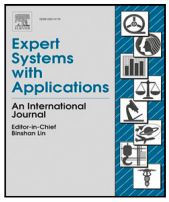
Expert Systems With Applications 235 (2024) 121084

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Improving the undersampling technique by optimizing the termination condition for software defect prediction

Shuo Feng [a], Jacky Keung [b], Yan Xiao [c,d], Peichang Zhang [e,*], Xiao Yu [f], Xiaochun Cao [c]

[a] School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, China
[b] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong, China
[c] School of Cyber Science and Technology, Shenzhen Campus of Sun Yat-sen University, China
[d] School of Computing, National University of Singapore, Singapore
[e] College of Electronics and Information Engineering, Shenzhen University, Shenzhen, China
[f] School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China

ARTICLE INFO

Keywords:
Software defect prediction
Class imbalance
Learning-to-rank
Undersampling
Oversampling
Data resampling

ABSTRACT

The class imbalance problem significantly hinders the ability of the software defect prediction (SDP) models to distinguish between defective (minority class) and non-defective (majority class) software instances. Recent studies on the data resampling technique have shown that Random UnderSampling (RUS) is more effective than several complex oversampling techniques at alleviating this problem. However, RUS blindly removes majority class instances, leading to significant information loss. These studies have also pointed out that the conventional termination condition (i.e., terminating the data resampling technique when the number of instances for both the minority and majority classes are the same) of the data resampling technique can result in suboptimal performance.

In fact, the undersampling technique can be likened to a recommender system or a web search engine that recommends majority class instances to SDP models. Therefore, we propose the Learning-To-Rank Undersampling technique (LTRUS). Our work is novel in two aspects: (1) We consider the undersampling process as a learning-to-rank task, optimizing a linear model to rank majority class instances and remove them from the bottom of the rank to alleviate the class imbalance problem. (2) We propose two termination conditions for the undersampling technique, which differ from the conventional termination condition.

LTRUS significantly outperforms RUS, the clustering-based undersampling technique, the complexity-based oversampling technique, SMOTUNED, and Borderline-SMOTE in terms of F-measure, AUC, and MCC by 8.9%, 7.6%, and 18.0% on average under the conventional termination condition. Furthermore, LTRUS under the two termination conditions we propose yield similar performance, and both outperform LTRUS and all the other baselines under the conventional termination condition. The experimental results demonstrate the effectiveness of LTRUS and indicate that the conventional termination condition for the data resampling technique is improper.

1. Introduction

Software defect prediction (SDP) relies on machine learning models that use the historical instances (e.g., classes, files and packages) to predict whether the instances introduced in the future are defective or non-defective (Feng, Keung, Zhang, Xiao, & Zhang, 2022; Jin, 2021; Song & Minku, 2022; Tahir, Bennin, Xiao, & MacDonell, 2021; Yu et al., 2022). This allows practitioners to allocate their limited testing resources effectively to those defect-prone instances based on the prediction (Li et al., 2023; Yu et al., 2023). However, datasets benefiting from software quality assurance activities (Tian, 2005) have more non-defective (i.e., majority class) instances than defective (i.e., minority class) ones. The imbalance between the minority class and majority class instances makes prediction models focus more on the majority class instances and ignore the minority class instances, leading to the degradation in the performance[1] of prediction models. This phenomenon is called the class imbalance problem (Feng et al., 2020;

* Corresponding author.
E-mail addresses: fengshuo@zzu.edu.cn (S. Feng), jacky.keung@cityu.edu.hk (J. Keung), xiaoy367@mail.sysu.edu.cn (Y. Xiao), pzhang@szu.edu.cn (P. Zhang), xiaoyu@whut.edu.cn (X. Yu), caoxiaochun@mail.sysu.edu.cn (X. Cao).

[1] The performance in this work refers to the values of different performance measures (e.g., F-measure, the Area Under the ROC Curve, or the Matthews Correlation Coefficient) calculated based on the prediction results of prediction models.

https://doi.org/10.1016/j.eswa.2023.121084
Received 6 January 2023; Received in revised form 28 July 2023; Accepted 29 July 2023
Available online 3 August 2023

Feng, Keung, Yu, Xiao, & Zhang, 2021; Japkowicz & Stephen, 2002). Generally, the techniques to alleviate the class imbalance problem can be categorized into three types: the cost-sensitive learning technique (Gupta, Jindal, & Bedi, 2022; Iranmehr, Masnadi-Shirazi, & Vasconcelos, 2019; Petrides & Verbeke, 2022; Yu et al., 2019), the ensemble learning technique (Abedin, Guotai, Hajek, & Zhang, 2022; Bai, Jia, & Capretz, 2022; Chen, Jing, Zhou, Li, & Xu, 2022; Feng, Huang, & Ren, 2018; Jiang et al., 2021; Jiang, Yu, Gong, & Du, 2022; Wu, Lin, & Ji, 2018), and the data resampling technique (Bennin, Keung, Phannachitta, Monden, & Mensah, 2017; Douzas, Bacao, & Last, 2018; Gao, Zhu, & Zhao, 2022; Maldonado, López, & Vairetti, 2019; Tong, Lu, Xing, Liu, & Wang, 2022; Zhang et al., 2022). The data resampling technique (Agrawal & Menzies, 2018; Bennin et al., 2017; Feng et al., 2020), including the undersampling technique and the oversampling technique, is commonly used to alleviate the class imbalance problem in SDP. The undersampling technique removes majority class instances, while the oversampling technique generates minority class instances to balance datasets.

Based on Tantithamthavorn and Bennin's studies (Bennin, Keung, & Monden, 2019; Tantithamthavorn, Hassan, & Matsumoto, 2020), the Random UnderSampling technique (RUS), which is the most common and simplest undersampling technique, outperforms several complex oversampling techniques in SDP, such as Synthetic Minority Oversampling TEchnique (SMOTE) (Chawla, Bowyer, Hall, & Kegelmeyer, 2002), ADAptive SYNthetic sampling technique (ADASYN) (He, Bai, Garcia, & Li, 2008), and SMOTUNED (Agrawal & Menzies, 2018). This is because current oversampling techniques generate synthetic minority class instances that introduce noise into datasets (Feng et al., 2021). On the contrary, RUS removes majority class instances and does not introduce any noise. The superiority of RUS motivates us to explore the undersampling technique for alleviating the class imbalance problem. However, RUS blindly removes the majority class instances to balance datasets, which ignores that some instances (e.g., borderline instances Han, Wang, & Mao, 2005) provide more information to prediction models than others. Removing these instances can significantly hinder the performance of prediction models, while the removal of others does not.

In fact, the undersampling technique is similar to a recommender system (Lu, Wu, Mao, Wang, & Zhang, 2015). It recommends some majority class instances to SDP models while removing others to alleviate the class imbalance problem. Therefore, we regard the undersampling process as a learning-to-rank task and propose the Learning-To-Rank UnderSampling technique (LTRUS). First, we optimize a linear model $f(X)$ to calculate the relevance[2] of majority class instances to the performance of prediction models. These instances are then ranked in descending order based on the relevance. Finally, LTRUS removes majority class instances from the bottom of the rank until the final defect ratio of the dataset is 0.5 (i.e., the number of instances for both the minority and majority classes are equal). By ranking instances based on their relevance to the performance of prediction models, we can avoid the drawback of RUS blindly removing majority class instances and improve the undersampling technique. Fig. 1 illustrates the procedure of LTRUS.

Furthermore, a previous study (Agrawal & Menzies, 2018) has suggested that terminating the data resampling technique when the final defect ratio of datasets reaches 0.5 may be inappropriate. Therefore, we propose two termination conditions to further improve LTRUS. The first condition optimizes the final defect ratio, instead of rigidly setting the number of instances for both classes to be equal. We refer to LTRUS under this condition as LTRUS-ratio in this work. The second condition optimizes a threshold. If the relevance of the majority class instances calculated by the optimized model $f(X)$ in LTRUS is larger than the optimized threshold, it indicates that these instances are highly relevant to achieving a good performance of prediction models and should be preserved. Otherwise, the instances are removed. We refer to LTRUS under this condition as LTRUS-thres in this study. The intuition behind LTRUS-thres is that a reasonable termination condition of the undersampling technique should be one that preserves all the instances that can improve the performance of prediction models, while removing those that cannot enhance the performance, regardless of the final defect ratio. If LTRUS-ratio and LTRUS-thres perform better than LTRUS, it indicates that the conventional termination condition of the data resampling technique is improper. We use the differential evolution algorithm (DE) (Storn & Price, 1997) to optimize the linear model, the final defect ratio, and the threshold for LTRUS.

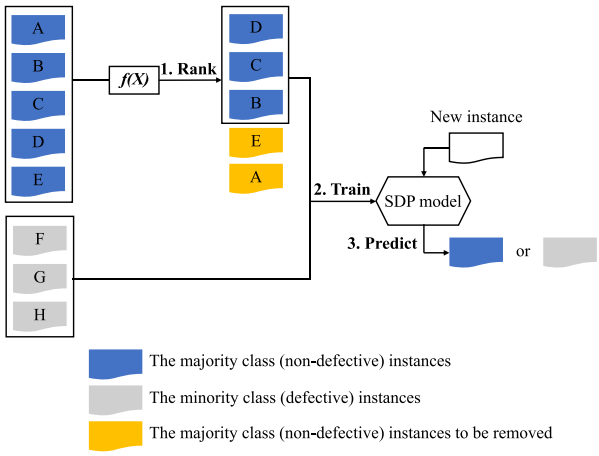


Fig. 1. The illustration of LTRUS.

We conducted extensive experiments to compare the performance of LTRUS with several data resampling techniques (i.e., RUS, the complexity-based oversampling technique (COSTE) Feng et al., 2020, the clustering-based undersampling technique (Cluster) Lin, Tsai, Hu, & Jhang, 2017, SMOTUNED Agrawal & Menzies, 2018, and Borderline-SMOTE (Borderline) Han et al., 2005) across 21 datasets collected from the AEEEM (D'Ambros, Lanza, & Robbes, 2012), Relink (Wu, Zhang, Kim, & Cheung, 2011), MORPH (Nam & Kim, 2015), SOFTLAB (Blanchard & Loubere, 2016), and NASA (Shepperd, Song, Sun, & Mair, 2013) repositories on four common classifiers (i.e., K-Nearest Neighbor (KNN), Random Forest (RF), Logistic Regression (LR), and Naive Bayes (NB)). Our experimental results show that LTRUS significantly outperforms the baselines by 8.9%, 7.6%, and 18.0% in terms of F-measure, the Area Under the ROC Curve (AUC), and the Matthews Correlation Coefficient (MCC) (Chicco & Jurman, 2020; Song, Guo, & Shepperd, 2018) on average under the conventional termination condition.

We have also observed that LTRUS-ratio and LTRUS-thres exhibit similar performances, with both outperforming LTRUS. However, the final defect ratios of the datasets processed by LTRUS-ratio and LTRUS-thres are significantly different. LTRUS-ratio removes more majority class instances, resulting in greater alleviation of the class imbalance problem, but also causing more information loss. On the other hand, LTRUS-thres removes fewer majority class instances, preserving more information, but it fails to alleviate the class imbalance problem as effectively as LTRUS-ratio does. This results in a similar performance and reveals that there is a trade-off inherent in the undersampling technique.

In summary, our contributions are:

- We propose an effective undersampling technique (LTRUS) and two termination conditions for LTRUS to alleviate the class imbalance problem. LTRUS learns to rank the majority class instances based on their relevance to the prediction model, so that less important instances are removed.

[2] The higher the relevance of specific majority class instances, the better the performance of the prediction model trained using these instances.

- We empirically reveal that (1) the conventional termination condition of the undersampling technique is improper, and (2) there is an undersampling trade-off between removing more majority class instances to balance datasets and preserving more majority class instances to provide prediction models with more information.
- We demonstrate the effectiveness of LTRUS and the two termination conditions. We have open-sourced the implementation of LTRUS and the datasets we used in this work to facilitate future studies (https://codeocean.com/capsule/8719212/tree/v1).

The rest of this paper is structured as follows: We introduce the motivation of our work in Section 2. Section 3 presents the related work and background. We provide details of the proposed methodology in Section 4. In Section 5, we describe the experimental design. The experimental results are presented in Section 6. We discuss our work in Section 7. Finally, Section 8 summarizes our conclusions and suggests future work.

2. Motivation

There is a major drawback of RUS that prevents its further improvement. RUS ignores the fact that the information provided by each majority class instance is different (e.g., borderline instances and non-borderline instances Han et al., 2005) for prediction models. It treats each majority class instance equally and randomly removes them. However, some majority class instances are indeed less important and removing these instances will not significantly hinder the performance of prediction models. Conversely, some majority class instances are crucial for prediction models, and removing them could result in significant performance degradation of prediction models. Therefore, careful consideration is required to determine which majority class instances should be removed to improve an undersampling technique.

Essentially, deciding which majority class instances to be removed by the undersampling technique is similar to some learning-to-rank tasks such as a recommender system (Lu et al., 2015) or a web search engine (Karmaker Santu, Sondhi, & Zhai, 2017) to some extent. For example, a sound recommender system can recommend more suitable products to users. Prediction models in SDP are like users searching for instances that can improve their performance, while majority class instances are like products that meet the requirement of prediction models to varying degrees. Meanwhile, the undersampling technique is like a recommender system to decide which majority class instances should be recommended to prediction models. Therefore, in this study, we regard the undersampling process as a learning-to-rank task and propose a novel undersampling technique called the Learning-To-Rank UnderSampling technique (LTRUS).

Additionally, considering two imbalanced datasets, one includes majority class instances that provide redundant information to prediction models, while the other one does not. If we apply the undersampling technique to these two datasets and rigidly set the final defect ratios of both datasets as 0.5, the latter dataset will lose more information compared to the former one. Therefore, we propose the termination condition of optimizing the final defect ratio of LTRUS, instead of rigidly setting the final defect ratio as 0.5.

Another novel termination condition is also proposed. Since we optimize a linear model to calculate the relevance of majority class instances to the performance of prediction models, we optimize a threshold to determine which instances should be removed based on their output through the optimized linear model. If the output of an instance is larger than the optimized threshold, it indicates the relevance of the instance to the performance of the prediction models is high, and thus it should be preserved. Otherwise, the instance should be removed. In this condition, we do not consider the final defect ratio of the entire dataset. Instead, we focus on each individual instance of the dataset based on its relevance to the prediction models.

3. Background and related work

In this section, we introduce the background and studies related to our work.

3.1. The data resampling technique

Many data resampling techniques have been proposed to alleviate the class imbalance problem in SDP. Although RUS outperforms many oversampling techniques, the community focuses more on the oversampling technique than the undersampling technique. SMOTE (Chawla et al., 2002) randomly generates new instances along a line between a randomly selected minority class instance and one of its K minority class instances. Based on SMOTE, Han et al. (2005) proposed Borderline-SMOTE (Borderline), which improves SMOTE by only using the borderline minority class instances to generate synthetic instances. Agrawal and Menzies (2018) proposed an automatic version of SMOTE named SMOTUNED, which optimizes three hyperparameters of SMOTE automatically. These SMOTE-based oversampling techniques (Feng et al., 2021) all select the minority class instances at a close distance to generate the synthetic minority class instances, so that less noise is generated. However, the synthetic minority class instances generated by those minority class instances at a close distance are less diverse, which leads to the over-fitting problem (Bennin et al., 2017; Wong, Leung, & Ling, 2013). Bennin et al. (2017) proposed MAHAKIL to alleviate the over-fitting problem caused by the SMOTE-based oversampling techniques. MAHAKIL leverages the Mahalanobis distance to improve the diversity of the generated instances and thus overcomes the shortcoming of the SMOTE-based oversampling techniques. However, the Mahalanobis distance cannot be calculated when the number of the minority class instances is smaller than the number of their metrics, which however is common in the data used to study the class imbalance problem, limiting the application of MAHAKIL. Additionally, the prediction model trained using data preprocessed by MAHAKIL fails to retain the ability to correctly predict the minority class instances, making MAHAKIL less practical. Feng et al. (2020) proposed the complexity-based oversampling technique (COSTE), which generates diverse instances while maintaining the ability of prediction models to find the minority class instances. COSTE is designed to mitigate the class imbalance problem in SDP. Although COSTE can be applied to other fields than SDP, it will be less explainable by doing so. While the oversampling technique is important in alleviating the class imbalance problem, it has an unavoidable drawback. The instances generated by the oversampling technique are automatically classified as the minority class instances, which is not always true. A recent study conducted by Tarawneh, Hassanat, Altarawneh, and Almuhaimeed (2022) recommends stopping the use of the oversampling technique. However, our previous study (Feng et al., 2021) indicates that the oversampling technique benefits the performance of prediction models, even if some noise instances may be introduced by the oversampling technique. This is probably because common classifiers used to build prediction models have certain noise-resistant abilities (Feng et al., 2022; Kim, Zhang, Wu, & Gong, 2011), making the noise instances generated by the oversampling technique not significantly impair the performance of prediction models. Meanwhile, the oversampling technique does indeed alleviate the class imbalance problem. In aggregate, the performance of prediction models improves, indicating the oversampling technique still has a positive effect.

Regarding the undersampling technique, RUS is the most common and simplest one. It randomly removes the majority class instances, during which useful information provided by some instances may lose. Lin et al. (2017) proposed a clustering-based undersampling technique. This undersampling technique can reduce the risk of removing useful data from the majority class. The performance of this technique is better than that of RUS. This technique leverages the K-means algorithm to cluster the majority class instances and only retains the center majority

class instances to reduce the size of the majority class. However, choosing an appropriate K value for the K-means algorithm can be difficult. This technique determines the K value based on the number of minority class instances, which likely leads to a clustering result that differs from the real distribution of the data.

Agrawal and Menzies (2018), Bennin et al. (2019), Kamei, Monden, Matsumoto, Kakimoto, and Matsumoto (2007), Riquelme, Ruiz, Rodríguez, and Moreno (2008), Song et al. (2018), Tan, Tan, Dara, and Mayeux (2015), Tantithamthavorn et al. (2020), and Wang and Yao (2013) have focused on empirically investigating the performance of different data resampling techniques for SDP. In the two recent studies, Bennin et al. (2019) found that RUS outperformed SMOTE, ADASYN, Borderline, Safe-level SMOTE, and Random OverSampling (ROS) on 40 defect datasets. Tantithamthavorn et al. (2020) found that RUS can achieve higher recall, AUC, and F1-measure values than SMOTE, ROS, SMOTUNED on 101 defect datasets.

The cost-sensitive learning (Iranmehr et al., 2019; Khan, Hayat, Bennamoun, Sohel, & Togneri, 2017; Yu et al., 2019) and the ensemble learning technique (Feng et al., 2018; Liu, Wang, Zhang, Chen, & Xiang, 2017; Wu et al., 2018) are techniques used from different angles to alleviate the class imbalance problem. Some studies further combine the data resampling technique with the ensemble learning technique (Hassanat et al., 2022; Liu, Wu and Zhou, 2009). However, we focus on the studies of the data resampling technique in this work and thus only use these techniques as baselines.

3.2. Learning-to-rank

The learning-to-rank algorithm is normally used to construct a ranking model that produces a desirable rank of objects based on their importance, relevance, or preference (Liu et al., 2009; Usta, Altingovde, Ozcan, & Ulusoy, 2021; Wang, Wu, Qi, & Zhao, 2021). The learning-to-rank algorithm is commonly applied in recommender systems, search engines, answer selections, and other similar areas. The learning-to-rank algorithm can be further categorized as the pointwise algorithm, the pairwise algorithm, and the listwise algorithm. The pointwise algorithm directly predicts the number of defects or defect density of instances and ranks the instances based on their predicted values (Kamei et al., 2012; Ohlsson & Alberg, 1996). The pairwise algorithm predicts the relationship between any two instances (Nguyen, An, Hai, & Phuong, 2014; Yu et al., 2019). For example, there are three instances A, B, and C. The pairwise algorithm predicts that the rank of A is higher than that of B, and the rank of B is higher than that of C. Therefore, the final rank of A, B, and C becomes A > B > C. The listwise algorithm directly ranks all instances as a list by optimizing performance measures (Yang, Tang, & Yao, 2014).

In this study, we construct the rank of the majority class instances based on their relevance to the performance of prediction models. We preserve the instances that rank high and remove the bottom instances to alleviate the class imbalance problem.

3.3. The differential evolution algorithm

DE is an evolutionary algorithm proposed by Storn and Price (1997) that is based on the mutation, crossover and selection operations to explore the optimal solution that best fits a fitness function. The steps of DE are briefly outlined below:

First, DE initializes a group of candidate solutions. Each solution is a vector, and the range of each feature of this vector is limited by a predefined minimum and maximum bound. Then, the mutant vector is generated by perturbing the candidate vector based on a scaling factor. After the mutation operation, the crossover operation is applied to increase the diversity of the vector based on the crossover rate. Finally, the vector that best fits the fitness function is recorded as a new member of the next generation. The process is iterated until the stop criterion of DE is met, and the best-performing candidate is selected as the final solution.

4. Methodology

In this section, we describe the Learning-To-Rank UnderSampling technique (LTRUS).

4.1. LTRUS

Our goal is to build a model that can correctly rank the majority class instances so that instances less relevant to the performance of prediction models are removed from the rank to alleviate the class imbalance problem in SDP. Weyuker, Ostrand, and Bell (2010) concludes that linear models are good and realistic enough for SDP; therefore, we build a linear model for the majority class instances, given by:

$$f(X) = A \cdot X = \sum_{i=1}^{d} \alpha_i x_i, \tag{1}$$

where $X \in \mathbf{R}^{d\times 1}$ represents the feature vector of majority class instances from a dataset, x_i is the ith feature of X, d is the dimension of X, $A \in \mathbf{R}^{1\times d}$ is the weight vector, and α_i is the corresponding weight of the feature x_i to be optimized. We employ DE to optimize the weight vector. For an evolutionary algorithm, a fitness function needs to be set. In this study, we designated the fitness function to maximize the Matthews Correlation Coefficient (MCC) (Chicco & Jurman, 2020) values of prediction models. The MCC, being an unbiased performance metric, offers a superior reflection of the overall performance of prediction models (Song et al., 2018) on the whole data of each dataset. Upon determining the optimized model $f(X)$, we apply it to each majority class instance to obtain a corresponding numerical value. Subsequently, we rank these instances based on these numerical values. As the optimization of $f(X)$ is tailored to maximize the performance of prediction models, the numerical value derived from each majority class instance can provide insights into its relevance to model performance. Instances ranked higher correlate with superior model performance, while lower-ranked instances are less impactful. Removing the lower-ranked majority class instances thus results in an insignificant impact on model performance, whilst successfully addressing the class imbalance problem.

Fig. 2 shows an illustration of LTRUS, where A_N represents the candidate solution generated by DE, and A represents the final optimal solution for the linear model $f(X)$ of LTRUS. X_1, X_2, X_3, X_4, and X_5 represent the majority class instances in the dataset. Upon determining A, LTRUS performs a dot product operation on A with each instance (i.e., $A \cdot X_1$, $A \cdot X_2$, $A \cdot X_3$, $A \cdot X_4$, and $A \cdot X_5$), producing a numerical value for each. These instances, X_1, X_2, X_3, X_4, and X_5, are then ranked by LTRUS based on their respective numerical values. The colored instances in Fig. 2 represent the instances to be removed as they are ranked bottom and less relevant to the performance of prediction models.

The next subsections present the detailed steps of LTRUS.

4.1.1. Generate candidate solutions

DE in LTRUS first randomly generates N vectors $A_i^{(G)}$ at generation (G) = 0, where N is an integer hyperparameter predefined by users. These N vectors $A_i^{(G)}$ constitute a population. Each vector $A_i^{(G)}$ will be seen as a candidate solution for Eq. (1) in LTRUS. The dimension d of $A_i^{(G)}$ equals to the dimension of X.

$$A_i^{(G)} = (\alpha_{i,1}^{(G)}, \alpha_{i,2}^{(G)}, \dots, \alpha_{i,d}^{(G)}), \tag{2}$$

where $\alpha_{i,d}^{(G)}$ is each feature of $A_i^{(G)}$ to be optimized. There is a maximum bound and a minimum bound limiting the initial value of each feature of $A_i^{(G)}$. The following is the maximum bound:

$$A_{max}^{(G)} = (\alpha_{max,1}^{(G)}, \alpha_{max,2}^{(G)}, \dots, \alpha_{max,d}^{(G)}), \tag{3}$$

and the minimum bound is

$$A_{min}^{(G)} = (\alpha_{min,1}^{(G)}, \alpha_{min,2}^{(G)}, \dots, \alpha_{min,d}^{(G)}). \tag{4}$$

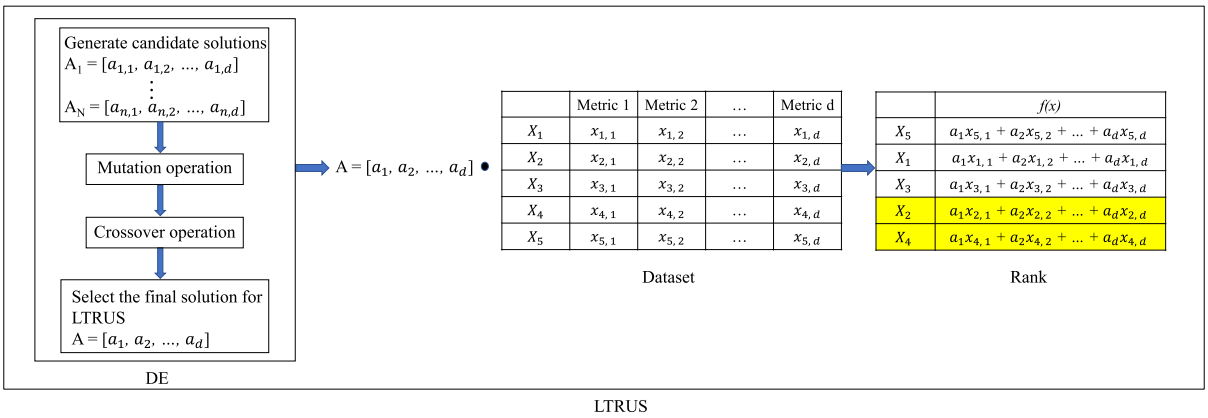


Fig. 2. The illustration of LTRUS.

The bound for each feature is the same. The value of the jth feature of $A_i^{(G)}$ is initialized according to the following equation:

$$\alpha_{i,j}^{(G)} = \alpha_{min,j}^{(G)} + rand * (\alpha_{max,j}^{(G)} - \alpha_{min,j}^{(G)}), \tag{5}$$

where $rand$ is a random value ranging from 0 to 1.

4.1.2. Operate on the population

To increase the diversity of the population, the mutation operation is applied to the vectors. The mutant vector is generated by performing operations on $A_i^{(G)}$.

$$V_i^{(G+1)} = A_{r_1}^{(G)} + F * (A_{r_2}^{(G)} - A_{r_3}^{(G)}), \tag{6}$$

where $A_{r_1}^{(G)}$, $A_{r_2}^{(G)}$, and $A_{r_3}^{(G)}$ are randomly chosen from N vectors of the population. F is the scaling factor.

Then a trial vector is introduced to further increase the diversity.

$$U_i^{(G+1)} = (u_{i,1}^{(G+1)}, u_{i,2}^{(G+1)}, \dots, u_{i,d}^{(G+1)}). \tag{7}$$

Each feature $u_{i,j}^{(G+1)}$ of the trial vector $U_i^{(G+1)}$ is calculated according to the following crossover operation:

$$u_{i,j}^{(G+1)} = \begin{cases} v_{i,j}^{(G+1)} & \text{if } (randb(j) \le \text{CR}) \text{ or } j = rnbr(i) \\ \alpha_{i,j}^{(G)} & \text{if } (randb(j) > \text{CR}) \text{ and } j \ne rnbr(i) \end{cases}. \tag{8}$$

In Eq. (8), $v_{i,j}^{(G+1)}$ is the jth feature of the ith mutant vector. CR is a hyperparameter controlling the crossover rate. $randb(j)$ is randomly generated between 0 and 1. $rnbr(i)$ is also randomly generated to decide the value of the feature $u_{i,j}^{(G+1)}$.

4.1.3. Select the final solution

Finally, if the trial vector $U_i^{(G+1)}$ performs better than $A_i^{(G)}$, it will replace $A_i^{(G)}$ as a new candidate solution for the next generation of the population. When DE converges, LTRUS will select the best-performing candidate A as the final solution and the model $f(X)$ is finally built.

4.1.4. Apply to the imbalanced dataset

Then, the majority class instances in the imbalanced dataset are fed into $f(X)$. Based on the outputs of $f(X)$, the instances are ranked in descending order and the majority class instances are removed from the bottom. The termination condition of LTRUS is the same as the conventional data resampling technique. The termination condition means that the technique is terminated once the condition is met. LTRUS terminates when the number of the majority class is equal to the number of the minority class instances (i.e., the termination condition of LTRUS). Fig. 1 shows a dataset including 5 majority class instances (A, B, C, D, and E) and 3 minority class instances (F, G, and H). The majority class instances A, B, C, D, and E are fed into $f(X)$, and then the rank of these majority class instances is generated. To achieve the balance of the dataset, we remove the bottom instances E and A. Finally, we apply the instances B, C, D, F, G, and H to classifiers to train prediction models. Notably, the optimal linear model $f(X)$ is only used to rank the majority class instances. It is not the prediction model. Additionally, the values of each feature of the majority class instances are not changed by the optimal weight α_i throughout the whole process. We can see that the dataset is balanced after applying LTRUS.

4.2. LTRUS-ratio

Compared with LTRUS, LTRUS-ratio optimizes one additional parameter (i.e., the final defect ratio) as the termination condition. Unlike LTRUS, whose termination condition is that the number of majority class instances is the same as the number of minority class instances (i.e., the dataset is perfectly balanced), LTRUS-ratio removes the majority class instances from the bottom of the rank until the final defect ratio meets the optimized defect ratio. Then, LTRUS-ratio is terminated.

4.3. LTRUS-thres

LTRUS-thres optimizes another parameter (i.e., the threshold) as the termination condition, which is different from LTRUS-ratio. When each majority class instance is fed into $f(X)$, its corresponding output from $f(X)$ is obtained. If the output of a given majority class instance is greater than the optimized threshold, the instance is preserved. Otherwise, it is removed.

4.4. Hyperparameter configuration

In LTRUS, we set the number N of the initial candidate solutions as ten times the dimension d of an instance by convention. The number G of the generations is set as ten, as convergence is achieved on all classifiers with no more than ten generations. The maximum bound $\alpha_{max,d}$ and the minimum bound $\alpha_{min,d}$ are respectively set as 1 and -1. Different combinations of the scaling factor F and the crossover rate CR are tried. Experimentally, we set the scaling factor as 0.3 and the crossover rate as 0.9 to achieve satisfactory performance with the least execution time. Besides the hyperparameters in LTRUS, we set the range of the final defect ratio to be optimized from 0 to 1 in LTRUS-ratio and the range of the threshold to be optimized from $-d$ to d in LTRUS-thres, where d is the vector dimension of an instance. This setting is because we perform the min–max normalization across each feature vector of instances, which means the outputs of $f(X)$ will be neither larger than d nor smaller than $-d$. The details of the hyperparameter configuration are shown in Table 1.

Table 1
Hyperparameter configuration.

Hyperparameters	Values
The candidate solutions, N	$10 \times d$
The number of generations, G	10
The scaling factor, F	0.3
The crossover rate, CR	0.9
The minimum bound $\alpha_{min,d}$	-1
The maximum bound $\alpha_{max,d}$	1
The final defect ratio for LTRUS-ratio	(0, 1)
The threshold for LTRUS-thres	$(-d, d)$

5. Experimental design

This section presents the baseline techniques, the information of the datasets, the classifiers, the performance measures, the detailed experimental procedure, and the statistical test.

5.1. Baselines

To investigate the performance of LTRUS for SDP, we compare it with five baseline techniques, i.e., RUS, Cluster, COSTE, SMOTUNED, and Borderline. RUS and Borderline are widely used as baseline data resampling techniques in SDP (Bennin et al., 2017; Feng et al., 2020), while SMOTUNED and COSTE are recently proposed oversampling techniques for SDP. SMOTE is the most common baseline in the field of the class imbalance problem. However, since we adopt SMOTUNED, which is the automatic version of SMOTE, we do not adopt SMOTE in this study. The following is a brief introduction to the baseline techniques.

RUS. RUS is an undersampling technique that randomly removes instances from the majority class and keeps the minority class instances unchanged until the balance of a dataset is achieved.

Cluster. Cluster leverages the K-means algorithm to cluster the majority class instances and only reserves the center majority class instances for each cluster to reduce the size of the majority class. The number of K in the K-means algorithm is set to be equal to the number of the minority class instances.

COSTE. COSTE leverages the complexity instead of the distance to aid in selecting the instances used to generate synthetic instances. The complexity is calculated as the weighted sum of each feature. DE is employed to explore the optimal weight of the features.

SMOTUNED. There are three hyperparameters in SMOTE: the number K of the nearest neighbor minority class instances, the power parameter of the Minkowski distance metric, and the final defect ratio in a dataset. SMOTUNED employs DE to explore the optimal values of these three hyperparameters. Since SMOTUNED is the automatic version of SMOTE, we adopt SMOTUNED instead of SMOTE as the baseline.

Borderline. Borderline puts more focus on the borderline instances, because these instances are more informative. Borderline first randomly selects one of the borderline instances. Then one of the K nearest neighbor minority class instances of the selected borderline instance is selected, and one synthetic instance is randomly generated on the line between the two selected instances. This procedure is iterated until the number of instances for both the minority and majority classes are the same.

COSTE and SMOTUNED both employ DE to explore the optimal values of the parameters. The fitness functions of DE in the original COSTE and SMOTUNED are set to maximize the AUC values. In this study, we modify the fitness functions of these two techniques to maximize the MCC values, as LTRUS does, to make a fair comparison.

5.2. Datasets

To ensure the generalizability of our experimental result, we utilized 21 imbalanced datasets from the AEEEM (D'Ambros et al., 2012), ReLink (Wu et al., 2011), PROMISE (Menzies et al., 2012), SOFTLAB (Turhan, Menzies, Bener, & Di Stefano, 2009), and NASA repositories (Shepperd et al., 2013). For the PROMISE repository, there are multiple versions for each project. We only used the first version of each project (Gong, Jiang, Wang, & Jiang, 2019).

The AEEEM datasets were collected by D'Ambros et al. These datasets contain the Chidamber and Kemerer (CK) metrics (Chidamber & Kemerer, 1994), the object-oriented (OO) metrics (Basili, Briand, & Melo, 1996), the metric of the previous defect number (Kim, Zimmermann, Whitehead, & Zeller, 2007), the change metrics (Moser, Pedrycz, & Succi, 2008), the complexity code change metrics (Hassan, 2009), and the churn and entropy of the CK and OO metrics (D'Ambros, Lanza, & Robbes, 2010). There are 61 metrics in the AEEEM datasets. The ReLink datasets were collected by Wu et al. These datasets contain 26 complexity metrics (e.g. *lines of code* and *number of classes*). The Promise datasets contain the CK metrics and the OO metrics. The number of metrics in the Promise datasets is 21. The NASA and the SOFTLAB datasets were collected from the National Aeronautics and Space Administration (NASA) and a Turkish software company, respectively. They share the same metrics which are Halstead (Halstead, 1977) and McCabe's cyclomatic metrics (McCabe, 1976). In addition, the NASA datasets contain additional metrics such as *parameter count* and *percentage of comments*. We used the cleaned version (Shepperd et al., 2013) of the NASA datasets since previous studies have pointed out that there are mislabeled instances in the original version.

We present the details of the datasets we utilized in Table 2, where # Instances represents the number of instances in a dataset, and % Defect ratio is the defect ratio of a dataset.

5.3. Classifiers

In this study, we adopt four commonly-used classifiers in SDP. Specifically, K-nearest neighbor (K-NN), Random Forest (RF), Logistic Regression (LR), and Naive Bayes (NB) are adopted. Many researchers (Feng et al., 2020; Gong et al., 2019) adopted these classifiers to build prediction models in SDP. We implement these classifiers using the Sklearn package (Pedregosa et al., 2011) to avoid reinventing the wheel.

5.4. Performance measures

Performance measures (e.g., F-measure, the Area Under the ROC Curve (AUC), the Matthews Correlation Coefficient (MCC), and *balance*) not significantly impacted by the class imbalance problem are preferable in SDP. Generally, defective (i.e., minority class) instances are regarded as positive, while non-defective (i.e., majority class) instances are regarded as negative in SDP. For the predicted outcomes of a prediction model, there are four categories, namely the number of correctly predicted positive instances (TP), the number of correctly predicted negative instances (TN), the number of instances that are positive but predicted as negative (FN), and the number of instances that are negative but predicted as positive (FP). Then, the performance measures are calculated based on the outcomes of the confusion matrix (Table 3). In this study, we adopt Recall, Precision, F-measure, AUC, and MCC as performance measures to reflect the performance of prediction models on the whole data of each dataset.

$$Recall = \frac{TP}{TP + FN}, \tag{9}$$

$$Precision = \frac{TP}{TP + FP}, \tag{10}$$

Table 2
Description of 21 datasets.

Group	Dataset	Language	Granularity	Number of metrics	# Instances	% Defect ratio
NASA	CM1	C	Function	37	327	12.84
NASA	MW1	C	Function	37	253	10.67
NASA	PC1	C	Function	37	705	8.65
NASA	PC3	C	Function	37	1077	12.44
NASA	PC4	C	Function	37	1287	13.75
SOFTLAB	AR1	C	Function	29	121	7.44
SOFTLAB	AR3	C	Function	29	163	12.70
SOFTLAB	AR4	C	Function	29	107	18.69
SOFTLAB	AR5	C	Function	29	36	22.22
SOFTLAB	AR6	C	Function	29	101	14.85
AEEEM	Equinox Framework	Java	Class	61	324	39.81
AEEEM	Eclipse JDT core	Java	Class	61	997	20.66
AEEEM	Mylyn	Java	Class	61	1862	13.16
AEEEM	Eclipse PDE UI	Java	Class	61	1497	13.96
PROMISE	ant1.3	Java	Class	20	125	16.00
PROMISE	camel1.0	Java	Class	20	339	3.83
PROMISE	jedit3.2	Java	Class	20	272	33.09
PROMISE	log4j1.0	Java	Class	20	135	25.19
PROMISE	xalan2.4	Java	Class	20	723	15.21
ReLink	OpenIntents Safe	Java	File	26	56	39.29
ReLink	ZXing	Java	File	26	399	29.57

Table 3
Confusion matrix.

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

$$F-measure = \frac{(\beta^2+1)\times Precision \times Recall}{\beta^2 \times Precision + Recall}, \tag{11}$$

$$MCC = \frac{TP\times TN - FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{12}$$

In Eq. (11), we set β as 1 to make Eq. (11) become F1-measure. The higher values of F-measure, MCC, and AUC represent the better performance of prediction models.

5.5. Experimental procedure

The whole experimental procedure is presented in Fig. 3. We run the steps in the outermost dotted box 10 times.

(1) First, the min–max method is applied to each dataset to adjust the range of the features of each instance into 0 to 1, which alleviates the negative impact of the different magnitudes of the features.

(2) Next, the 5-fold cross-validation with the stratification method is employed to divide the dataset into five folds. The stratification method is selected to ensure that the defect ratio in each fold is the same as the original data.

(3) Four folds are used as the training data, and RUS, Cluster, and Borderline are applied only to those four folds. The left fold is used as the testing data to validate the performance of the techniques.

(4) For COSTE, SMOTUNED, LTRUS, LTRUS-ratio, and LTRUS-thres, the four training folds are further divided into five sub-folds.

(5) The four sub-folds of the five sub-folds are used as the sub-training data. DE is then applied. Specifically, DE first generates candidate solutions for COSTE, SMOTUNED, LTRUS, LTRUS-ratio, and LTRUS-thres, respectively. The generated candidate solutions for each technique are then applied to the sub-training data. Next, the sub-training data processed by the candidate solutions is used to train one classifier adopted in this study (i.e., the KNN, RF, LR, and NB classifiers) and build the prediction model. The sub-testing data is used to validate the performance of the prediction model in terms of MCC. For each candidate solution, this procedure is repeated five times to ensure all sub-folds are used to train the classifier and validate the performance of the prediction model. For each iteration of a candidate solution, an MCC value of the prediction model validated by the sub-testing data is obtained. After five iterations, the average MCC value for each candidate solution is calculated. The candidate solution achieving the highest average MCC value is selected by DE as the optimal parameters for each technique. Notably, both the sub-training data and the sub-testing data are generated by dividing the training data. DE is only employed in this step and is only applied to the training data (Agrawal & Menzies, 2018).

(6) Once the optimal parameters of COSTE, SMOTUNED, LTRUS, LTRUS-ratio, and LTRUS-thres are found, these techniques are applied to the four training folds.

(7) The processed training data is then used to train a new classifier, which is the same kind of the classifier selected in Step 5 but not the same one, to build the prediction model. The left testing fold is used to obtain different performance measures of the prediction model. The above procedure is repeated five times to ensure each fold is used as both the training and testing data. Five results of each performance measure are obtained.

(8) The average value of each performance measure is then calculated as one outcome.

(9) For the combination of each technique and each classifier on each dataset, the steps of 1 to 7 are further repeated ten times to reduce the variance and bias. Ten results of each performance measure for each dataset are obtained. Lastly, the average value of each performance measure is calculated based on the ten results, and these performance measures are used as the evaluation criteria to compare the performance of each technique. Because 21 datasets are selected to conduct experiments, 21 results are obtained for each combination of each technique and each classifier. These results are presented in Section 6. The default hyperparameters are adopted for all the baseline techniques.

5.6. Statistical test

To statistically compare the performance of the techniques, we employ the Wilcoxon signed-rank test (Wilcoxon) (Rey & Neuhäuser, 2011). Wilcoxon is commonly used for pairwise comparison. It is a non-parametric test. The null hypothesis of Wilcoxon is that the values are paired and collected from the same distribution. In this study, we adopt a confidence level of 95%. If the p-value is less than 0.05, the null hypothesis is rejected, indicating that the pairwise values are

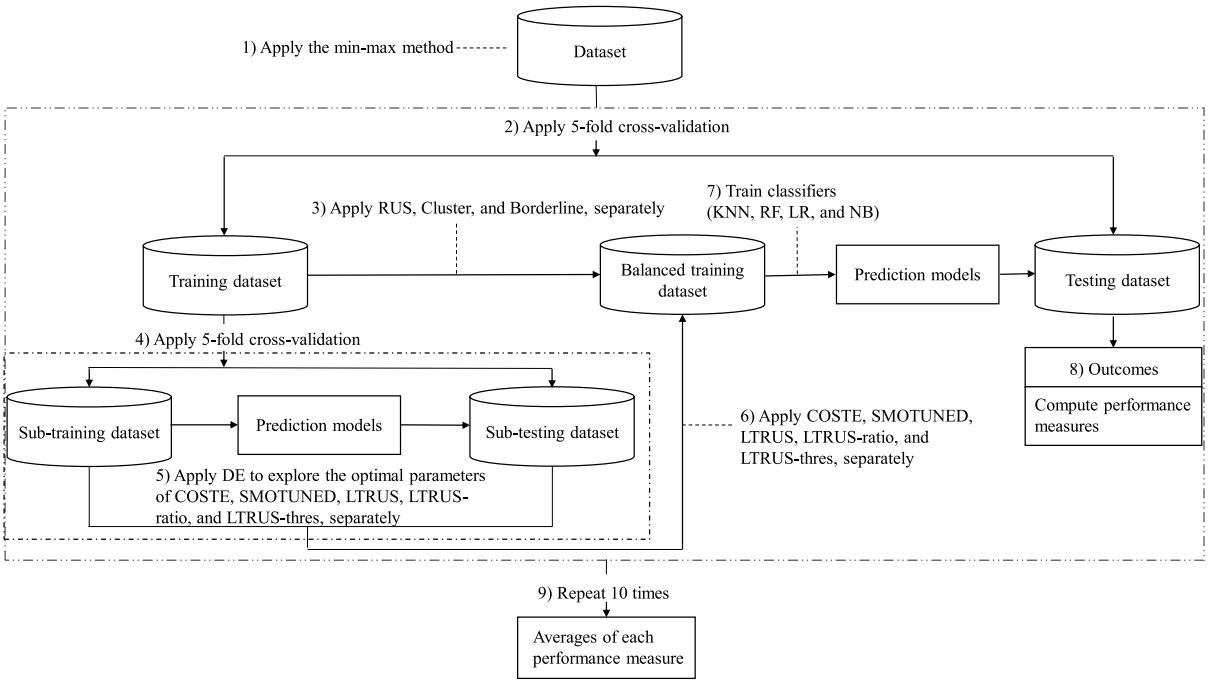


Fig. 3. Experimental procedure.

Table 4
The performance of LTRUS, RUS, COSTE, SMOTUNED, Borderline, and Cluster on the selected classifiers across 21 datasets in terms of Recall.

	LTRUS	RUS	COSTE	SMOTUNED	Borderline	Cluster
KNN	**0.821**	0.666	0.678	0.650	0.631	0.659
p-value		**<.05**	**<.05**	**<.05**	**<.05**	**<.05**
Cliff's δ		0.800	0.832	0.633	0.837	0.712
RF	**0.810**	0.667	0.449	0.496	0.431	0.693
p-value		**<.05**	**<.05**	**<.05**	**<.05**	**<.05**
Cliff's δ		0.909	0.995	0.946	0.995	0.628
LR	**0.813**	0.675	0.661	0.591	0.631	0.687
p-value		**<.05**	**<.05**	**<.05**	**<.05**	**<.05**
Cliff's δ		0.628	0.660	0.692	0.694	0.605
NB	**0.679**	0.534	0.514	0.516	0.506	0.604
p-value		**<.05**	**<.05**	**<.05**	**<.05**	>.05
Cliff's δ		0.488	0.528	0.512	0.537	0.152

Table 5
The performance of LTRUS, RUS, COSTE, SMOTUNED, Borderline, and Cluster on the selected classifiers across 21 datasets in terms of Precision.

	LTRUS	RUS	COSTE	SMOTUNED	Borderline	Cluster
KNN	0.375	0.353	**0.420**	0.390	0.379	0.316
p-value		>.05	**<.05**	>.05	>.05	>.05
Cliff's δ		0.007	0.211	0.102	0.093	0.166
RF	0.358	0.369	0.462	**0.473**	0.450	0.320
p-value		**<.05**	**<.05**	**<.05**	**<.05**	>.05
Cliff's δ		0.075	0.338	0.379	0.329	0.170
LR	0.386	0.372	0.430	**0.437**	0.390	0.381
p-value		>.05	**<.05**	>.05	>.05	>.05
Cliff's δ		0.002	0.152	0.168	0.066	0.043
NB	**0.444**	0.398	0.434	0.421	0.402	0.377
p-value		**<.05**	>.05	>.05	**<.05**	>.05
Cliff's δ		0.179	0.016	0.052	0.147	0.179

from different distributions, and that there exists statistical significance between the pairwise values. Otherwise, the null hypothesis cannot be rejected.

Besides, we employ the effect size (i.e., Cliff's δ) (Macbeth, Razumiejczyk, & Ledesma, 2011) to quantify the difference between different techniques. We follow the way that Kampenes, Dybå, Hannay, and Sjøberg (2007) interprets the effect size. The effect size is negligible ($0 < \text{Cliff's } \delta < 0.147$), small ($0.147 < \text{Cliff's } \delta < 0.33$), medium ($0.33 < \text{Cliff's } \delta < 0.474$) or large ($\text{Cliff's } \delta > 0.474$), respectively.

Finally, we employ the win-draw-loss strategy. We record the MCC values of each technique and present whether LTRUS performs better or worse than every other technique across each dataset.

6. Experimental results

In this section, we present the experimental results by answering the research question.

6.1. RQ1: How is the performance of LTRUS compared with the baselines

Table 4 shows the Recall values of each technique, with LTRUS performing the best across all four classifiers. It significantly outperforms all the baselines with practical effect sizes. The only exception is that the difference between the performance of LTRUS and Cluster is not significant on the NB classifier, although LTRUS still outperforms Cluster. The high Recall values of LTRUS reflect that LTRUS is better at finding minority class instances than the baseline techniques.

Generally, there is a trade-off between Recall and Precision (Buckland & Gey, 1994). High Recall values often lead to low Precision values. As expected, Table 5 shows that LTRUS performs worse than the three oversampling techniques in terms of Precision on the KNN, RF, and LR classifiers. Compared with RUS and Cluster, the performance of LTRUS is better. Especially, LTRUS outperforms all the baselines on the NB classifier in terms of Precision.

Table 6 shows that LTRUS performs well in terms of F-measure, with the best performance on the RF and NB classifiers. On the KNN and LR classifier, While it fails to outperform COSTE on the KNN and LR classifiers, the difference between the performance of LTRUS and COSTE on these two classifiers is insignificant. Especially on the NB classifier, LTRUS significantly outperforms all other baselines with the practical effect sizes. A similar trend is observed from Table 7, with LTRUS performing the best on the RF, LR, and NB classifiers in terms of AUC. It significantly outperforms RUS, SMOTUNED, Borderline, and Cluster on all classifiers with practical effect sizes.

Recent studies show that AUC and F-measure are biased (Song et al., 2018). Instead, MCC is unbiased. Therefore, we adopt MCC to obtain

Table 6
The performance of LTRUS, RUS, COSTE, SMOTUNED, Borderline, and Cluster on the selected classifiers across 21 datasets in terms of F-measure.

	LTRUS	RUS	COSTE	SMOTUNED	Borderline	Cluster
KNN	0.470	0.428	**0.495**	0.455	0.455	0.391
p-value		<.**05**	>.05	>.05	>.05	>.05
Cliff's δ		0.125	0.079	0.070	0.007	0.234
RF	**0.464**	0.446	0.437	**0.464**	0.422	0.396
p-value		<.**05**	<.**05**	>.05	<.**05**	>.05
Cliff's δ		0.057	0.061	0.029	0.152	0.259
LR	0.474	0.444	**0.487**	0.466	0.455	0.439
p-value		<.**05**	>.05	>.05	>.05	>.05
Cliff's δ		0.093	0.066	0.011	0.075	0.147
NB	**0.493**	0.410	0.431	0.422	0.412	0.374
p-value		<.**05**	<.**05**	<.**05**	<.**05**	<.**05**
Cliff's δ		0.338	0.252	0.274	0.302	0.397

Table 7
The performance of LTRUS, RUS, COSTE, SMOTUNED, Borderline, and Cluster on the selected classifiers across 21 datasets in terms of AUC.

	LTRUS	RUS	COSTE	SMOTUNED	Borderline	Cluster
KNN	0.730	0.676	**0.732**	0.704	0.694	0.646
p-value		<.**05**	>.05	<.**05**	<.**05**	<.**05**
Cliff's δ		0.442	0.039	0.234	0.279	0.596
RF	**0.735**	0.704	0.669	0.684	0.659	0.659
p-value		<.**05**	<.**05**	<.**05**	<.**05**	<.**05**
Cliff's δ		0.320	0.565	0.410	0.646	0.433
LR	**0.743**	0.703	0.731	0.702	0.704	0.691
p-value		<.**05**	>.05	<.**05**	<.**05**	<.**05**
Cliff's δ		0.302	0.052	0.279	0.270	0.388
NB	**0.729**	0.663	0.674	0.666	0.658	0.628
p-value		<.**05**	<.**05**	<.**05**	<.**05**	<.**05**
Cliff's δ		0.546	0.388	0.460	0.519	0.678

an unbiased and more objective conclusion. We use the win-draw-loss strategy to record the MCC values of LTRUS and the baselines across every single dataset. It can be seen from Tables 8, 9, 10, 11 that LTRUS consistently obtains positive win-loss values against RUS, SMOTUNED, Borderline, and Cluster on all classifiers. The poor performance of RUS is due to the equal treatment between majority and minority class instances when balancing the dataset. Cluster performs unsatisfactorily because setting the K value of the K-means algorithm equal to the number of the minority class instances cannot reflect the real distribution of the datasets, leading to removing informative instances instead of reserving them. The inferior performance of SMOTUNED and Borderline is due to the introduced noise when synthesizing instances. Comparing LTRUS and COSTE, their performances are similar on the KNN, RF, and LR classifiers, where there is no significant difference between the performance of LTRUS and COSTE in terms of MCC. Compared with SMOTUNED and Borderline, COSTE generates high-quality minority instances. However, it still has a chance to introduce noise. And LTRUS might lose information when removing majority instances. Therefore, LTRUS and COSTE are comparative in most classifiers. But the number of noise introduced by COSTE is beyond the noise-resistant ability (Feng et al., 2022; Kim et al., 2011) of the NB classifier, leading to significant degradation in performance. As a result, LTRUS overwhelms COSTE.

> ***Finding 1:*** *The performance of LTRUS is 18.0% better than all the baselines on average in terms of MCC, reflecting that LTRUS indeed preserves more instances that contribute more to prediction models than RUS and Cluster, while still keeping its advantage of not introducing any noise over the oversampling technique.*

Table 8
MCC values of each technique on the KNN classifier across 21 datasets.

KNN	LTRUS	RUS	COSTE	SMOTUNED	Borderline	Cluster
EQ	0.531	0.408	0.483	0.541	0.423	0.344
JDT	0.498	0.471	0.536	0.481	0.475	0.342
ML	0.314	0.287	0.366	0.324	0.311	0.125
PDE	0.290	0.252	0.303	0.251	0.262	0.125
CM1	0.245	0.164	0.270	0.292	0.161	0.126
MW1	0.333	0.239	0.332	0.305	0.365	0.186
PC1	0.259	0.191	0.319	0.258	0.247	0.171
PC3	0.333	0.314	0.373	0.334	0.324	0.286
PC4	0.335	0.317	0.459	0.445	0.420	0.304
ant1.3	0.361	0.247	0.353	0.204	0.314	0.265
camel1.0	0.104	0.051	0.097	0.082	0.061	0.027
jedit3.2	0.502	0.451	0.524	0.499	0.484	0.458
log4j1.0	0.433	0.348	0.550	0.462	0.474	0.243
xalan2.4	0.262	0.266	0.350	0.275	0.268	0.169
Safe	0.513	0.358	0.432	0.393	0.397	0.357
ZXing	0.302	0.258	0.276	0.253	0.252	0.025
AR1	0.151	−0.153	0.313	0.146	0.155	−0.03
AR3	0.717	0.645	0.519	0.526	0.403	0.563
AR4	0.465	0.369	0.425	0.358	0.275	0.338
AR5	0.614	0.605	0.579	0.612	0.664	0.652
AR6	0.255	0.123	0.344	0.224	0.194	−0.003
Average	0.372	0.296	0.391	0.346	0.330	0.241
p-value		<.**05**	>.05	>.05	<.**05**	<.**05**
Cliff's δ		0.256	0.161	0.111	0.156	0.401
W/D/L		20/0/1	9/0/12	14/0/7	15/0/6	20/0/1

Table 9
MCC values of each technique on the RF classifier across 21 datasets.

RF	LTRUS	RUS	COSTE	SMOTUNED	Borderline	Cluster
EQ	0.576	0.514	0.517	0.536	0.516	0.568
JDT	0.468	0.475	0.506	0.496	0.503	0.186
ML	0.307	0.329	0.336	0.353	0.352	0.009
PDE	0.252	0.264	0.281	0.297	0.270	0.009
CM1	0.280	0.169	0.114	0.133	0.084	0.161
MW1	0.229	0.213	0.308	0.287	0.283	0.082
PC1	0.300	0.289	0.286	0.257	0.272	0.259
PC3	0.369	0.327	0.285	0.323	0.295	0.343
PC4	0.481	0.489	0.512	0.551	0.506	0.479
ant1.3	0.359	0.320	0.273	0.379	0.255	0.304
camel1.0	0.152	0.133	0.054	0.050	0.188	0.068
jedit3.2	0.527	0.490	0.495	0.497	0.492	0.541
log4j1.0	0.423	0.388	0.434	0.326	0.310	0.317
xalan2.4	0.327	0.295	0.273	0.301	0.244	0.202
Safe	0.471	0.461	0.496	0.454	0.499	0.477
ZXing	0.294	0.222	0.273	0.264	0.237	−0.013
AR1	0.207	0.107	0.141	0.139	0.097	0.206
AR3	0.396	0.387	0.364	0.517	0.353	0.266
AR4	0.411	0.405	0.397	0.514	0.383	0.351
AR5	0.583	0.476	0.554	0.617	0.401	0.565
AR6	0.256	0.214	0.288	0.401	0.251	−0.016
Average	0.365	0.332	0.342	0.366	0.323	0.256
p-value		<.**05**	>.05	>.05	<.**05**	>.05
Cliff's δ		0.125	0.066	0.043	0.179	0.356
W/D/L		17/0/4	13/0/8	11/0/10	14/0/7	19/0/2

6.2. RQ2: How is the performance of LTRUS compared with LTRUS-ratio and LTRUS-thres

Fig. 4 presents the boxplots of the performance of each technique in terms of MCC values. The black triangle represents the mean value, and the black line represents the median value in Fig. 4. We can clearly see that LTRUS-ratio and LTRUS-thres significantly outperform all the other techniques, including LTRUS, regarding the maximum, median, mean, and minimum MCC values. Fig. 4 shows that removing the majority class instances that optimize the threshold or optimize the final defect ratio as the termination condition instead of setting the final defect ratio strictly as 0.5 is effective to improve the undersampling technique.

Table 10
MCC values of each technique on the LR classifier across 21 datasets.

LR	LTRUS	RUS	COSTE	SMOTUNED	Borderline	Cluster
EQ	0.537	0.480	0.492	0.483	0.477	0.457
JDT	0.544	0.504	0.529	0.544	0.520	0.528
ML	0.334	0.326	0.350	0.343	0.340	0.120
PDE	0.298	0.284	0.293	0.318	0.259	0.082
CM1	0.274	0.204	0.247	0.216	0.222	0.141
MW1	0.284	0.310	0.337	0.363	0.327	0.287
PC1	0.285	0.250	0.288	0.284	0.259	0.213
PC3	0.357	0.313	0.345	0.318	0.327	0.270
PC4	0.397	0.402	0.470	0.509	0.448	0.380
ant1.3	0.431	0.379	0.439	0.420	0.456	0.364
camel1.0	0.142	0.082	0.207	0.201	0.247	0.138
jedit3.2	0.549	0.518	0.564	0.535	0.486	0.533
log4j1.0	0.485	0.425	0.518	0.447	0.483	0.359
xalan2.4	0.279	0.308	0.327	0.278	0.279	0.257
Safe	0.469	0.388	0.467	0.313	0.465	0.491
ZXing	0.247	0.156	0.153	0.123	0.133	−0.076
AR1	0.194	0.041	0.119	−0.035	−0.006	0.212
AR3	0.523	0.427	0.563	0.641	0.502	0.641
AR4	0.472	0.431	0.507	0.394	0.308	0.356
AR5	0.723	0.534	0.612	0.612	0.446	0.666
AR6	0.298	0.224	0.384	0.321	0.234	0.233
Average	0.387	0.333	0.391	0.363	0.343	0.317
p-value		**<.05**	>.05	>.05	**<.05**	>.05
Cliff's δ		0.166	0.066	0.029	0.166	0.252
W/D/L		18/0/3	9/0/12	13/1/7	15/1/5	17/0/4

Table 11
MCC values of each technique on the NB classifier across 21 datasets.

NB	LTRUS	RUS	COSTE	SMOTUNED	Borderline	Cluster
EQ	0.534	0.420	0.421	0.430	0.422	0.423
JDT	0.523	0.467	0.479	0.475	0.483	0.470
ML	0.313	0.259	0.275	0.263	0.263	−0.015
PDE	0.348	0.291	0.304	0.305	0.296	0.184
CM1	0.243	0.159	0.184	0.169	0.118	0.028
MW1	0.345	0.269	0.301	0.248	0.263	0.050
PC1	0.335	0.269	0.296	0.276	0.295	0.308
PC3	0.373	0.225	0.200	0.136	0.115	−0.013
PC4	0.453	0.328	0.450	0.424	0.427	0.224
ant1.3	0.447	0.341	0.389	0.374	0.304	0.352
camel1.0	0.221	0.160	0.292	0.282	0.156	0.129
jedit3.2	0.502	0.437	0.441	0.440	0.391	0.381
log4j1.0	0.527	0.463	0.476	0.451	0.463	0.460
xalan2.4	0.354	0.257	0.282	0.279	0.237	0.266
Safe	0.434	0.311	0.321	0.399	0.386	0.375
ZXing	0.232	0.123	0.184	0.197	0.129	−0.073
AR1	0.234	−0.008	−0.091	−0.104	−0.072	0.118
AR3	0.537	0.319	0.526	0.425	0.410	0.236
AR4	0.418	0.370	0.405	0.373	0.432	0.428
AR5	0.724	0.585	0.543	0.543	0.530	0.693
AR6	0.276	0.205	0.219	0.223	0.196	0.068
Average	0.399	0.298	0.328	0.315	0.297	0.243
p-value		**<.05**	**<.05**	**<.05**	**<.05**	**<.05**
Cliff's δ		0.406	0.261	0.279	0.351	0.465
W/D/L		21/0/0	20/0/1	20/0/1	20/0/1	20/0/1

Finding 2: *LTRUS-ratio and LTRUS-thres both perform 9.7% better than LTRUS in terms of MCC on average. By optimizing the final defect ratio or the threshold, the performance of the undersampling technique could be improved. The conventional termination condition of the data resampling technique, where the final defect ratio reaches 0.5, is improper.*

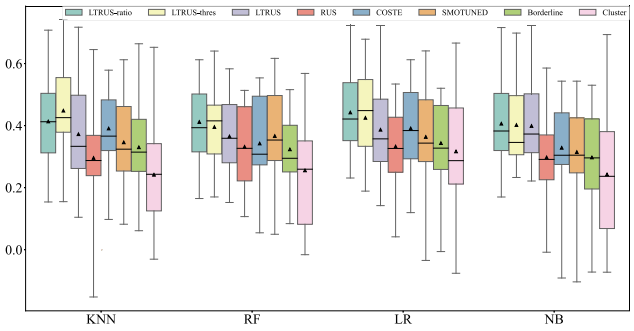


Fig. 4. The boxplots of the MCC values of LTRUS, LTRUS-ratio, LTRUS-thres, RUS, COSTE, SMOTUNED, and Borderline.

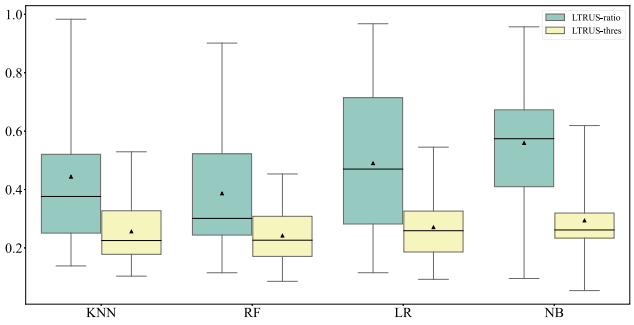


Fig. 5. The defect ratios of the 21 datasets after being processed by LTRUS-ratio and LTRUS-thres.

6.3. RQ3: How is the comparison between LTRUS-ratio and LTRUS-thres

Fig. 4 shows that LTRUS-ratio and LTRUS-thres perform similarly. The median MCC values of LTRUS-thres are higher than those of LTRUS-ratio on the KNN, RF and LR classifiers. The mean MCC values of LTRUS-ratio are higher than those of LTRUS-thres on the RF, LR and NB classifiers. The average MCC values of LTRUS-ratio and LTRUS-thres are the same across the four classifiers, at 0.418.

We also present the final defect ratios of the 21 datasets after they were processed by LTRUS-ratio and LTRUS-thres. Fig. 5 shows that the defect ratios of the 21 datasets processed by LTRUS-ratio are generally larger than those processed by LTRUS-thres. The average final defect ratios of LTRUS-ratio are 44.38%, 38.65%, 48.99%, and 55.95% on the KNN, RF, LR, and NB classifiers. Meanwhile, the defect ratios of LTRUS-thres are only 25.62%, 24.20%, 27.06%, and 29.38% on the KNN, RF, LR, and NB classifiers respectively. Table 12 shows the defect ratio of every single dataset after being processed by LTRUS-ratio and LTRUS-thres. "NONE" in Table 12 represents the original defect ratio of datasets without any processing. We can see from Table 12 that almost all datasets processed by LTRUS-ratio have a higher defect ratio than those corresponding ones processed by LTRUS-thres, with the exception of only three datasets. LTRUS-ratio removes more majority class instances than LTRUS-thres, which means LTRUS-ratio could mitigate more negative impact of the class imbalance problem than LTRUS-thres. Meanwhile, LTRUS-thres removes fewer instances than LTRUS-ratio while achieving similar performance to LTRUS-ratio, indicating that LTRUS-thres reserves more information provided by the remaining instances.

Finding 3: *The similar performance, but the different defect ratios, shows that different strategies adopted by LTRUS-ratio and LTRUS-thres benefit the undersampling technique from different perspectives. Finding an effective way to combine these strategies could further improve the undersampling technique.*

Table 12
The defect ratio (%) of the 21 datasets after being processed by LTRUS-ratio and LTRUS-thres.

	NONE	KNN LTRUS-ratio	KNN LTRUS-thres	RF LTRUS-ratio	RF LTRUS-thres	LR LTRUS-ratio	LR LTRUS-thres	NB LTRUS-ratio	NB LTRUS-thres
EQ	39.81	98.32	52.92	90.15	45.33	92.06	54.51	95.73	52.76
JDT	20.66	40.26	25.79	27.98	28.29	43.32	28.42	69.34	31.60
ML	13.16	26.10	18.05	21.69	22.64	43.39	23.42	33.43	22.49
PDE	13.96	23.85	17.01	18.67	26.17	28.19	18.60	57.41	25.27
CM1	12.84	37.62	28.66	52.23	33.81	47.01	26.40	51.09	24.15
MW1	10.67	13.83	11.54	14.51	11.78	22.14	15.13	36.11	12.75
PC1	8.65	20.49	10.33	11.49	8.65	11.50	9.24	25.37	15.23
PC3	12.44	25.06	18.45	35.75	22.55	49.25	25.89	57.52	19.61
PC4	13.75	21.96	17.63	32.47	19.10	24.95	17.01	38.12	23.36
ant1.3	16.00	48.79	22.22	35.22	23.48	51.45	33.86	42.85	26.16
camel1.0	3.83	14.99	12.73	28.13	9.87	16.89	11.10	9.52	5.38
jedit3.2	33.09	60.28	36.40	69.81	35.92	72.40	39.40	85.06	41.52
log4j1.0	25.19	52.06	34.97	48.43	29.22	53.58	32.32	61.58	29.81
xalan2.4	15.21	33.06	22.52	27.16	19.91	32.33	24.89	60.67	26.15
Safe	39.29	94.72	44.22	70.09	41.02	77.44	47.07	95.65	51.35
ZXing	29.57	72.00	41.48	68.48	38.21	96.78	44.44	90.70	61.91
AR1	7.44	35.28	17.81	24.32	10.38	18.00	14.12	47.54	24.24
AR3	12.70	44.95	19.35	30.11	15.31	40.35	18.69	40.97	25.00
AR4	18.69	34.87	23.73	26.53	18.65	55.88	25.11	47.50	26.36
AR5	22.22	81.37	32.72	54.05	30.83	80.32	32.61	67.30	39.92
AR6	14.85	52.07	29.41	24.39	17.10	71.46	26.13	61.45	31.94
Average	18.29	44.38	25.62	38.65	24.20	48.99	27.06	55.95	29.38

7. Discussion

7.1. The analysis of LTRUS-ratio and LTRUS-thres

As stated in Sections 4.2 and 4.3, LTRUS-ratio optimizes the final defect ratio and does not directly operate on majority class instances. Whereas LTRUS-thres operates on the data level. Although the performances of LTRUS-ratio and LTRUS-thres are similar, LTRUS-ratio removes more majority class instances than LTRUS-thres, indicating that (1) the defect ratio impacts the performance of prediction models from a global view, (2) the information provided by each instance impacts the performance of prediction models from a local view, and (3) there is a trade-off between the global view (i.e., optimizing the final defect ratio to balance datasets as much as possible) and the local view (i.e., preserving as many instances that could provide prediction models with more information). To effectively alleviate the class imbalance problem, LTRUS-ratio removes extra majority class instances to achieve the balance between the majority class and the minority class instances. However, prediction models also lose the information that could have been provided by these majority class instances but additionally removed by LTRUS-ratio. As a comparison, LTRUS-thres removes fewer instances, providing prediction models with more information. But the class imbalance problem has a more negative impact on the performance of LTRUS-thres, leading to a similar performance to LTRUS-ratio.

7.2. The time consumption analysis of LTRUS

Because COSTE, SMOTUNED, and LTRUS all utilize DE to optimize parameters, their runtime is longer than RUS, Cluster, and Borderline. The time consumption of DE is mostly governed by the number of parameters to be optimized (i.e., the searching space). The number of parameters to be optimized in LTRUS depends on the number of data metrics. In SDP, the number of data metrics is usually no more than 100, and the maximum number of data metrics adopted in this study is 61. Meanwhile, DE is often applied to solve the optimization problems with searching space consisting of hundreds or even thousands of dimensions (Guan, Zhao, Yin, & Li, 2021; Laloy & Vrugt, 2012), which implies that the runtime of LTRUS is not excessively long.

Furthermore, the process of exploring the optimal parameters for LTRUS with DE is performed offline. Once the process is complete, the final output is simply a linear model. Majority class instances are fed into the linear model and ranked based on the outputs of the linear model. The time consumption of this process is negligible. Therefore, LTRUS is practical in terms of time consumption.

7.3. Threats to validity

To ensure the generalizability of our experimental results, we utilized 21 datasets collected from various repositories. These datasets were widely adopted in previous studies and vary in terms of features, granularity, and programming languages. However, there are other datasets that differ from these datasets, which may lead to different conclusions. Additionally, we utilized four classifiers that have shown satisfactory performance in previous studies to generalize our results.

We used five performance measures, including one threshold-independent measure (AUC) and four threshold-dependent measures (Recall, Precision, F-measure, and MCC), to evaluate the performance. There are also other performance measures (e.g., G-mean and Balance). If those different datasets, classifiers, or performance measures are adopted, different conclusions may be drawn. Therefore, we detail our experiments to make it easy for others to replicate our work. Furthermore, biases and variances may be introduced when conducting the experiments. To minimize these, we adopt the 5-fold cross-validation with the stratification method to obtain the training and testing data and repeat the entire experiment ten times.

8. Conclusion and future work

According to previous studies, RUS, the most common and simplest undersampling technique, performs better than several oversampling techniques in SDP. However, the loss of important information is the major drawback of RUS. Besides, rigidly setting the final defect ratio equal to 0.5 as the termination condition of the undersampling technique goes against intuition. To solve these issues and improve the performance of the undersampling technique, we propose the Learning-To-Rank UnderSampling technique (LTRUS). LTRUS first learns an optimal linear model for the majority class instances by using DE. Then, the majority class instances are ranked in descending order based on the outputs of the linear model. Finally, the majority class instances are removed from the bottom of the rank to alleviate the class imbalance problem. The experiments conducted across 21 datasets on four common classifiers show that LTRUS performs significantly better than the baselines in terms of F-measure, AUC, and MCC. Furthermore,

the performances of LTRUS-ratio and LTRUS-thres show that (1) the conventional termination condition of the undersampling technique (fixed defect ratio 0.5) is improper, and (2) there is a trade-off lying in the undersampling technique.

For future work, although we propose LTRUS to mitigate the class imbalance problem in SDP, we do not introduce any prior knowledge of SDP into it. Therefore, we believe that LTRUS is applicable to other fields. We will adopt datasets collected from other fields to validate the generalizability of LTRUS. Besides, we plan to extend LTRUS to alleviate the multi-class imbalance problem. We also intend to explore a mechanism to handle the trade-off between the two different perspectives (the global view and the local view discussed in Section 7) to further improve the undersampling technique. Furthermore, whether the optimal parameters of LTRUS are universal among different classifiers is worthy our further investigation.

CRediT authorship contribution statement

Shuo Feng: Conceptualization, Methodology, Experiment, Writing. **Jacky Keung:** Reviewing. **Yan Xiao:** Writing – review & editing. **Peichang Zhang:** Review & editing. **Xiao Yu:** Review & editing. **Xiaochun Cao:** Review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have used the public data.

Acknowledgments

This work is supported in part by the Foundation of Shenzhen under Grant 20220810142731001 and 20200823154213001.

References

Abedin, M. Z., Guotai, C., Hajek, P., & Zhang, T. (2022). Combining weighted SMOTE with ensemble learning for the class-imbalanced prediction of small business credit risk. *Complex & Intelligent Systems*, 1–21.

Agrawal, A., & Menzies, T. (2018). Is" better data" better than" better data miners"? In *2018 IEEE/ACM 40th international conference on software engineering (ICSE)* (pp. 1050–1061). IEEE.

Bai, J., Jia, J., & Capretz, L. F. (2022). A three-stage transfer learning framework for multi-source cross-project software defect prediction. *Information and Software Technology*, *150*, Article 106985.

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, *22*(10), 751–761.

Bennin, K. E., Keung, J. W., & Monden, A. (2019). On the relative value of data resampling approaches for software defect prediction. *Empirical Software Engineering*, *24*(2), 602–636.

Bennin, K. E., Keung, J., Phannachitta, P., Monden, A., & Mensah, S. (2017). Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Transactions on Software Engineering*, *44*(6), 534–550.

Blanchard, G., & Loubere, R. (2016). High-Order Conservative Remapping with a posteriori MOOD stabilization on polygonal meshes.

Buckland, M., & Gey, F. (1994). The relationship between recall and precision. *Journal of the American Society for Information Science*, *45*(1), 12–19.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357.

Chen, H., Jing, X.-Y., Zhou, Y., Li, B., & Xu, B. (2022). Aligned metric representation based balanced multiset ensemble learning for heterogeneous defect prediction. *Information and Software Technology*, *147*, Article 106892.

Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, *21*, 1–13.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, *20*(6), 476–493.

D'Ambros, M., Lanza, M., & Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *2010 7th IEEE working conference on mining software repositories (MSR 2010)* (pp. 31–41). IEEE.

D'Ambros, M., Lanza, M., & Robbes, R. (2012). Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, *17*(4–5), 531–577.

Douzas, G., Bacao, F., & Last, F. (2018). Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE. *Information Sciences*, *465*, 1–20.

Feng, W., Huang, W., & Ren, J. (2018). Class imbalance ensemble learning based on the margin theory. *Applied Sciences*, *8*(5), 815.

Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K. E., Kabir, M. A., et al. (2020). COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction. *Information and Software Technology*, Article 106432.

Feng, S., Keung, J., Yu, X., Xiao, Y., & Zhang, M. (2021). Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. *Information and Software Technology*, Article 106662.

Feng, S., Keung, J., Zhang, P., Xiao, Y., & Zhang, M. (2022). The impact of the distance metric and measure on SMOTE-based techniques in software defect prediction. *Information and Software Technology*, *142*, Article 106742.

Gao, Y., Zhu, Y., & Zhao, Y. (2022). Dealing with imbalanced data for interpretable defect prediction. *Information and Software Technology*, *151*, Article 107016.

Gong, L., Jiang, S., Wang, R., & Jiang, L. (2019). Empirical evaluation of the impact of class overlap on software defect prediction. In *2019 34th IEEE/ACM international conference on automated software engineering* ASE, (pp. 698–709). IEEE.

Guan, B., Zhao, Y., Yin, Y., & Li, Y. (2021). A differential evolution based feature combination selection algorithm for high-dimensional data. *Information Sciences*, *547*, 870–886.

Gupta, N., Jindal, V., & Bedi, P. (2022). CSE-IDS: Using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems. *Computers & Security*, *112*, Article 102499.

Halstead, M. H. (1977). *Elements of software science (Operating and programming systems series)*. Elsevier Science Inc..

Han, H., Wang, W.-Y., & Mao, B.-H. (2005). Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing* (pp. 878–887). Springer.

Hassan, A. E. (2009). Predicting faults using the complexity of code changes. In *2009 IEEE 31st international conference on software engineering* (pp. 78–88). IEEE.

Hassanat, A. B., Tarawneh, A. S., Abed, S. S., Altarawneh, G. A., Alrashidi, M., & Alghamdi, M. (2022). Rdpvr: Random data partitioning with voting rule for machine learning from class-imbalanced datasets. *Electronics*, *11*(2), 228.

He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)* (pp. 1322–1328). IEEE.

Iranmehr, A., Masnadi-Shirazi, H., & Vasconcelos, N. (2019). Cost-sensitive support vector machines. *Neurocomputing*, *343*, 50–64.

Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, *6*(5), 429–449.

Jiang, F., Yu, X., Du, J., Gong, D., Zhang, Y., & Peng, Y. (2021). Ensemble learning based on approximate reducts and bootstrap sampling. *Information Sciences*, *547*, 797–813.

Jiang, F., Yu, X., Gong, D., & Du, J. (2022). A random approximate reduct-based ensemble learning approach and its application in software defect prediction. *Information Sciences*, *609*, 1147–1168.

Jin, C. (2021). Cross-project software defect prediction based on domain adaptation learning and optimization. *Expert Systems with Applications*, *171*, Article 114637.

Kamei, Y., Monden, A., Matsumoto, S., Kakimoto, T., & Matsumoto, K.-i. (2007). The effects of over and under sampling on fault-prone module detection. In *First international symposium on empirical software engineering and measurement (ESEM 2007)* (pp. 196–204). IEEE.

Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., et al. (2012). A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, *39*(6), 757–773.

Kampenes, V. B., Dybå, T., Hannay, J. E., & Sjøberg, D. I. (2007). A systematic review of effect size in software engineering experiments. *Information and Software Technology*, *49*(11–12), 1073–1086.

Karmaker Santu, S. K., Sondhi, P., & Zhai, C. (2017). On application of learning to rank for e-commerce search. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval* (pp. 475–484).

Khan, S. H., Hayat, M., Bennamoun, M., Sohel, F. A., & Togneri, R. (2017). Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(8), 3573–3587.

Kim, S., Zhang, H., Wu, R., & Gong, L. (2011). Dealing with noise in defect prediction. In *2011 33rd international conference on software engineering* ICSE, (pp. 481–490). IEEE.

Kim, S., Zimmermann, T., Whitehead, E. J., Jr., & Zeller, A. (2007). Predicting faults from cached history. In *29th international conference on software engineering (ICSE'07)* (pp. 489–498). IEEE.

Laloy, E., & Vrugt, J. A. (2012). High-dimensional posterior exploration of hydrologic models using multiple-try DREAM (ZS) and high-performance computing. *Water Resources Research*, *48*(1).

Li, F., Lu, W., Keung, J. W., Yu, X., Gong, L., & Li, J. (2023). The impact of feature selection techniques on effort-aware defect prediction: An empirical study. *IET Software*.

Lin, W.-C., Tsai, C.-F., Hu, Y.-H., & Jhang, J.-S. (2017). Clustering-based undersampling in class-imbalanced data. *Information Sciences*, *409*, 17–26.

Liu, S., Wang, Y., Zhang, J., Chen, C., & Xiang, Y. (2017). Addressing the class imbalance problem in twitter spam detection using ensemble learning. *Computers & Security*, *69*, 35–49.

Liu, X.-Y., Wu, J., & Zhou, Z.-H. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *39*(2), 539–550. http://dx.doi.org/10.1109/TSMCB.2008.2007853.

Liu, T.-Y., et al. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, *3*(3), 225–331.

Lu, J., Wu, D., Mao, M., Wang, W., & Zhang, G. (2015). Recommender system application developments: a survey. *Decision Support Systems*, *74*, 12–32.

Macbeth, G., Razumiejczyk, E., & Ledesma, R. D. (2011). Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, *10*(2), 545–555.

Maldonado, S., López, J., & Vairetti, C. (2019). An alternative SMOTE oversampling strategy for high-dimensional datasets. *Applied Soft Computing*, *76*, 380–389.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, (4), 308–320.

Menzies, T., Caglayan, B., Kocaguneli, E., Krall, J., Peters, F., & Turhan, B. (2012). The promise repository of empirical software engineering data.

Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on software engineering* (pp. 181–190).

Nam, J., & Kim, S. (2015). Clami: Defect prediction on unlabeled datasets (t). In *2015 30th IEEE/ACM international conference on automated software engineering* ASE, (pp. 452–463). IEEE.

Nguyen, T. T., An, T. Q., Hai, V. T., & Phuong, T. M. (2014). Similarity-based and rank-based defect prediction. In *2014 international conference on advanced technologies for communications (ATC 2014)* (pp. 321–325). IEEE.

Ohlsson, N., & Alberg, H. (1996). Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, *22*(12), 886–894.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Petrides, G., & Verbeke, W. (2022). Cost-sensitive ensemble learning: a unifying framework. *Data Mining and Knowledge Discovery*, *36*(1), 1–28.

Rey, D., & Neuhäuser, M. (2011). Wilcoxon-signed-rank test. In *International encyclopedia of statistical science* (pp. 1658–1659). Berlin, Heidelberg: Springer.

Riquelme, J., Ruiz, R., Rodríguez, D., & Moreno, J. (2008). Finding defective modules from highly unbalanced datasets. *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos*, *2*(1), 67–74.

Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, *39*(9), 1208–1215.

Song, Q., Guo, Y., & Shepperd, M. (2018). A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering*, *45*(12), 1253–1269.

Song, L., & Minku, L. L. (2022). A procedure to continuously evaluate predictive performance of just-in-time software defect prediction models during software development. *IEEE Transactions on Software Engineering*.

Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341–359.

Tahir, A., Bennin, K. E., Xiao, X., & MacDonell, S. G. (2021). Does class size matter? An in-depth assessment of the effect of class size in software defect prediction. *Empirical Software Engineering*, *26*(5), 1–38.

Tan, M., Tan, L., Dara, S., & Mayeux, C. (2015). Online defect prediction for imbalanced data. In *2015 IEEE/ACM 37th IEEE international conference on software engineering, Vol. 2* (pp. 99–108). IEEE.

Tantithamthavorn, C., Hassan, A. E., & Matsumoto, K. (2020). The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Transactions on Software Engineering*, *46*(11), 1200–1219.

Tarawneh, A. S., Hassanat, A. B., Altarawneh, G. A., & Almuhaimeed, A. (2022). Stop oversampling for class imbalance learning: A review. *IEEE Access*.

Tian, J. (2005). *Software quality engineering: Testing, quality assurance, and quantifiable improvement*. John Wiley & Sons.

Tong, H., Lu, W., Xing, W., Liu, B., & Wang, S. (2022). SHSE: A subspace hybrid sampling ensemble method for software defect number prediction. *Information and Software Technology*, *142*, Article 106747.

Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, *14*, 540–578.

Usta, A., Altingovde, I. S., Ozcan, R., & Ulusoy, Ö. (2021). Learning to rank for educational search engines. *IEEE Transactions on Learning Technologies*, *14*(2), 211–225.

Wang, Q., Wu, W., Qi, Y., & Zhao, Y. (2021). Deep Bayesian active learning for learning to rank: a case study in answer selection. *IEEE Transactions on Knowledge and Data Engineering*.

Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, *62*(2), 434–443.

Weyuker, E. J., Ostrand, T. J., & Bell, R. M. (2010). Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering*, *15*(3), 277–295.

Wong, G. Y., Leung, F. H., & Ling, S.-H. (2013). A novel evolutionary preprocessing method based on over-sampling and under-sampling for imbalanced datasets. In *Iecon 2013-39th annual conference of the ieee industrial electronics society* (pp. 2354–2359). IEEE.

Wu, Z., Lin, W., & Ji, Y. (2018). An integrated ensemble learning model for imbalanced fault diagnostics and prognostics. *IEEE Access*, *6*, 8394–8402.

Wu, R., Zhang, H., Kim, S., & Cheung, S.-C. (2011). 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).

Yang, X., Tang, K., & Yao, X. (2014). A learning-to-rank approach to software defect prediction. *IEEE Transactions on Reliability*, *64*(1), 234–246.

Yu, X., Dai, H., Li, L., Gu, X., Keung, J. W., Bennin, K. E., et al. (2023). Finding the best learning to rank algorithms for effort-aware defect prediction. *Information and Software Technology*, Article 107165.

Yu, X., Keung, J., Xiao, Y., Feng, S., Li, F., & Dai, H. (2022). Predicting the precise number of software defects: Are we there yet? *Information and Software Technology*, *146*, Article 106847.

Yu, X., Liu, J., Keung, J. W., Li, Q., Bennin, K. E., Xu, Z., et al. (2019). Improving ranking-oriented defect prediction using a cost-sensitive ranking SVM. *IEEE Transactions on Reliability*, *69*(1), 139–153.

Zhang, C., Soda, P., Bi, J., Fan, G., Almpanidis, G., Garcia, S., et al. (2022). An empirical study on the joint impact of feature selection and data resampling on imbalance classification. *Applied Intelligence*, 1–13.