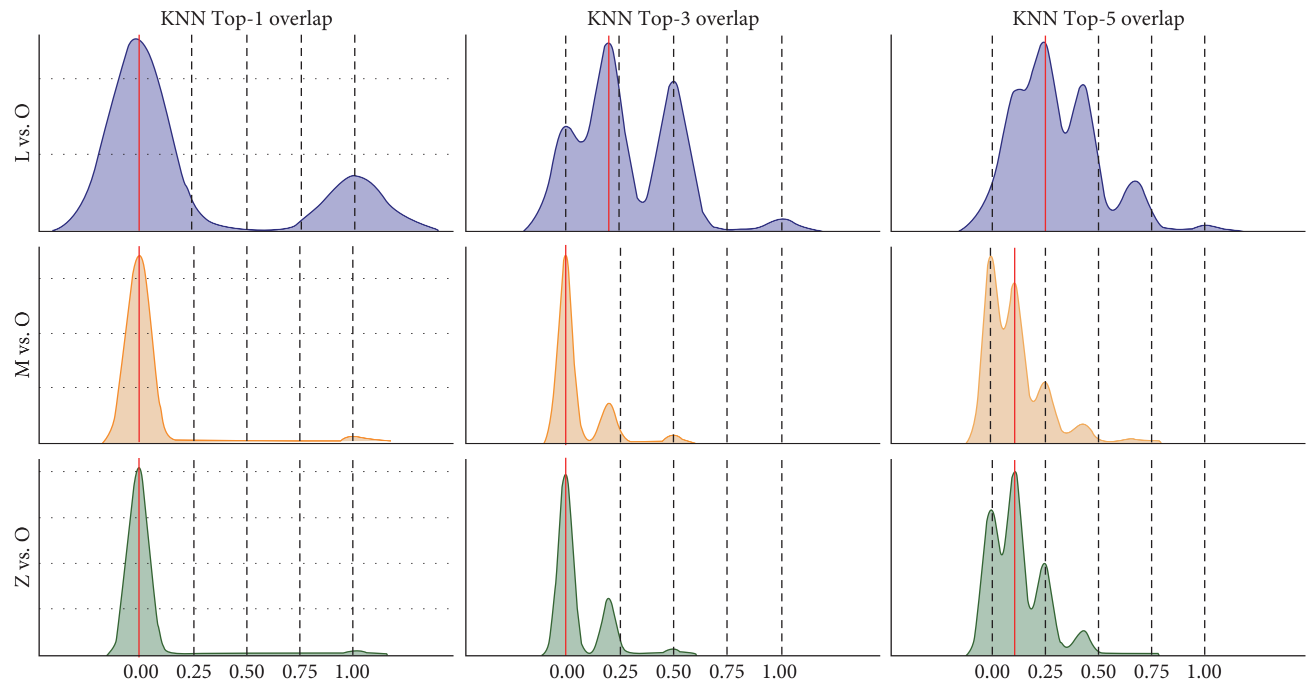


(b)

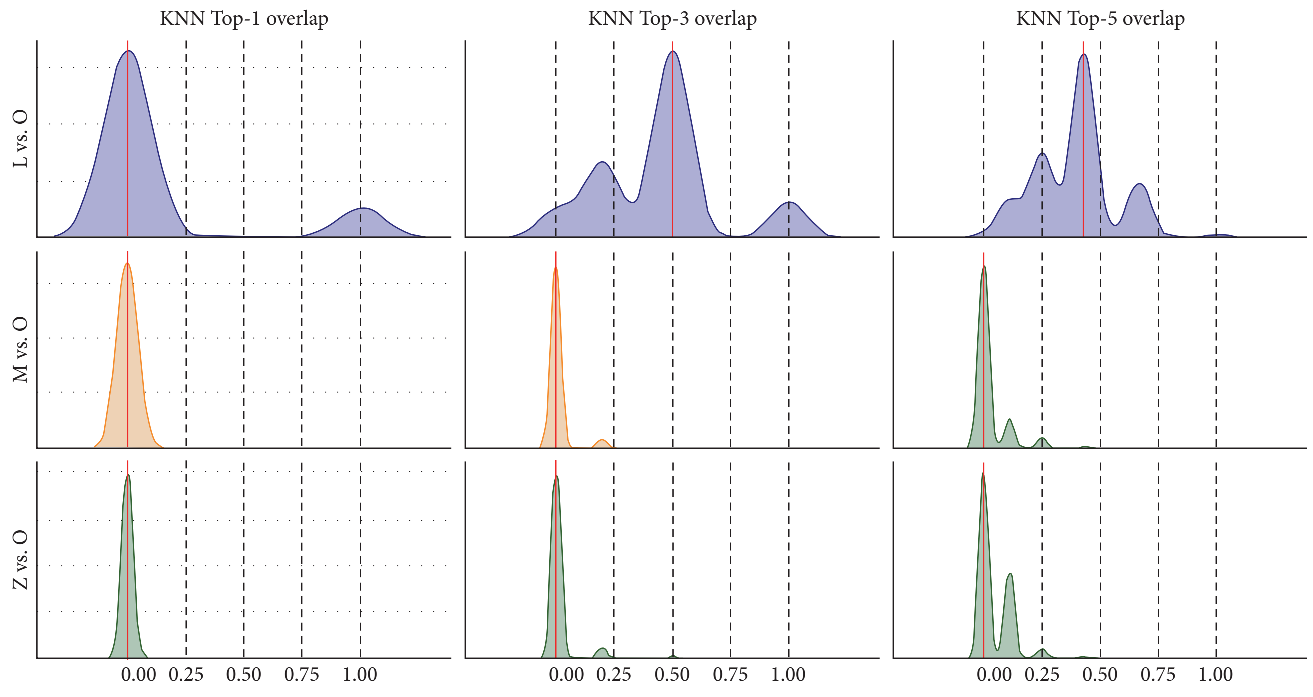
Figure 9: Density plot of the Top-*k* feature importance rank overlap computed by permutation and SHAP between the (a) LR transformed models and (b) original models.

When using NB technique to build SDP models, there is high consistency in the Top-5 feature importance rank overlap and small differences in the Top-5 feature importance rank between Original Models and Transformed Models using Minmax and Z-score transformation. However, there is low consistency in the Top-5 feature importance rank overlap and large differences in the Top-5 feature importance rank between Original Models and Transformed Models using Log transformation.

Figure 12 shows the consistency of feature rank overlap of the NB model calculated by permutation and SHAP,



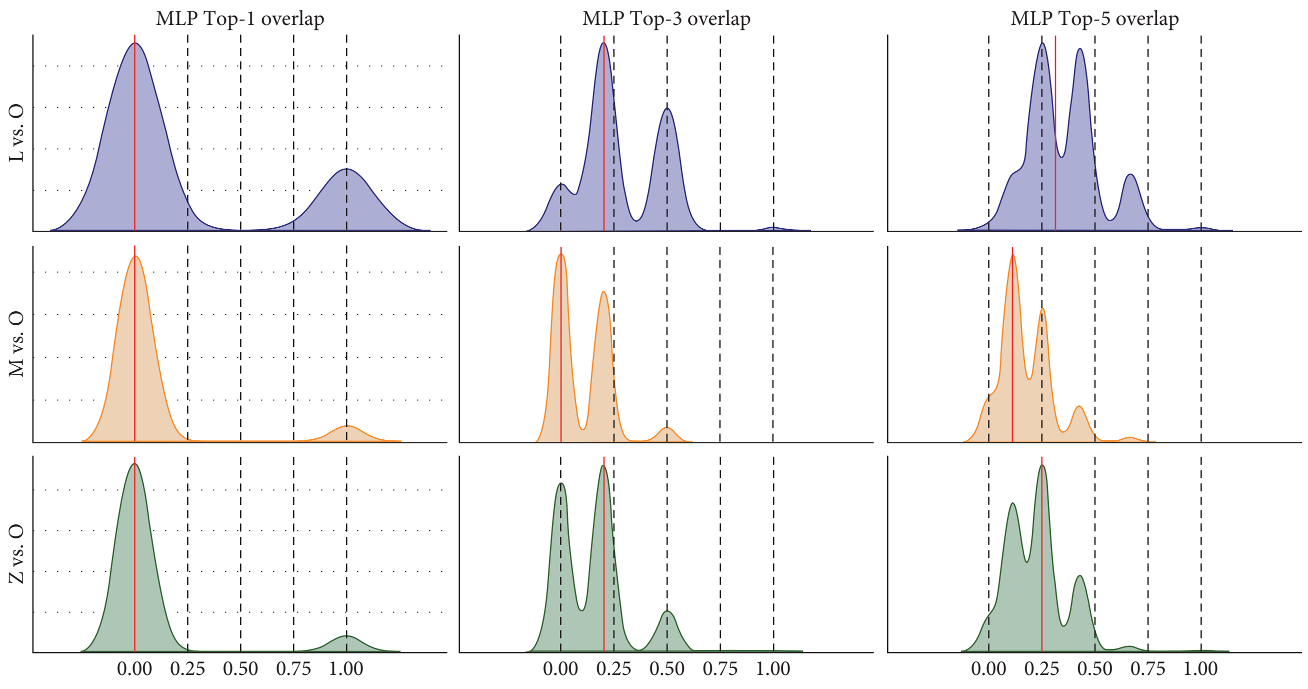
(a)



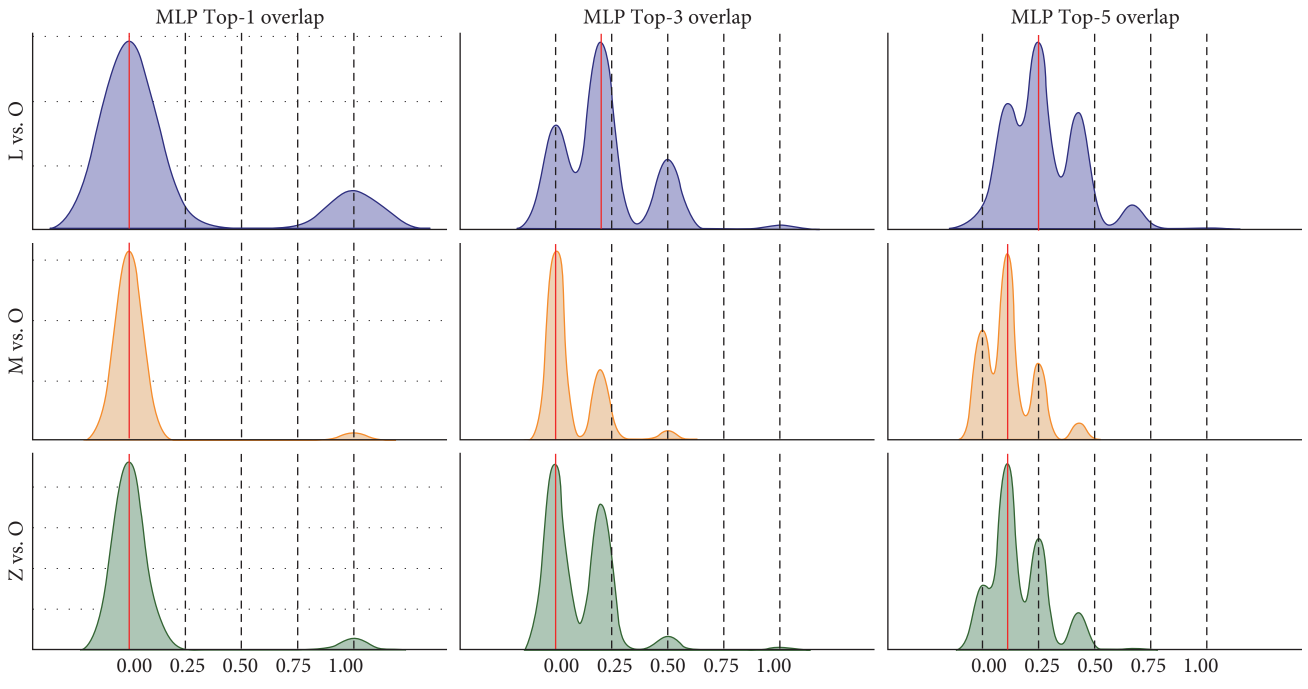
(b)

Figure 10: Density plot of the Top-*k* feature importance rank overlap computed by permutation and SHAP between the (a) KNN transformed models and (b) original models.

respectively. We can find that whether Minmax transformation or *Z*-score transformation is used, both the highest peak and the median of the density plot of the Top-*k* feature rank overlap calculated by permutation or SHAP are around 1. And the difference between the peak value at 1 and the peak value at 0 is extremely large. This means that the Minmax transformation and *Z*-score transformation make the NB model achieve high consistency on the Top-*k* feature rank overlap. But for log transformation, the highest peak and median of the density plot of the Top-1 feature rank overlap are around 0, and the peak at 0 is very different from the peak at 1. This means that the log transformation makes the consistency of the NB model on the most important features extremely small. On the density plot of the Top-3 and

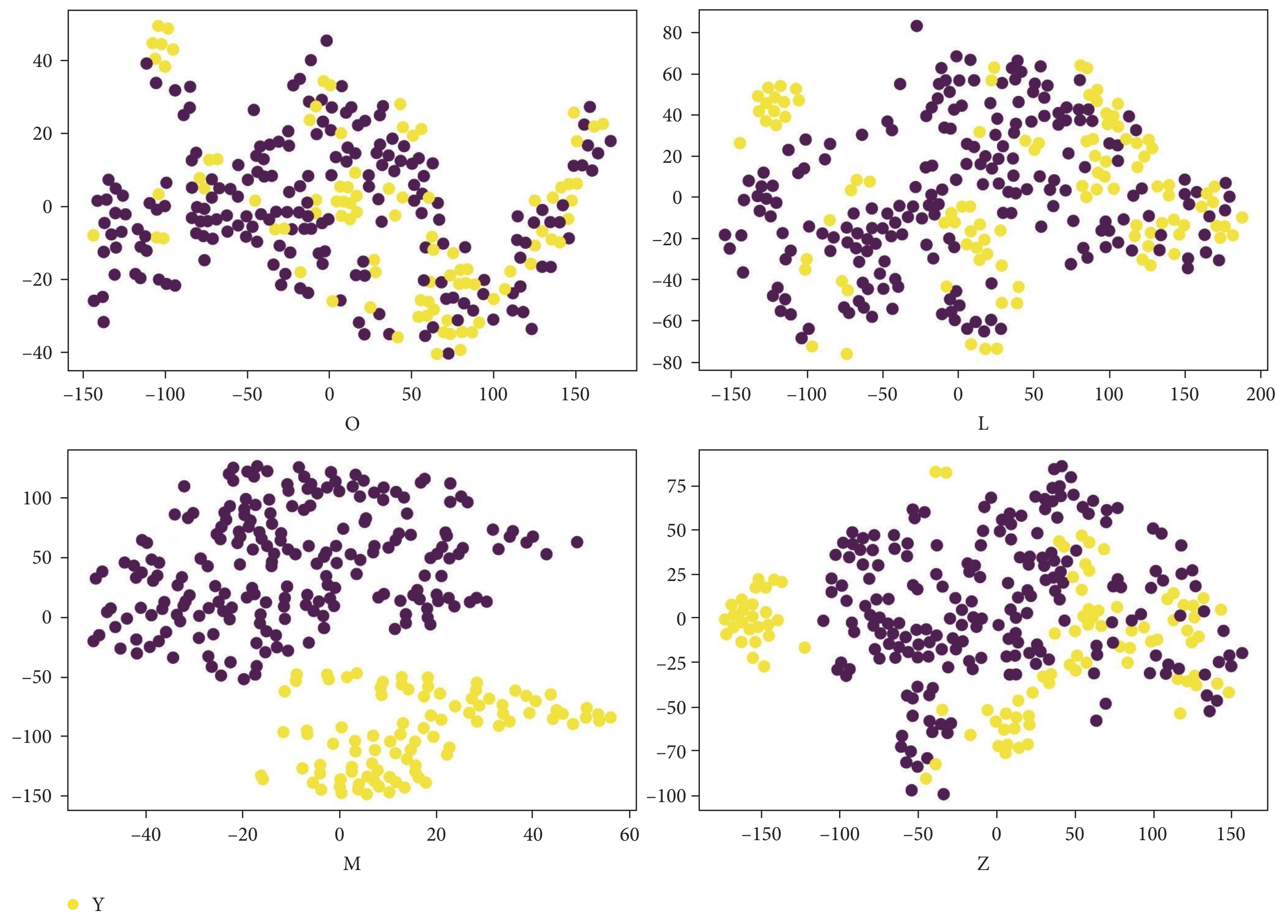


(a)



(b)

Figure 11: Continued.



(c)

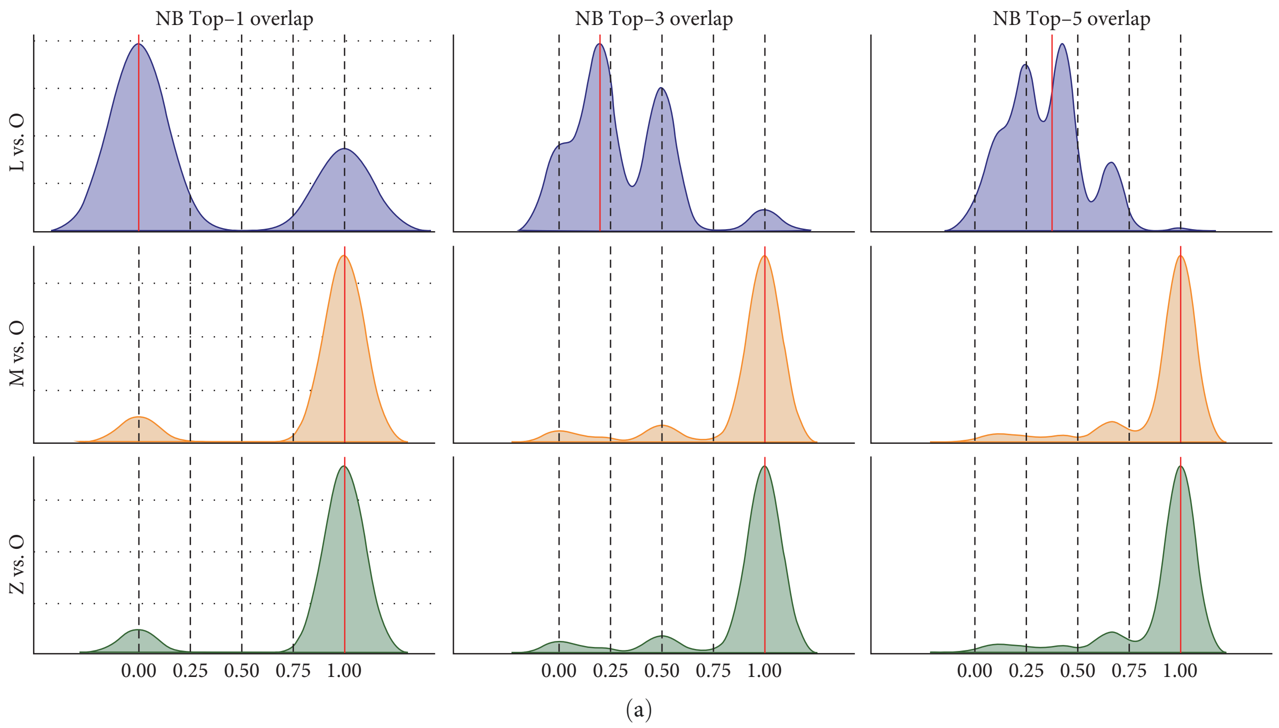
Figure 11: (a) and (b) are density plots of the Top-k feature importance rank overlap computed by permutation and SHAP between the MLP transformed models and original models, and (c) is weighted t-SNE on the original dataset and transformed datasets.

Top-5 feature rank overlap, we can find that there are more peaks and medians between 0 and 0.25, and only the median of Top-5 feature rank overlap calculated by permutation is between 0.25 and 0.5, which means that the log transformation makes the consistency of the NB model on the Top-k features low, at most small. From the perspective of feature differences (the third row of Figure 13), the rank of the Top-k ($k = 1,2,3,4$) features of Minmax transformation and Z-score transformation remains unchanged in a high proportion, regardless of permutation or SHAP, but the rank of the Top-5 feature with more than 2 differences is also close to 60%. This means that the NB model built on the transformed dataset using the Minmax transformation and Z-score transformation methods has a small rank difference on the Top-4 most important features and a large rank difference on the fifth most important feature. For log transformation, we can find that the difference of the Top-k feature rank calculated by permutation and SHAP is between 40% and 80%, which indicates that log transformation causes the feature importance rank of the NB model with large differences.

5. Discussion and Implication

This section discusses the experimental results and the implications drawn from the results. In order to understand the results of this paper more intuitively, we briefly summarize the conclusions in Table 4, which shows the models and indicators with significant performance improvements as well as changes in the feature importance. Below we will analyze performance, interpretability, and tradeoffs between the two, and give suggestions.

5.1. Effects on Model Performance. The three data transformation methods studied in this paper help to improve the performance of the SDP model overall. More detailed experimental results show that the three data transformation methods can improve the performance of the multiple models on various evaluation indicators, which can also be observed in Table 4. Even if there is no significant performance improvement on some models and indicators, it will rarely cause performance degradation. This shows that it is helpful to improve the performance of the SDP model by



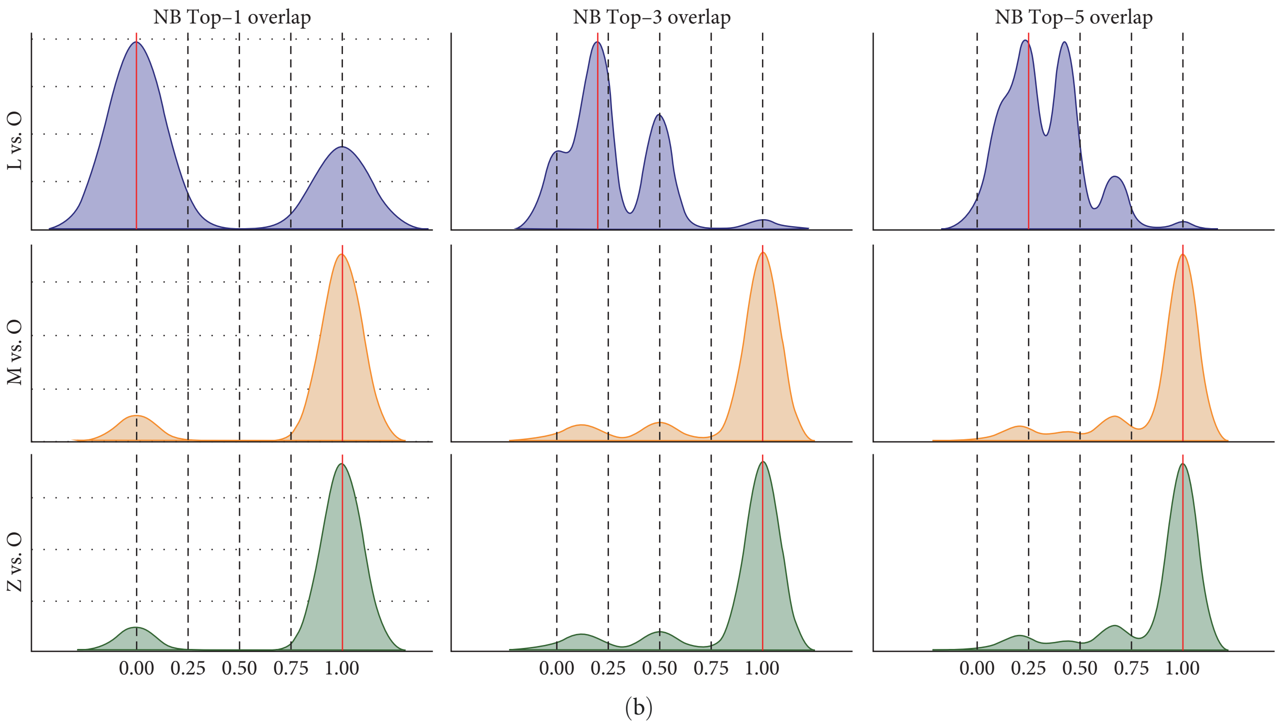
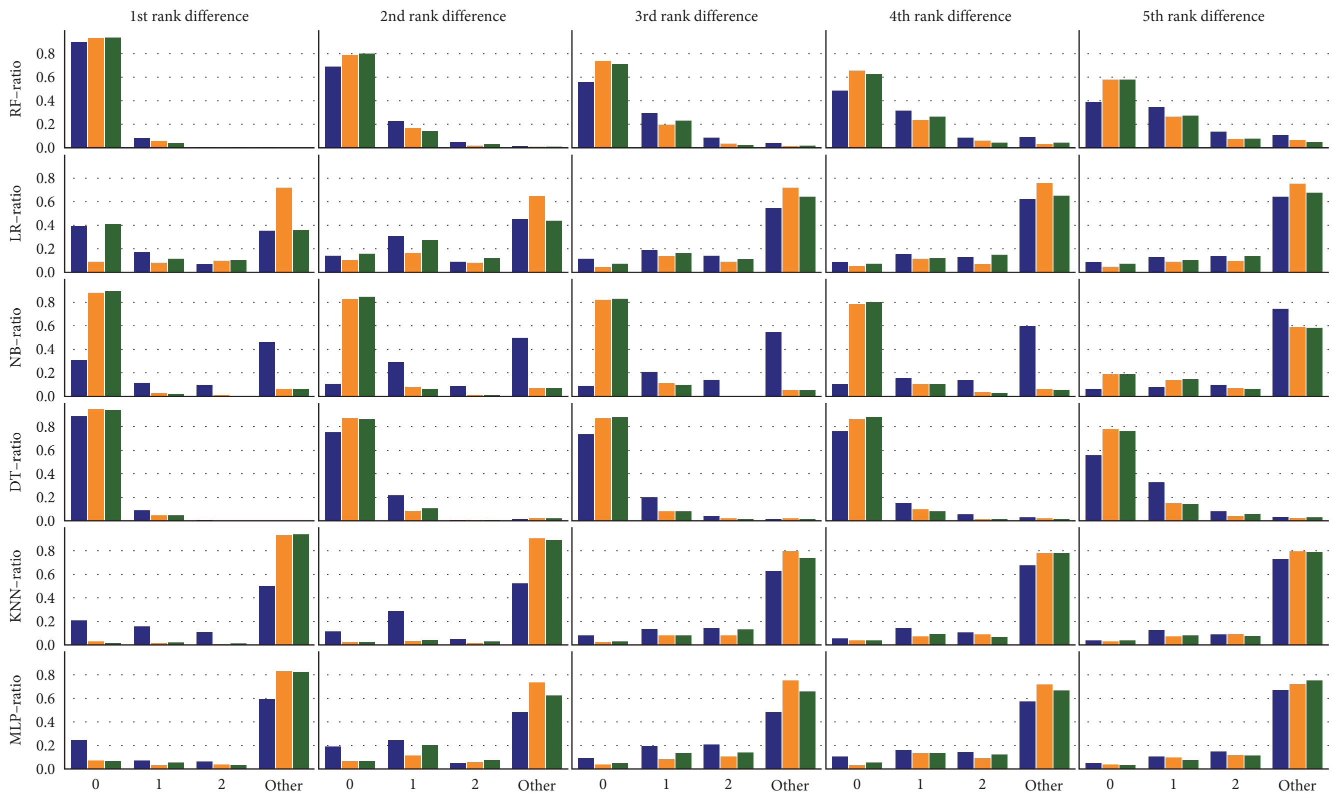


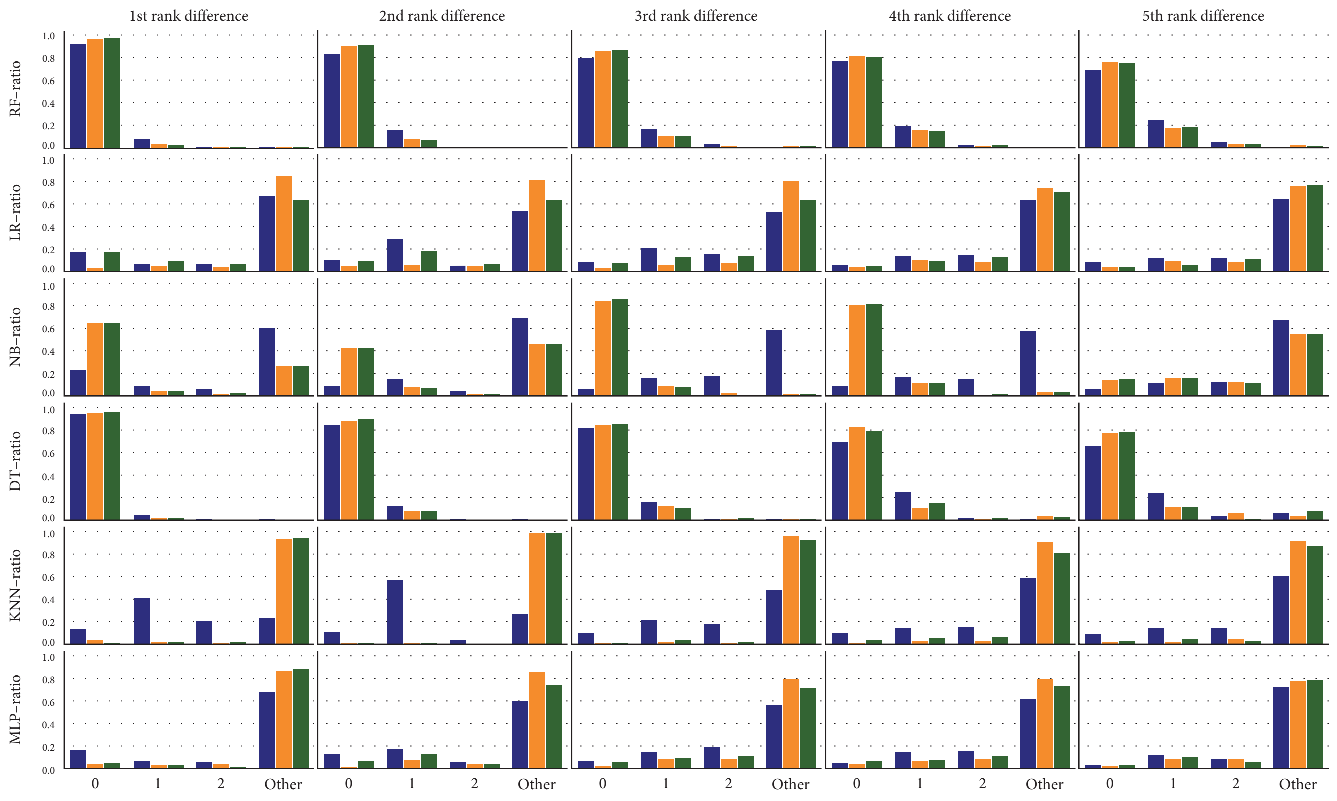
Figure 12: Density plot of the Top-*k* feature importance rank overlap computed by permutation and SHAP between the (a) NB transformed models and (b) original models.

selecting appropriate techniques and corresponding data preprocessing methods when designing defect prediction methods based on the results in Table 4. For example, without elaborately designing the defect prediction method, if the NB technique is selected to train the SDP model, simply using the log transformation method to process the training dataset can increase AUC by 7%, MCC by 26%, F1 by 31%, and Recall by 61%.

Previous historical studies proposed many defect prediction methods, and they reported a certain percentage improvement in performance, such as a 20% improvement in AUC. However, 10% of the 20% performance improvement is likely



(a)



(b)

FIGURE 13: Bar plot of the classifier Top-*k* feature importance rank difference computed by permutation and SHAP between the (a) transformed models and (b) original models.

Table 4: Summary of the impact of the transformation methods studied on the performance and interpretability of different SDP models.

Methods	Models	Performance increase	Importance change
Log	RF	Recall	None
	LR	Accuracy, Recall, F1, AUC, MCC	Top-1, 3, 5
	DT	AUC	None
	NB	Recall, F1, AUC, MCC	Top-1, 3, 5
	KNN	Accuracy, Precision, Recall, F1, AUC, MCC	Top-1, 3, 5
	MLP	Recall, F1	Top-1, 3, 5
Minmax	RF	None	None
	LR	Accuracy	Top-1, 3, 5
	DT	AUC	None
	NB	Accuracy, Recall, F1, AUC, MCC	None
	KNN	Precision	Top-1, 3, 5
	MLP	Accuracy, Precision, F1, AUC, MCC	Top-1, 3, 5
Z-score	RF	None	None
	LR	Accuracy, Precision, F1, AUC, MCC	Top-1, 3, 5
	DT	AUC	None
	NB	Accuracy, Recall, F1, AUC, MCC	None
	KNN	Precision	Top-1, 3, 5
	MLP	Accuracy, Precision, Recall, F1, AUC, MCC	Top-1, 3, 5

Note. Only significantly improved indicators are displayed. None indicates that there are no significantly improved indicators or no feature importance changes.

contributed by the simple data preprocessing step of data transformation. Since it is not known whether defect prediction methods proposed in historical studies applied this data preprocessing step, the extent to which these defect prediction methods are actually effective is questioned. Therefore, we suggest that in the following research on improving the performance of design defect prediction methods, the details of the experimental settings should be described in detail, and the performance of the SDP model should be compared before and after applying the data transformation method, so as to enhance the effectiveness of the method and conclusions.

5.2. Effects on Model Interpretability. The three data transformation methods studied in this paper have a certain impact on the interpretability of the SDP model overall. It can be clearly seen from Table 4 that the feature importance of the SDP model constructed on the dataset transformed by the three data transformation methods using the RF and DT techniques is not affected. However, the feature importance of the LR model, KNN model, and MLP model will change significantly after the three data transformation methods are used. The feature importance of the NB model will be seriously affected by the log transformation but the Minmax transformation and *Z*-score transformation will not affect its feature importance. These results illustrate whether the use of data transformation methods affects the interpretability of SDP models depending on the classification technique used in the research. Therefore, the conclusion about which features are the most contributing features of the model prediction obtained in the past research on the interpretability of the SDP model should be questioned because simply using of data transformation methods will result in significant changes in the feature importance ranks. However, we do not know whether this method has been used as the preprocessing step in the historical research and this effect is not fixed. For example, Minmax transformation and *Z*-score transformation will not affect the importance rank of the Top-4 most important features of the NB model, but they affect the fifth most important feature. These minor changes may also cause differences in the final interpretability of the SDP model, or even lead to completely opposite conclusions.

5.3. Tradeoffs between Performance and Interpretability. There is indeed a tradeoff between the performance and interpretability of SDP models. Mori and Uchihira [19] analyzed the interpretability of various SDP models from the perspective of model transparency. They confirmed the hypothesis that a generally accepted assumption about the relationship between the prediction accuracy and interpretability of the SDP is the negative correlation through the accuracy indicators achieved by the SDP model and the transparency of the model components. That is, the improvement of model prediction accuracy will lead to the decline of model interpretability. As can be seen from Table 4, this hypothesis is also confirmed by the impact of three data transformation methods on SDP model performance and feature importance ranks in this paper. For example, although the SDP models built on the dataset transformed by the data transformation methods using the RF and DT techniques have achieved performance improvement on some indicators, the improvement is not significant. Correspondingly, the SDP models built on the dataset transformed by the three data transformation methods using the RF and DT techniques can basically maintain the feature importance rank unchanged compared with the SDP model built on the original dataset, so it will not affect the interpretability of the SDP model. That is to say, the data transformation methods do not significantly improve the performance of the RF model and DT model, nor do they affect the change of their most important feature ranks. A more obvious example is the NB model built by using the log transformation method, which shows significant performance improvement on a variety of evaluation indicators (i.e., Recall, F1, AUC, and MCC). Correspondingly, the most important feature of the model shows extremely low consistency rank overlap and great rank difference, which seriously affects the interpretability of the SDP model.

But Minmax transformation and *Z*-score transformation of the dataset and using NB to build the SDP model do achieve a balance between performance and interpretability to some extent. As can be seen from Table 4, Minmax transformation and *Z*-score transformation of the dataset improve the performance of the NB model on multiple indicators, while the importance of the Top-5 most important features has not changed. Therefore, a suggestion is to use Minmax and *Z*-score transformation to process the dataset and use the NB algorithm to build the SDP model. This ensures performance improvement while also ensuring that feature importance does not change.

In addition, the interpretability of the SDP model needs to meet certain performance requirements. If the prediction performance of the SDP model cannot meet the basic requirements, it is meaningless to pursue such explanations. However, how to achieve this balance based on actual scenarios of defect prediction has not been covered in this paper, and it is worth focusing on in the future.

5.4. Recommendations for Future Research. When the objective of designing defect prediction methods is to improve the performance of SDP models and prioritize plans under the limited testing resources, data transformation methods can be used for preprocessing the training dataset to build the SDP model. However, it is important to clearly indicate this step and conduct comparative experiments to demonstrate its effectiveness.

Choose a variety of common evaluation indicators for the experimental comparisons to enhance the effectiveness of methods and conclusions. Since we found in the experiment that the MCC of the SDP model built on the datasets after applying the data transformation method has an average performance improvement ratio of more than 10% on the five common models. This means that unreliable conclusions may be obtained if only MCC is selected as the performance evaluation indicator. This is the same as Tim et al. [23], who noted a flaw in past defect prediction studies that only rely on one or a few indicators like AUC to evaluate models, which may potentially introduce bias. This recommendation is not limited to the field of defect prediction, all classification scenarios should benefit from comprehensive evaluation metrics.

When the objective of designing defect prediction methods is to understand the features associated with software defects in the past and identify the most influential features for predicting whether a file is defective, it is not recommended to use data transformation methods. Because it will affect the change of feature importance and thus affect the interpretability of the SDP model. It may be difficult to maintain the high performance and strong interpretability of SDP models. However, if the data transformation method must be used to improve the performance of the SDP model, we suggest that the changes in the most important features of the SDP model constructed by the dataset before and after the transformation should be tracked to obtain a more comprehensive explanation.

6. Threats to Validity

The following is a brief discussion of the threats to the validity of conclusions.

6.1. Data Transformation Methods. The First is the data transformation method of the research object. This paper only studies the impact of three data transformation methods, namely log transformation, Minmax transformation, and *Z*-score transformation on the performance and interpretability of the SDP model. Although these three data transformation methods are commonly used data preprocessing methods, there are also other data transformation methods used in past defect prediction research. For example, rank transformation [81, 82], Box–Cox transformation [83, 84], discretization transformation [85, 86], and so on. These methods, like the three transformation methods studied in this paper, can also reduce the dimension difference of features and enhance the normality of data distribution. Therefore, the research conclusion of this paper may be limited by the number of data transformation methods and hinder the validity of the conclusion.

6.2. Datasets. The second is the datasets used in this paper. We conduct experiments on 16 open-source software projects in three datasets. Although these datasets and software projects are often used in the past research, the research conclusions based on these datasets may not be generalizable to all datasets, especially some private software projects that are not open source. In addition, we only conduct research on software projects with a defect rate of 30%–70% and exclude extremely unbalanced software projects with a defect rate lower than 30% and higher than 70%. Since the class imbalance phenomenon may affect the conclusions about model performance and interpretability [61, 62], the conclusions of this paper should be revalidated on multiple class-imbalanced datasets.

6.3. Classification Techniques. Then it is the classification techniques used in this paper. We use six classification techniques frequently used by the researchers including RF, LR, NB, DT, KNN, and MLP. These six classifiers take into account the requirements of quantity and type at the same time, and the conclusions obtained should be generalizable. However, there are also many models and types of classifiers, including composite models with stronger classification capabilities formed by combining these basic classification models. Therefore, the conclusions of this paper may be limited by this.

6.4. Evaluation Indicators. The next is the evaluation indicators used in this paper. For performance evaluation indicators, we selected six performance evaluation indicators, including Accuracy, Precision, Recall, F1, MCC, and AUC. Similar to classification techniques, these indicators also take into account the requirements of quantity and type, and these indicators have been widely used in the previous defect prediction research [86, 87]. Nevertheless, more other types of indicators are not used in this paper [88–90], such as G-measure and FPR, etc. Recently effort-aware indicators have also been proposed to measure the performance of SDP models [91–93]. For the evaluation of the interpretability of SDP models, we evaluate model interpretability by computing Top-k feature rank overlap and Top-k feature rank differences to measure changes in Top-k feature importance [63, 64]. However, it is also a feasible idea to evaluate the interpretability of SDP models through the transparency and complexity of models, components, and algorithms [23]. In addition, evaluating interpretability from the perspective of user perception is a suitable evaluation method in line with the engineering practices. Therefore, the validity of the conclusions may also be affected by the different evaluation indicators.

6.5. Experiment Procedure. Finally is the process implementation of this paper. In order to enhance the reproducibility of the experiments and the credibility of the research conclusions, we implemented all the experimental steps using Python on the Anaconda platform. The classification techniques and evaluation indicators involved in the paper come from the scikit-learn package. In order to enhance the reproducibility of the results, we set the seed to 0 for each model. For the performance indicators value, we choose the average value of 25 sampling experiments, and the seeds of these 25 samplings are from 0 to 24, respectively. We also calculated the average of 25 random permutations when calculating permutation feature importance. In this way, the possibility of errors caused by the randomness of the experimental results is also minimized. However, there may still be manual errors. We will make all the data and codes during the experiment public and put them in the open-source code repository to ensure reproducibility and promote future research. We hope to attract the interest of relevant researchers and continue to integrate more in-depth research.

7. Conclusion

In this paper, through empirical research on 16 software projects in three open-source software datasets, we investigate the impact of three data transformation methods, namely log transformation, Minmax transformation, and Z-score transformation, on the performance and interpretability of SDP models. We found that using Log, Minmax, and Z-score transformation can significantly improve the performance indicator values of most SDP models. But the importance of features has also changed greatly in the most SDP models. The impact on the performance and interpretability of the SDP model depends on the classification technique used to design the defect prediction method.

According to the different objectives of defect prediction, we give the following suggestions. When the goal is to improve the performance of the SDP model and allocate limited test resources reasonably, the data transformation method can be used for data preprocessing of the training dataset to build the SDP model. But this step should be indicated and ablation experiments should be performed. When the goal is to gain knowledge and insights from SDP models but data transformation methods have to be used to improve the model performance. We propose that changes in the most important features of SDP models constructed from datasets before and after transformation should be tracked so that more comprehensive, accurate, and traceable conclusions can be drawn on the interpretability of the SDP model.

Data Availability

Data supporting this research article are available in the paper.

Conflicts of Interest

The authors and co-authors all declare no conflicts of interest.

Authors' Contributions

Yu Zhao, Zhiqiu Huang, and Lina Gong contributed in the conceptualization, methodology, formal analysis, investigation, resources, and writing–original draft preparation. Yi Zhu, Qiao Yu, Yu Zhao, Zhiqiu Huang, Lina Gong, and Yuxiang Gao contributed in the software, validation, data curation, writing–review and editing, visualization, supervision, project administration, and funding acquisition.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under grant No. U2241216, No. 62077029, and No. 62202223, the Natural Science Foundation of Jiangsu Province, China under grant No. BK20220881, the Foundation of the Key National Laboratory of New Technology in Computer Software (Nanjing University) under grant No. KFKT2022B20, and the Foundation of the Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics).

References

[1] T. McDermott, D. DeLaurentis, P. Beling, M. Blackburn, and M. Bone, "AI4SE and SE4AI: a research roadmap," *INSIGHT*, vol. 23, no. 1, pp. 8–14, 2020.

[2] N. C. Shrikanth and T. Menzies, "The early bird catches the worm: better early life cycle defect predictors," arXiv preprint arXiv: 2105.11082, 2021.

[3] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?" *Software Quality Journal*, vol. 26, no. 2, pp. 525–552, 2018.

[4] Q. Yu, S. Jiang, J. Qian, L. Bo, L. Jiang, and G. Zhang, "Process metrics for software defect prediction in object-oriented programs," *IET Software*, vol. 14, no. 3, pp. 283–292, 2020.

[5] L. Gong, G. K. Rajbahadur, A. E. Hassan, and S. Jiang, "Revisiting the impact of dependency network metrics on software defect prediction," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 5030–5049, 2021.

[6] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering*, pp. 297–308, Association for Computing Machinery, 2016.

[7] Z. Xu, S. Li, J. Xu et al., "LDFR: learning deep feature representation for software defect prediction," *Journal of Systems and Software*, vol. 158, Article ID 110402, 2019.

[8] F. Yang, G. Zeng, F. Zhong, W. Zheng, and P. Xiao, "Interpretable software defect prediction incorporating multiple rules," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 940–947, IEEE, 2023.

[9] Y. Gao, Y. Zhu, and Q. Yu, "Evaluating the effectiveness of local explanation methods on source code-based defect prediction models," in *Proceedings of the 19th International Conference on Mining Software Repositories (MSR)*, pp. 640–645, Association for Computing Machinery, 2022.

[10] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on the interpretation of defect models," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 320–331, 2019.

[11] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2107–2145, 2016.

[12] X. Y. Jing, S. Ying, Z. W. Zhang, S. S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pp. 414–423, Association for Computing Machinery, 2014.

[13] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2006.

[14] A. Amin, B. Shah, A. M. Khattak, T. Baker, and S. Anwar, "Just-in-time customer churn prediction: with and without data transformation," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–6, IEEE Press, 2018.

[15] H. Jia, F. Shu, Y. Yang, and Q. Li, "Data transformation and attribute subset selection: do they help make differences in software failure prediction?" in *2009 IEEE International Conference on Software Maintenance*, pp. 519–522, IEEE, 2009.

[16] W. Zheng, T. Shen, X. Chen, and P. Deng, "Interpretability application of the just-in-time software defect prediction model," *Journal of Systems and Software*, vol. 188, Article ID 111245, 2022.

[17] H. K. Dam, T. Tran, and A. Ghose, "Explainable software analytics," in *Proceedings of the 40th International Conference on Software Engineering: new ideas and emerging results*, pp. 53–56, Association for Computing Machinery, 2018.

[18] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "The impact of automated feature selection techniques on the interpretation of defect models," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3590–3638, 2020.

[19] T. Mori and N. Uchihira, "Balancing the trade-off between accuracy and interpretability in software defect prediction," *Empirical Software Engineering*, vol. 24, no. 2, pp. 779–825, 2019.

[20] K. E. Bennin, J. W. Keung, and A. Monden, "On the relative value of data resampling approaches for software defect prediction," *Empirical Software Engineering*, vol. 24, no. 2, pp. 602–636, 2019.

[21] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Computing*, vol. 22, pp. 77–88, 2019.

[22] S. Qiu, L. Lu, and S. Jiang, "Multiple-components weights model for cross-project software defect prediction," *IET Software*, vol. 12, no. 4, pp. 345–355, 2018.

[23] A. Agrawal and T. Menzies, "Is "Better Data" better than "Better Data Miners"?" in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, pp. 1050–1061, Association for Computing Machinery, 2018.

[24] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology*, vol. 107, pp. 125–136, 2019.

[25] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *Information and Software Technology*, vol. 114, pp. 204–216, 2019.

[26] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, B. Xu, and S. Ying, "Heterogeneous defect prediction with two-stage ensemble learning," *Automated Software Engineering*, vol. 26, no. 3, pp. 599–651, 2019.

[27] H. Wei, C. Hu, S. Chen, Y. Xue, and Q. Zhang, "Establishing a software defect prediction model via effective dimension reduction," *Information Sciences*, vol. 477, pp. 399–409, 2019.

[28] F. Wu, X.-Y. Jing, Y. Sun et al., "Cross-project and within-project semisupervised software defect prediction: a unified approach," *IEEE Transactions on Reliability*, vol. 67, no. 2, pp. 581–597, 2018.

[29] Z. Xu, J. Liu, X. Luo, and T. Zhang, "Cross-version defect prediction via hybrid active learning with kernel principal component analysis," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 209–220, IEEE, 2018.

[30] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction," *Software Quality Journal*, vol. 26, no. 1, pp. 97–125, 2018.

[31] Z. Xu, S. Pang, T. Zhang et al., "Cross project defect prediction via balanced distribution adaptation based transfer learning," *Journal of Computer Science and Technology*, vol. 34, pp. 1039–1062, 2019.

[32] Y. Jiang, B. Cukic, and T. Menzies, "Can data transformation help in the detection of fault-prone modules?" in *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, pp. 16–20, Association for Computing Machinery, 2008.

[33] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The misuse of the NASA metrics data program data sets for automated software defect prediction," in *Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering*, pp. 96–103, IET, 2011.

[34] J. Jiarpakdee, C. K. Tantithamthavorn, and J. Grundy, "Practitioners' perceptions of the goals and visual explanations of defect prediction models," in *Proceedings of the 18th International Conference on Mining Software Repositories (MSR)*, pp. 432–443, IEEE, 2021.

[35] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, and X. Yang, "Perceptions, expectations, and challenges in defect prediction," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1241–1266, 2018.

[36] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? A case study of the Qt, VTK, and ITK projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 171–180, IEEE, 2015.

[37] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146–2189, 2016.

[38] Y. Fan, X. Xia, D. Lo, and S. Li, "Early prediction of merged code changes to prioritize reviewing tasks," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3346–3393, 2018.

[39] L. Bao, X. Xia, D. Lo, and G. C. Murphy, "A large scale study of long-time contributor prediction for github projects," *IEEE Transactions on Software Engineering*, vol. 47, no. 6, pp. 1277–1298, 2021.

[40] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? An empirical case study of Windows Vista," in *2009 IEEE 31st International Conference on Software Engineering*, pp. 518–528, IEEE, 2009.

[41] J. Moeyersoms, E. Junqué de Fortuny, K. Dejaeger, B. Baesens, and D. Martens, "Comprehensible software fault and effort prediction: a data mining approach," *Journal of Systems and Software*, vol. 100, pp. 80–90, 2015.

[42] G. Esteves, E. Figueiredo, A. Veloso, M. Viggiato, and N. Ziviani, "Understanding machine learning software defect predictions," *Automated Software Engineering*, vol. 27, pp. 369–392, 2020.

[43] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Transactions on Software Engineering*, 2020.

[44] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The Impact of mislabelling on the performance and interpretation of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 812–823, IEEE, 2015.

[45] S. Herbold, "Comments on ScottKnottESD in response to "An empirical comparison of model validation techniques for defect prediction models"," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1091–1094, 2017.

[46] S. Feng, J. Keung, X. Yu et al., "Complexity-based Over-Sampling TEchnique to alleviate the class imbalance problem in software defect prediction," *Information and Software Technology*, vol. 129, Article ID 106432, 2021.

[47] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Software Engineering*, vol. 21, no. 1, pp. 43–71, 2016.

[48] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, 2020.

[49] A. E. C. Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 460–463, IEEE, 2009.

[50] Ö. F. Arar and K. Ayan, "A feature dependent Naive Bayes approach and its application to the software defect prediction problem," *Applied Soft Computing*, vol. 59, pp. 197–209, 2017.

[51] A. Marjuni, T. B. Adji, and R. Ferdiana, "Unsupervised software defect prediction using signed Laplacian-based spectral classifier," *Soft Computing*, vol. 23, no. 24, pp. 13679–13690, 2019.

[52] M. A. Pradan, M. B. Mizan, M. Howlader, and S. Ripon, "An efficient approach to software fault prediction," in *International Conference on Communication, Computing and Electronics Systems*, V. Bindhu, J. M. R. S. Tavares, A. A. A. Boulogeorgos, and C. Vuppalapati, Eds., vol. 733 of *Lecture Notes in Electrical Engineering*, pp. 221–237, Springer, Singapore, 2021.

[53] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[54] A. Iqbal, S. Aftab, U. Ali et al., "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 5, pp. 300–308, 2019.

[55] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: some comments on the NASA Software Defect Datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[56] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 789–800, IEEE, 2015.

[57] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "The jinx on the NASA software defect data sets," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, pp. 1–5, Association for Computing Machinery, 2016.

[58] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, pp. 1–10, Association for Computing Machinery, 2010.

[59] R. Wu, H. Zhang, S. Kim, and S. C. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pp. 15–25, Association for Computing Machinery, 2011.

[60] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, pp. 531–577, 2012.

[61] L. Gong, S. Jiang, L. Bo, L. Jiang, and J. Qian, "A novel class-imbalance learning approach for both within-project and cross-project defect prediction," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 40–54, 2020.

[62] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[63] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1200–1219, 2020.

[64] Y. Gao, Y. Zhu, and Y. Zhao, "Dealing with imbalanced data for interpretable defect prediction," *Information and Software Technology*, vol. 151, Article ID 107016, 2022.
[65] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2017.
[66] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "Auto-Spearman: automatically mitigating correlated software metrics for interpreting defect models," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 92–103, 2018.
[67] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 321–332, IEEE, 2016.
[68] G. K. Rajbahadur, S. Wang, G. A. Oliva, Y. Kamei, and A. E. Hassan, "The impact of feature importance methods on the interpretation of defect classifiers," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2245–2261, 2022.
[69] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.
[70] A. Fisher, C. Rudin, and F. Dominici, "All models are wrong, but many are useful: learning a variable's importance by studying an entire class of prediction models simultaneously," *Journal of Machine Learning Research*, vol. 20, no. 177, pp. 1–81, 2019.
[71] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 4768–4777, Curran Associates Inc, 2017.
[72] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and Information Systems*, vol. 41, no. 3, pp. 647–665, 2014.
[73] J. Jiarpakdee, "Towards a more reliable interpretation of defect models," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 210–213, IEEE, 2019.
[74] I. C. Covert, S. Lundberg, and S.-I. Lee, "Understanding global feature contributions with additive importance measures," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, vol. 33, pp. 17212–17223, Curran Associates Inc, 2020.
[75] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should i trust you?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, Association for Computing Machinery, 2016.
[76] S. Wattanakriengkrai, P. Thongtanunam, C. Tantithamthavorn, H. Hata, and K. Matsumoto, "Predicting defective lines using a model-agnostic technique," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1480–1496, 2022.
[77] D. Bowes, T. Hall, M. Harman, Y. Jia, F. Sarro, and F. Wu, "Mutation-aware fault prediction," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pp. 330–341, Association for Computing Machinery, 2016.
[78] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: the use of machine learning in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 40, no. 6, pp. 603–616, 2014.
[79] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bulletin de la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.
[80] B. I. Grisci, M. J. Krause, and M. Dorn, "Relevance aggregation for neural networks interpretability and knowledge discovery on tabular data," *Information Sciences*, vol. 559, pp. 111–129, 2021.
[81] F. Zhang, I. Keivanloo, and Y. Zou, "Data transformation in cross-project defect prediction," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3186–3218, 2017.
[82] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, pp. 182–191, Association for Computing Machinery, 2014.
[83] H. L. Shang, "Selection of the optimal Box–Cox transformation parameter for modelling and forecasting age-specific fertility," *Journal of Population Research*, vol. 32, no. 1, pp. 69–79, 2015.
[84] M. Begum and T. Dohi, "A neuro-based software fault prediction with Box–Cox power transformation," *Journal of Software Engineering and Applications*, vol. 10, no. 3, pp. 288–309, 2017.
[85] Z. A. Rana, M. A. Mian, and S. Shamail, "Improving recall of software defect prediction models using association mining," *Knowledge-Based Systems*, vol. 90, pp. 1–13, 2015.
[86] Z. Hu and Y. Zhu, "Cross-project defect prediction method based on genetic algorithm feature selection," *Engineering Reports*, Article ID e12670, 2023.
[87] X. Chen, D. Zhang, Z.-Q. Cui, Q. Gu, and X.-L. Ju, "DP-share: privacy-preserving software defect prediction model sharing through differential privacy," *Journal of Computer Science and Technology*, vol. 34, pp. 1020–1038, 2019.
[88] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," in *Proceedings of the 40th International Conference on Software Engineering*, pp. 1063–1063, Association for Computing Machinery, 2018.
[89] N. Limsettho, K. E. Bennin, J. W. Keung, H. Hata, and K. Matsumoto, "Cross project defect prediction using class distribution estimation and oversampling," *Information and Software Technology*, vol. 100, pp. 87–102, 2018.
[90] Y. Zhao, Y. Wang, D. Zhang, and Y. Gong, "Eliminating the high false-positive rate in defect prediction through BayesNet with adjustable weight," *Expert Systems*, vol. 39, no. 6, Article ID e12977, 2022.
[91] X. Yu, K. E. Bennin, J. Liu, J. W. Keung, X. Yin, and Z. Xu, "An empirical study of learning to rank techniques for effort-aware defect prediction," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 298–309, IEEE, 2019.
[92] Y. Yang, Y. Zhou, J. Liu et al., "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 157–168, Association for Computing Machinery, 2016.
[93] Y. Zhou, Y. Yang, H. Lu et al., "How far we have progressed in the journey? An examination of cross-project defect prediction," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 1, pp. 1–51, 2018.