DOI: 10.15514/ISPRAS-2024-36(6)-2

Class Balancing Approaches to Improve for Software Defect Prediction Estimations

Á. J. Sánchez-García, ORCID: 0000-0002-2917-2960 <angesanchez@uv.mx>
X. Limón, ORCID: 0000-0003-4654-636X <hlimon@uv.mx>
S. Domínguez-Isidro, ORCID: 0000-0002-9546-8233 <sauldominguez@uv.mx>
D. J. Olvera-Villeda, ORCID: 0009-0000-5615-1498 <olveradan@hotmail.com>
J. C. Pérez-Arriaga, ORCID: 0000-0003-2354-2462 <juaperez@uv.mx>

Faculty of Statistics and Informatics, Veracruz University,
Veracruz, Mexico.

Abstract. Addressing software defects is an ongoing challenge in software development, and effectively managing and resolving defects is vital for ensuring software reliability, which is in turn a crucial quality attribute of any software system. Software defect prediction supported by Machine Learning (ML) methods offers a promising approach to address the problem of software defects. However, one common challenge in ML-based software defect prediction is the issue of data imbalance. In this paper, we present an empirical study aimed at assessing the impact of various class balancing methods on the issue of class imbalance in software defect prediction. We conducted a set of experiments that involved nine distinct class balancing methods across seven different classifiers. We used datasets from the PROMISE repository, provided by the NASA software project. We also employed various metrics including AUC, Accuracy, Precision, Recall, and the F1 measure to gauge the effectiveness of the different class balancing methods. Furthermore, we applied hypothesis testing to determine any significant differences in metric results between datasets with balanced and unbalanced classes. Based on our findings, we conclude that balancing the classes in software defect prediction yields significant improvements in overall performance. Therefore, we strongly advocate for the inclusion of class balancing as a pre-processing step in this domain.

Keywords: software defect prediction; statistical analysis; imbalanced class; PROMISE; datasets; metrics; oversampling; undersampling.

For citation: Sánchez-García Á. J., Limón X., Domínguez-Isidro S., Olvera-Villeda D. J., Pérez-Arriaga J. C. Class balancing approaches to improve for software defect prediction estimations. Trudy ISP RAN/Proc. ISP RAS, vol. 36, issue 6, 2024. pp. 19-38. DOI: 10.15514/ISPRAS-2024-36(6)-2.

Подходы к балансировке классов для улучшения оценок прогнозирования дефектов программного обеспечения

А. Х. Санчес-Гарсия, ORCID: 0000-0002-2917-2960 <angesanchez@uv.mx>
К. Лимон, ORCID: 0000-0003-4654-636X <hlimon@uv.mx>
С. Домингес-Исидро, ORCID: 0000-0002-9546-8233 <sauldominguez@uv.mx>
Д. Х. Олвера-Вийеда, ORCID: 0009-0000-5615-1498 <olveradan@hotmail.com>
Х. К. Перес-Арриага, ORCID: 0000-0003-2354-2462 <juaperez@uv.mx>

Университет Веракруса, факультет статистики и информатики, Веракрус, Мексика.

Аннотация. Постоянной проблемой в разработке программного обеспечения является устранение дефектов в этом обеспечении. И эффективное управление и устранение дефектов имеют жизненно важное значение для обеспечения надежности программного обеспечения, что, в свою очередь, является важнейшим атрибутом качества любой системы программного обеспечения. Прогнозирование программных дефектов, поддерживаемое методами машинного обучения (ML) – это многообещающий подход к решению проблемы программных дефектов. Тем не менее, одной из общих проблем в прогнозировании дефектов программного обеспечения на основе ML является проблема дисбаланса данных. В этой статье мы представляем эмпирическое исследование, направленное на оценку влияния различных методов балансировки классов на проблему дисбаланса классов в прогнозировании дефектов программного обеспечения. Мы провели ряд экспериментов, которые включали девять различных методов балансировки классов по семи различным классификаторам. Мы использовали наборы данных из репозитория PROMISE, предоставленные программным проектом NASA. Мы также использовали различные метрики, включая AUC, точность, полнота, отзыв и меру F1, чтобы оценить эффективность методов балансировки различных классов. Кроме того, мы применили проверку гипотез, чтобы определить любые существенные различия в метрических результатах между наборами данных со сбалансированными и несбалансированными классами. Основываясь на наших выводах, мы пришли к выводу, что балансировка классов в прогнозировании дефектов программного обеспечения дает значительное улучшение общей производительности. Поэтому мы решительно выступаем за включение балансировки классов в качестве этапа предварительной обработки в этой области.

Ключевые слова: прогнозирование дефектов программного обеспечения; статистический анализ; несбалансированный класс; репозиторий PROMISE; наборы данных; метрики; избыточная выборка; недостаточная выборка.

Для цитирования: Санчес-Гарсия А. Х., Лимон К., Домингес-Исидро С., Олвера-Вийеда Д. Х., Перес-Арриага Х. К. Подходы к балансировке классов для улучшения оценок прогнозирования дефектов программного обеспечения. Труды ИСП РАН, том 36, вып. 6, 2024 г., стр. 19–38 (на английском языке). DOI: 10.15514/ISPRAS–2024–36(6)–2.

1. Introduction

This paper is an extension of work originally presented in 2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT) [1]. The original work addressed a systematic literature review on class balancing methods in software defect prediction data sets.

Software Engineering is a discipline that aims to develop and deliver quality software products, through the execution of quality processes. Quality can be defined as “the degree to which a component, system or process satisfies the specified requirements and/or the needs and expectations of the user/customer” [2].

One of the most important attributes to satisfy customer requirements is reliability, which is defined as “the probability of failure- free operation in a specific environment during a given period of time” [3]. System defects, which are imperfections or deficiencies in a system or component that could prevent it from carrying out its intended purpose, can have an impact on this reliability [4].

This is important because “quality is inversely proportional to the number of defects found in a system” [5], including an impact on software reliability.

An area of software engineering that is gaining importance due to the need to increase the reliability of a software system is the well-known software defect prediction (SDP). This area aims to predict the presence of defects in the software during or after its development. To make these predictions, it is necessary to have historical data that allows generating prediction models based on some metrics taken from software projects.

One of the most used historical data repositories for defect prediction is the PROMISE Software Engineering repository [6], which consists of several datasets for different purposes, such as effort or defect prediction.

The main data sets for SDP in PROMISE repository are provided by NASA software projects, which are:

1) CM1, which is a NASA spacecraft instrument written in the C programming language.
2) JM1 is written in the C programming language, and it is from a real-time predictive ground system.
3) KC1 written in C++ programming language, is a system implementing storage management for receiving and processing ground data.
4) KC2 data are from C++ functions as part of the same project of KC1.
5) PC1 data from C functions for a flight software for earth orbiting satellite.

These five datasets include data obtained from McCabe [7] and Halstead [8] static code metrics. These data sets contain a class label that states whether the record is defect-prone or not, making it a classification problem.

However, as Table 1 shows, the number of records of each class (yes or no) is unbalanced, which impacts the precision of prediction models, since models bias toward the most common value, causing prediction errors, which can impact software engineering practitioners.

Table 1. Distribution of classes by data set.

Data set	Instances	Class Yes	Class No
CM1	498	49 (9.83 %)	449 (90.16%)
JM1	10,885	8,779 (80.65%)	2,106 (19.35%)
KC1	2,109	326 (15.45%)	1,783 (84.54%)
KC2	522	105 (20.5%)	415 (79.5%)
PC1	1,1109	1,032 (94.05%)	77 (6.94%)

It is important to highlight that class unbalance is a prevalent challenge in software defect prediction. The five datasets discussed in this paper serve as illustrative examples to emphasize the prominence of this issue.

When a data set is unbalanced, machine learning (ML) algorithms will more frequently predict values from the majority class to increase the value of predictions with the training data. This also implies that learning models could not generalize new data that is not part of the training data. In conclusion, these datasets serve as valuable reference points for researchers aiming to enhance classifiers for SDP, as well as for software engineers who seek optimal classifier options for this task. Nevertheless, it is crucial to recognize that the accuracy of these classifiers may be affected by bias arising from the number of examples in the majority class.

As stated, and demonstrated in [9], there is no ML algorithm that has the best performance for all data sets. Therefore, given the class unbalance nature of SDP, we propose to carry out an empirical and statistical study, applying different algorithms and approaches for class balancing to the data sets of SDP, with the aim of establishing which approaches can be used by software engineers to obtain better predictions of software defects, and consequently, deliver reliable and quality products.

Due to what was mentioned above and what is described in section 2 (Related work), our main contributions in this work are:

- Unlike similar studies, we experimented with three class balancing approaches: oversampling, undersampling, and a combination of them.
- We tested the class balancing algorithms with eight classification algorithms from different approaches such as decision trees, K-nearest neighbor, Bayesian approaches, neural networks and ensemble classifiers (both boosting and bagging).
- We provide details for our experiments, such as the versions of the classification algorithms, type of implementation and parameters used, so that the results can be replicated.
- The data sets used are public, unlike studies where their own data sets are used.
- Our results not only report the approach that showed the best performance during the experiments, but the results are statistically validated to know if in the prediction results, there is a significant difference in performance when balancing the data sets.

This paper is structured as follows: Section 2 mentions related work. Section 3 presents the background of class balancing algorithms. Section 4 describes the characteristics of the experimentation and evaluation. Section 5 analyzes the results obtained. Section 6 identifies some validity threats of this study. Finally, section 7 draws conclusions and presents future work.

2. Related work

In our literature review reported in the last 6 years (until 2024), 40 studies were found that implemented proposals to balance classes in data sets of software defect prediction, of which 60% used Oversampling techniques, 18% Undersampling, 15% Oversamplig + Undersampling, and 8% Ensemble approaches. Table 2 shows the studies grouped by their balancing approach.

Table 2. Distribution of studies according to the balancing class approach.

Approach	Studies
Oversampling	[10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38]
Undersampling	[39], [40], [41], [11], [17], [18], [20], [34], [36], [37]
Combination (oversampling + undersampling)	[42], [43], [44], [45], [46], [47]
Ensemble	[48], [49], [50]

Of the 40 studies identified, only three of them [34-36] carry out an experiment like our proposal.

Pandey and Tripathi in [34] presented an empirical study in which the influence, in terms of F-score and Mathews Correlation Coefficient (MCC), of four sampling techniques (Class Balancer, Resample technique, Synthetic minority oversampling technique (SMOTE), and Spread subsample) were evaluated in twelve different classifiers: Naïve Bayes (NB), Logistic Regression (LR), Multilayer Perceptron (MLP), Instance-based learning (IBK), AdaBoost, Bagging (Bagg), Logistic Boost (LB), Repeated incremental pruning to produce error reduction (RIPPER), Decision Tree (J48), and Random Forest (RF). The authors considered 22 datasets in the Software Defect Prediction domain in their experiments. In their findings, the authors highlighted that MCC is the most helpful performance metric for the imbalance dataset; the Resample technique gave the best results in most ML techniques. Finally, the authors leave open the investigation regarding the relationship between the classifier and the sampling methods.

On the other hand, Zhang et al. in [35] measured the effect of using four Oversampling techniques for class balancing in just-in-time SDP: i) Random Oversampling, ii) SMOTE, iii) Borderline-SMOTE, and iv) ADASYN. The authors proposed the OSNECL method, which combines Oversampling techniques with ensemble classification methods, such as Bagging, AdaBoost, Random Forest, and GBDT. In their experiments, six opensource projects written in C++ and Java

(Bugzilla, Columba, Eclipse JDT, Mozilla, Eclipse Platform, and Postgres) were used, from which 16 characteristics related to the defects were extracted with the SZZ algorithm. As a result of the experiments, it was found that the combination of Random Forest and Random Oversampling provided the best results.

However, the authors left it open to continue researching class balancing with Deep Ensemble learning methods to explore new ways to improve the balancing problem.

Finally, Yang et al. [36] conducted a study that evaluated the impact of sampling, Random Under/Oversampling, SMOTE, and OSS techniques for the class balancing problem on Deep learning-based vulnerability detection (DLVD). The DLVD techniques used in their experiments were Deving, Reveal, IVDetect, and LineVul. Additionally, they selected three benchmark datasets, with which different findings could be obtained, such as Oversampling outperforms Undersampling approaches, sampling on raw data outperforms sampling on feature space; generally, Random Oversampling on raw data performing the best among all studied sampling methods, including SMOTE and OSS.

Table 3 compares the sampling techniques analyzed in the studies where we identify them as similar to our experimentation, where it can be seen that the comparison was carried out with a maximum of four class balancing algorithms, none of them is a combination of oversampling and undersampling. Our study proposes the comparison of nine class balancing algorithms, including a combination of oversampling and undersampling.

Table 3. Sampling techniques analyzed in related works.

Sampling Method	**[34]**	**[35]**	**[36]**
Random undersampling			✓
Random Oversampling		✓	✓
SMOTE	✓	✓	✓
OSS			✓
Class balancer	✓		
Resample	✓		
Spread subsample	✓		
Borderline – SMOTE		✓	
ADASYN		✓	

Table 4 presents the learning techniques with which the techniques were combined. In the case of the study [36], they used specific deep- learning techniques to detect vulnerabilities, which is not directly related to SDP. We also include different classification approaches and algorithms.

Table 4. Learning methods studied in related works.

Learning classifier	**[34]**	**[35]**	**[36]**
Naïve Bayes (NB)	✓		
Multiplayer perceptron (MLP)	✓	✓	
K-Nearest Neighbor (KNN)	✓		
Decision Tree (DT)	✓		
AdaBoost (AB)	✓	✓	
Bagging	✓	✓	
Logistic Boost (LB)	✓		
RIPPER	✓		
Random Forest (RF)	✓	✓	
Gradient Boosting (GB)		✓	
Deep learning approaches			✓

3. Class balancing methods

We examined nine representative methods to address class imbalance in our experiments. These methods can be classified into three categories as it is shown in Fig. 1: oversampling, undersampling, and combined. Below, we provide an explanation of each method.

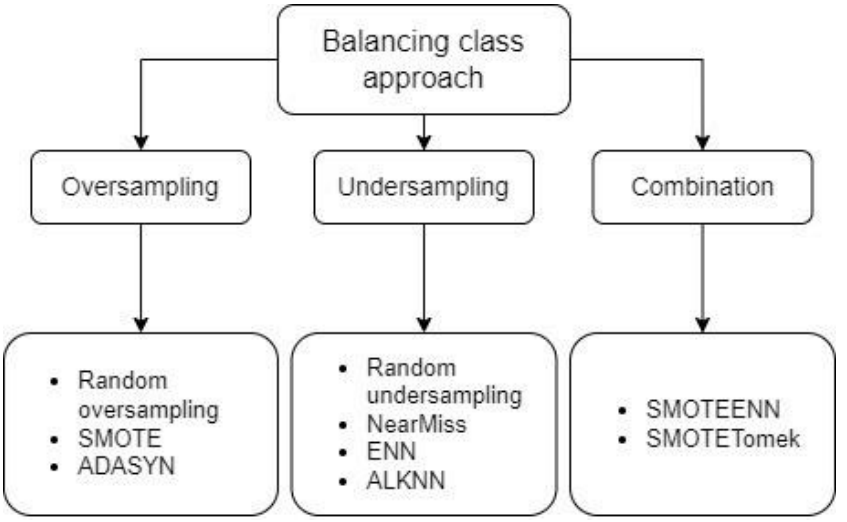

Fig. 1. Balancing class approaches chosen for experimentation.

3.1 Oversampling

Fig. 2 shows the strategy of class balancing by oversampling, where it consists of artificially increasing the number of samples of the minority class to balance the class distribution.

Three algorithms based on this approach are presented below.

1) Random over sampling: Also known as Random Over Sampling Examples (ROSE) [51], this simple method involves randomly selecting examples from the minority class or classes and repeating them with replacement. This process creates a new dataset that is balanced by increasing the representation of the minority class.
2) SMOTE [52], an acronym for Synthetic Minority Oversampling Technique, is a widely used oversampling method designed to address class imbalance. It involves the creation of synthetic data points in the feature space for the minority class. To over sample the minority class, SMOTE takes each minority class sample and generates synthetic examples along the line segments that connect the sample to its k-nearest neighbors from the same class. The value of k determines the number of nearest neighbors considered. The amount of oversampling required can be adjusted, and random selection is used to choose neighbors from the k-nearest neighbors.
3) ADASYN [53], short for Adaptive Synthetic Sampling Approach for Imbalanced Learning, is an oversampling technique specifically designed to address class imbalance. It involves the creation of synthetic data points for the minority class. The fundamental concept behind ADASYN is to use a weighted distribution to generate synthetic data based on the level of difficulty of learning for each minority class example. The algorithm identifies minority class examples that are harder to learn and generates a higher number of synthetic data points for them compared to examples that are easier to learn. ADASYN employs a density distribution based on the k-nearest neighbors of each minority class example to determine the number of synthetic samples to be generated.

3.2 Undersampling

Undersampling is another technique used to address class imbalance in data sets, where unlike oversampling, in this approach the number of samples of the majority class is reduced to equalize the class distribution with the minority class as can be seen in Fig. 3.

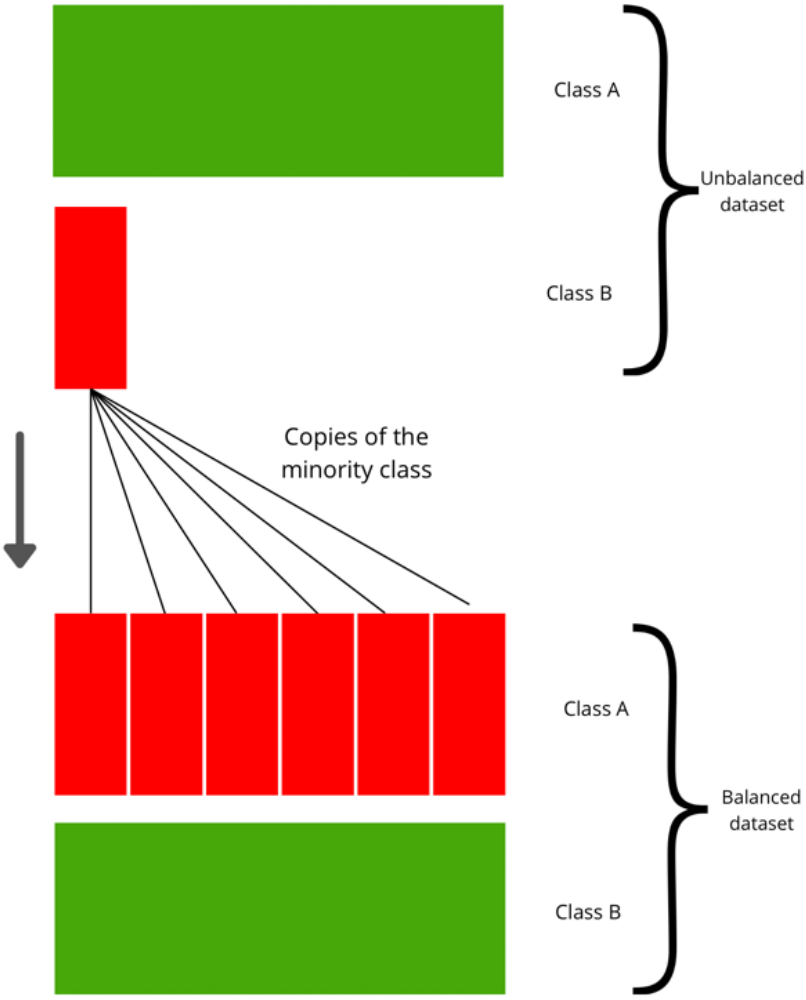

Fig. 2. Oversampling strategy.

Below, four selected undersampling methods are described.

1) Random undersampling is a technique used to reduce the number of majority class examples in a dataset. It is similar to random oversampling, but with the opposite objective. Random undersampling (RUS) [54] randomly selects examples from the majority class, either with or without replacement. The number of examples selected in random undersampling depends on the desired balancing goal. In a fully balanced dataset, the number of selected examples would equal the number of examples from the minority class.
2) NearMiss is an undersampling technique originally introduced in [55]. This method aims to balance class distribution by selecting a subset of majority class examples that are closest to the minority class examples based on their k-nearest neighbors. The NearMiss method

consists of three variants: NearMiss-1, NearMiss-2, and NearMiss-3.

- NearMiss-1: this variant focuses on selecting majority class examples that are closest to some of the minority class examples. It identifies majority class examples with the smallest average distance to their k-nearest minority class examples. The objective is to remove majority class examples that are farthest from the minority class.

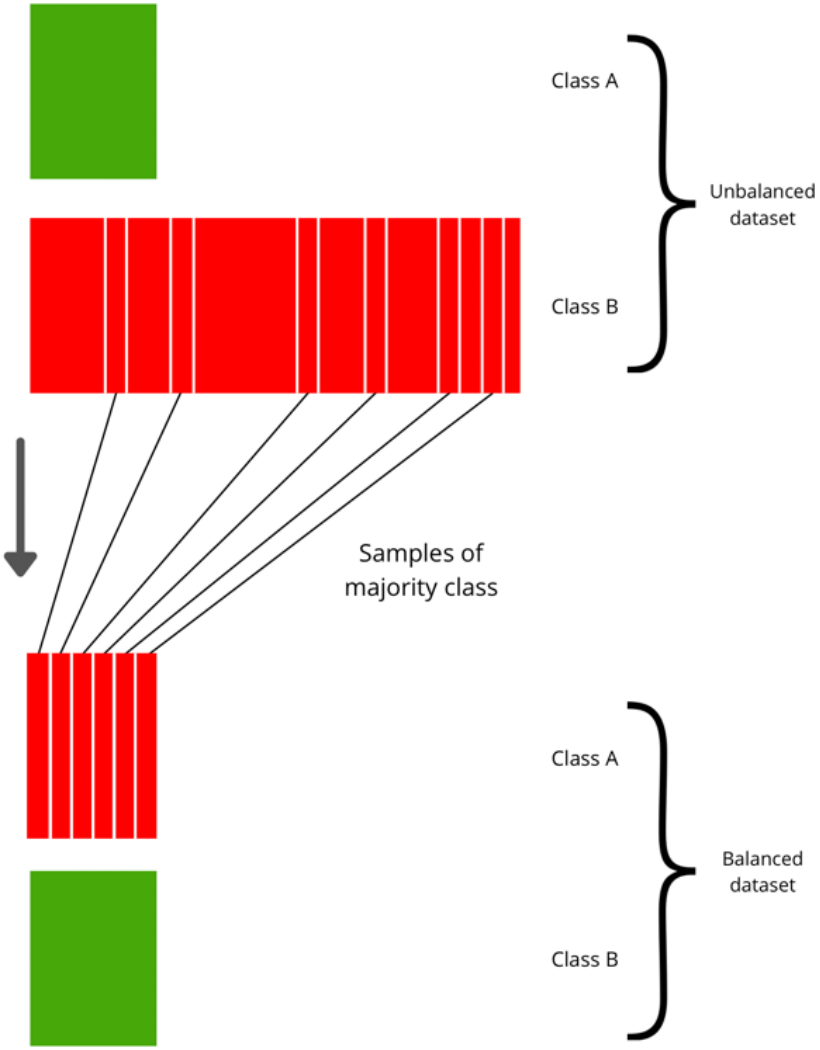

Fig. 3. Undersampling strategy.

- NearMiss-2: this variant selects majority class examples that are closest to all minority class examples. It chooses examples based on their average distances to the k farthest minority class examples.
- NearMiss-3: selects a specific number of closest majority class examples for each minority class example. The intention is to ensure that each minority class example is surrounded by some majority class examples.

3) ENN (Edited Nearest Neighbors), proposed by [56], is an undersampling method for

addressing class imbalance. The ENN algorithm operates by examining the k-nearest neighbors of each example in the dataset. If a majority class example is misclassified, it is removed from the dataset. On the other hand, if a minority class example is misclassified, its k neighbors belonging to the majority class are removed. The k value proposed in the original work is 3. It is important to note that ENN is primarily designed to remove noisy and ambiguous examples, rather than achieving perfect class balance.

4) AllKNN (All K-Nearest Neighbors), proposed in a study by [57], is an iterative undersampling method that extends the concept of ENN. Rather than using a fixed value of k, AllKNN repeatedly applies ENN with increasing values of k starting from 1. This iterative process continues until a maximum value of k is reached or until the minority class becomes the majority class. The use of progressively larger k values allows AllKNN to capture more information from the nearest neighbors and potentially achieve better imbalance reduction.

3.3 Combination

Finally, by combining oversampling and undersampling techniques, the strengths of each approach can be leveraged for better classification results. Two algorithms that combine the two previous approaches are described below.

1) SMOTEEN (SMOTE Edited Nearest Neighbors), proposed by [58], is a method that leverages both oversampling and undersampling techniques to address class imbalance. The SMOTEEN algorithm combines the synthetic minority oversampling technique (SMOTE) with the edited nearest neighbors (ENN) undersampling. It follows a two-step process to rebalance the dataset. First, SMOTE is applied to the minority class to generate synthetic instances, thereby increasing its representation. Next, ENN is employed to remove potentially noisy or mislabeled instances from both the minority and majority classes. The SMOTEEN process may be repeated until achieving a balanced dataset or reaching a stopping criterion.

2) SMOTETomek [59] is a combination of oversampling technique SMOTE, and undersampling technique Tomek Links [60]. SMOTETomek consists of two main steps. In the first step, SMOTE is applied to the minority class, oversampling with synthetic data and balancing the dataset. The second step of SMOTETomek utilizes Tomek Links. Tomek Links identify specific pairs of instances, one from the majority class and one from the minority class, that are nearest neighbors to each other. These pairs are ambiguous or potentially noisy instances, and thus are removed from the date set, undersampling at the same time examples from both classes.

4. Experimental design

Fig. 4 shows the steps followed for the experiments in this study. For our experiments, the only preprocessing performed was checking for missing values. In all five data sets, only JM1 included 5 records with at least one missing value. These records were removed.

After data preprocessing, the class balancing algorithms mentioned in Section 3 will be applied to each of the five data sets.

Subsequently, each classification learning algorithm described in section 4.1 will be tested on each data set (with and without class balancing).

The models are evaluated through a k-fold cross validation process in which the data set is divided into k sets (one subset is used for testing and the other k-1 subsets are used as training). This process is executed k times, such that each record is part of both a test and training set.

The metrics described in section 4.2 are applied to the models to evaluate the predictions. Finally, a statistical analysis is performed for each metric applied to each model developed by each classification algorithm and to each data set obtained with and without class balancing.

Finally, the results are statistically analyzed to empirically verify whether the results with the class balancing methods studied are significantly better.

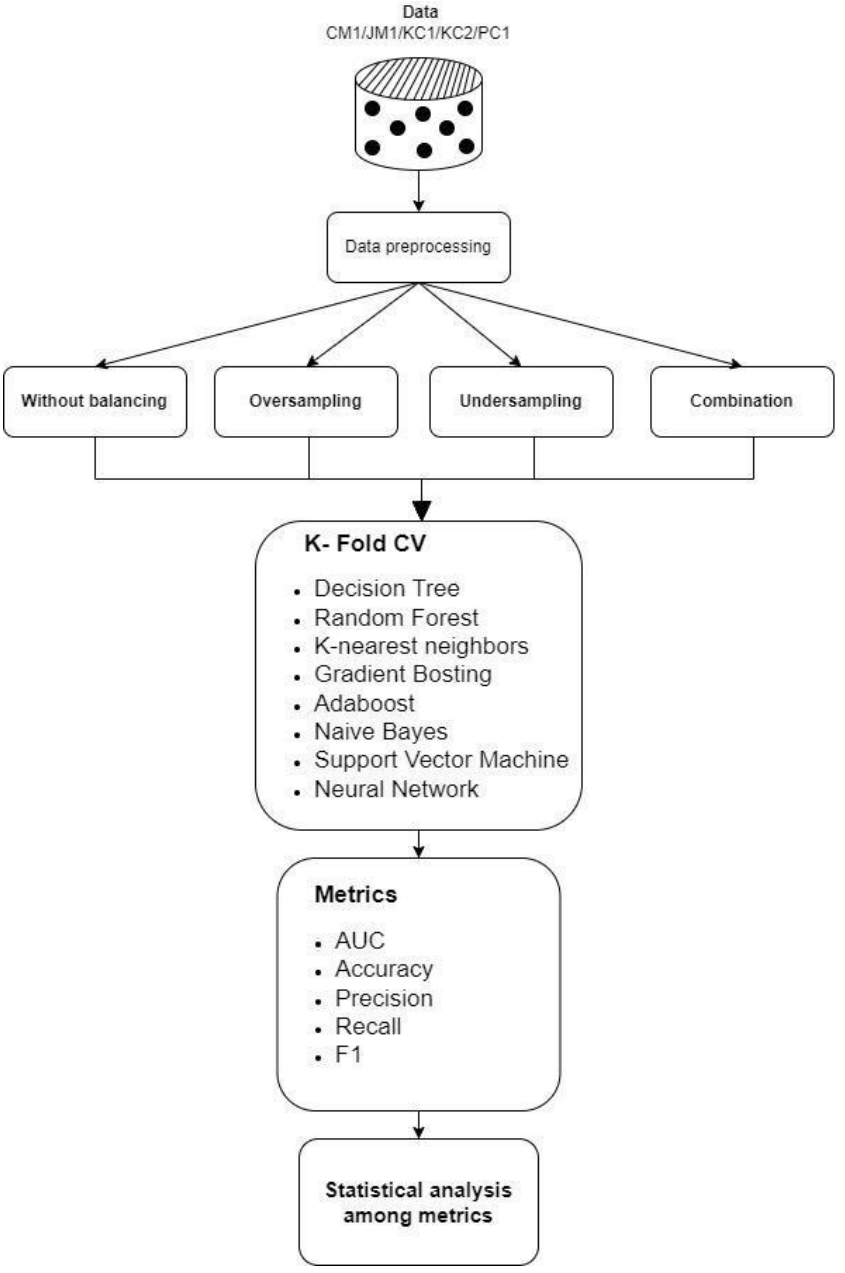

Fig. 4. Experimental design steps to validate models with balanced data sets.

4.1 Classification learning algorithms

As demonstrated in [9], there is no ML algorithm that performs better than the others for any data set. Therefore, we selected a variety of classification learning algorithms for experimentation, and they are described below, including their characteristics, advantages and disadvantages that could influence in the prediction results.

We chose several classification approaches, such as based on decision trees, class similarity (k-nearest neighbor), ensemble algorithms for both bagging and boosting, based on Bayesian approaches and finally on neural networks approach.

Decision Trees are a popular supervised learning method used for both classification and regression tasks [61]. They build a tree structure consisting of rules based on the input features, which guide the evaluation of new examples. This hierarchical representation enables visualizing the resulting model. Hence, Decision Trees employ a white model approach. Decision Trees have several advantages. Firstly, they require minimal data preparation, and they can handle both numeric and nominal data. Secondly, they are nonparametric, meaning they make no assumptions about the underlying data distribution. However, Decision Trees are prone to overfitting, to address this, pruning methods can be employed to simplify the tree and prevent overfitting. It is also important to note that Decision Trees can be affected by class imbalance [62].

K-Nearest Neighbors (KNN) [63] is a nonparametric and supervised method for classifying data instances. It utilizes a similarity metric, such as Euclidean distance, to identify the most similar examples and establish the concept of neighbors. By setting a parameter K, the algorithm determines the K-nearest neighbors. When classifying a new instance, a policy needs to be defined. A popular policy is majority vote, where the predicted class is the most common class among the K-nearest neighbors. It is important to note that the classifier's performance is sensitive to the choice of K. The KNN classifier may encounter limitations when dealing with large datasets. Additionally, it is important to consider feature scaling to ensure accurate results.

Random Forests [64] is an ensemble technique used for classification tasks. As an ensemble technique, it combines multiple classification models, specifically trees. Each tree is built from different random subsets of the dataset, allowing for improved generalization and robustness. One of the key features of Random Forests is the option to use a random sample of features when selecting the best splitting point in each tree. This further enhances the diversity and reduces correlation among the individual trees. During classification of a new example, each tree in the forest independently classifies the instance. Then, a decision policy, such as majority vote or averaging the predicted probabilities from each tree, is applied to determine the final class for the new instance. Random Forests are well-suited for handling large datasets with a high number of examples and features. They are also robust against data noise and overfitting.

AdaBoost [65] (Adaptive Boosting) is an iterative ensemble classification method that combines multiple weak classifiers to create a robust classifier. It utilizes a weighting strategy, where initially every example in the training set has equal weight. As the iterations progress, the weights of the examples are adjusted based on the classification performance of the classifiers. Additionally, AdaBoost assigns weights to each classifier based on their performance, with higher weights given to classifiers with better accuracy. During classification of a new instance, AdaBoost applies a weighted majority vote among the classifiers to determine the final class label.

Gradient Boosting [66] is an ensemble method used for both classification and regression tasks. It combines multiple weak models to create a stronger model. Gradient Boosting relies on Gradient optimization to iteratively build the ensemble by fitting weak models to the residuals. This process aims to correct errors in the current ensemble for Gradient optimization, residuals representing the gradients of the loss function are calculated with respect to the predicted probabilities. The subsequent weak model is then trained on these residuals to further improve the ensemble's performance. Additionally, each model in the ensemble is assigned a weight to assess its contribution. These weights consider the model's performance and are used to guide the ensemble

towards more accurate predictions. During the classification of a new example, a weighted vote policy is applied.

Naive Bayes: Naive Bayes is a supervised and probabilistic method used for classification tasks based on Bayes theorem [67]. It is a type of Bayesian Network method that operates under the assumption of feature independence. This strong hypothesis assumes that each feature in the data set is independent of each other given the class label. This assumption simplifies the model and allows for fast training as only the conditional probabilities need to be computed, while the Bayesian structure is predetermined.

To classify a new instance, Naive Bayes computes the conditional probabilities based on the Bayesian network parameters. It selects the class that maximizes the probability as the predicted class label. Naive Bayes is well-known for its simplicity, yet it has the potential to achieve good predictive performance in various problem domains.

A Multi-Layer Perceptron (MLP) [68] is an artificial neural network commonly used for classification and regression tasks. It enhances the single neuron perceptron by incorporating multiple layers of interconnected artificial neurons. The MLP model consists of an input layer for receiving data features, one or more hidden layers responsible for transforming input signals and adjusting connection weights between neurons, and an output layer that generates the final predictions. To optimize the MLP model, the log-loss function is typically used along with various optimization algorithms such as LBFGS (Limited-memory Broyden- Fletcher-Goldfarb-Shanno) or stochastic Gradient descent. These optimization algorithms aim to search for the optimal set of weights that minimize the loss function, allowing the model to make accurate predictions.

4.2 Evaluation models metrics

There are different metrics to evaluate the accuracy of a model. This is because each metric provides a different perspective on the performance of each algorithm and in some cases avoids biases due to majority classes. The metrics used to evaluate the algorithms used for this study are described below.

AUC is obtained directly from the ROC (Receiver Operating Characteristic) curve, which is a graphical representation of the true positive rate (sensitivity) versus the false positive rate (1 - specificity) for different classification threshold values.

The following metrics are estimated from the confusion matrix, where the true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) are placed:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = \frac{2 * TP}{2 * TP + FN + FP} \tag{4}$$

Accuracy is the proportion of modules correctly classified as defective and non- defective. It is calculated by (1).

Precision is the proportion of TP cases. In the SDP, it is the number of defective modules that were found correctly, and it is calculated by (2).

Recall measures the proportion of TP that there should be and is calculated using (3).

Finally, F1-measure is used when there is an unbalanced data set and combines the benefits of precision and recall as shown in (4).

4.3 Classifiers parameters

The implementations of algorithms were carried out in the Python language with the skit- learn 1.4.2 library. Table 5 shows the parameters of the classifiers used for out experiments. In the algorithms that shared some parameter, such as balancing the number of estimators, it was the same for all. The Random state parameter indicates the seed that was the same for all classification algorithms.

5. Results analysis

The average results obtained by the k-fold cross-validation process with k = 10 by each metric were analyzed, both for the data sets without applying any class balancing method and the application of each method described in section 3.

Table 5. Parameters used for each classifier.

Classifier algorithm	**Parameters**
Decision tree	Criterion: Gini Max Depth: None Max leaf nodes: None Min impurity decrease: 0.0 Class weight: None Random state: 42
K-NN	N neighbors: 3 Metric: Euclidean distance
Random Forest	N estimators: 50 Criterion: Gini Max Depth: None Max leaf nodes: None Min impurity decrease: 0.0 Class weight: None Random state: 42
Ada boost	N estimators: 50 Algorithm: SAMME.R Learning rate: 1.0 Random state: 42
Gradient Boosting	N estimators: 50 Loss: log loss Learning rate: 1.0 Criterion: Friedman mse Max depth: 3 Min impurity decrease: 0.0
Naive Bayes	Distribution: Gaussian
Multilayer perceptron	Hidden layer: 5 Random state: 42 Alpha: 0.0005

5.1 Balancing class approaches results

The results show that applying a class balancing method increases the prediction accuracy of different classifiers with different subsets of data through cross-validation, contributes to the increase of reliability in predictions.

With respect to the CM1 data set, performing a random undersampling class balancing procedure worsens the results, while with ALKNN practically the same results are obtained.

For the JM1 data set, all the balancing methods outperform the results without applying any class balancing method except with random undersampling, which obtains very similar results.

Regarding the KC1 data set, most class balancing algorithms outperform the average predictions of using an unbalanced data set. However, using undersampling in the assembled algorithms (AdaBoost and Gradient boosting) results in lower precision values.

All class balancing algorithms outperform the prediction values using the raw KC2 data set.

Finally, the results of the PC1 data set show that the ENN and ALLKNN undersampling algorithms have the worst performance, almost equal to not using a class balancing method.

5.2 Classifiers results

The results show that the classifiers that had the best results in the selected model evaluation metrics are the ensemble classifiers, that is, random forest, AdaBoost and Gradient boosting. The classifier that showed the worst results was the multilayer perceptron. With these results, software engineering practitioners can opt for ensemble classifiers, as it is shown that it is advisable to invest in the processing of several weak classifiers to obtain better prediction accuracy.

5.3 Metrics results

The results of the AUC, accuracy, precision, recall and F1 metrics are similar when the values of the classes of the data sets are balanced. It is quantitatively verified that when the data sets are not balanced, the recall, F1 and precision metrics tend to be lower than accuracy. This is because classifiers tend to predict the majority class, and by performing an evaluation on all the predictions, there is a greater probability that the majority class matches the predicted value. For this reason, it is better to have an overview of different metrics that evaluate accuracy rates differently.

5.4 Statistical Analysis

The metric values are not sufficient to establish a statistically significant improvement in the predictions. Therefore, a suitable statistical test was applied to compare the accuracies among the class balancing models [69].

First, χ^2, Shapiro-Wilk, skewness and kurtosis normality tests for each set of predictions (ten for each metric that were obtained from the cross-validation process) to check if the normality assumption was met were applied.

For each metric and for each data set (set of all classifiers) it was checked if at least some test was not significant at 95%, then a normal distribution was not considered. If all values of each metric had a normal distribution, an ANOVA test was performed, if at least one set did not meet the normality assumption, then Friedman non-parametric test was applied to determine whether at least one algorithm generated significantly different results from the others. The results of the statistical tests can be observed in [70], where it is concluded that in all groups by data set and by class balancing algorithm, there was at least one value statistically different from the rest of them.

Finally, to know if the statistically significant differences were the results related to the data sets without applying any class balancing algorithm, the values of the metrics of each balancing algorithm were compared with the values of not applying a class balancing. To do this, the four normality statistical tests were performed for each set of prediction metrics. If any of their four p-values were lower than 0.05, then data were non-normally distributed at 95% confidence, and a Wilcoxon test was applied (the medians of the models should be compared); otherwise, a t-paired test was performed (the means of models were compared) [71]. The results of these tests can be consulted in [72].

In [72] is shown that the class balancing algorithms for each metric, where the results of the predictions were statistically different from the predictions of the same data set without using any class balancing algorithm at 95% confidence.

These results show that despite different class balancing algorithms and approaches, Naive Bayes and MLP do not generate significantly better predictions. However, using the other classifiers with any metric generally makes a significant difference in the predictions.

6. Validity threats

There are four categories of validity threats in search-based predictive modeling for Software Engineering: conclusion, internal, external and construct [73] of which, we identify the following:

Construct: We remove five records that have null values for the JM1 data set.

Construct: We are not involved in the acquisition of data to avoid bias due to human error.

Internal: Data is based on static code metrics only.

Internal: We did not include any processing to deal with outliers, as we wanted to evaluate them with real data from the PROMISE repository systems.

7. Conclusions and future work

The present study experiments with different approaches to mitigate the class balancing problem in SDP. These approaches are obtained from a literature review, of which three oversampling algorithms, four undersampling algorithms and two from the combination of the mentioned approaches were selected. The data sets used were those from the well-known public repository PROMISE, to obtain predictions with the data used in software industry.

To not bias the results with a particular classifier, seven classification algorithms (based on different approaches) were tested. Also, different metrics such as AUC, precision, accuracy, recall and F1 were evaluated to measure the performance of the algorithms with and without class balancing through a cross- validation process. Finally, the results are statistically evaluated to conclude if there really is a significant improvement when balancing the data sets.

With respect to the class balancing methods, we conclude that:

- Balancing the classes significantly improves prediction performance since learning is not biased by the majority class.
- The oversampling and combination methods have better performance than the algorithms based on undersampling.

With respect to the classification algorithms, we conclude that:

- The algorithms that obtained the best results were the assembled classifiers (both boosting and bagging), which base the predicted class value on the results of several weak classifiers. Therefore, if software engineering practitioners require greater precision, it is important to invest resources in these types of methods.
- The multilayer perceptron and Naïve Bayes algorithms are the classifier that yielded the worst results.

In terms of evaluation metrics, we conclude that:

- When the data sets are unbalanced, the metrics show very variable results.
- When the data sets are balanced, the results of the metrics are more homogeneous.

Finally, it is concluded that to improve the reliability of a software product, it is necessary to have good precision in predictions. Therefore, when the historical data to make predictions of software defects is unbalanced, it is suggested to first apply an oversampling mechanism and then use an ensemble classifier that strengthens the predictions, regardless of the set used for training and testing.

As future work we propose the following:

Repeat the experimentation with other classifiers including deep learning, since we only chose some representative ones of each approach.

As could be seen in Table 5, the classification algorithms need to adjust parameters, so it is proposed to implement some optimization algorithm that allows knowing the best parameters for each classification algorithm.

It is also proposed to carry out this analysis by programming language [74], since the development language could influence the introduction of defects and the quality of the software [75].

Finally, continue with experimentation on other data sets, which allows exploring the performance of class balancing algorithms in more dept.

References

[1]. D. J. Olvera-Villeda, A. J. Sánchez-García, X. Limón, and S. Domínguez Isidro, "Class balancing approaches in dataset for software defect prediction: A systematic literature review" in 2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT). IEEE, 2023, pp. 1–6.

[2]. M. Glinz, "A glossary of requirements engineering terminology", Standard Glossary of the Certified Professionalfor Requirements Engineering (CPRE) Studies and Exam, Version, vol. 1, p. 56, 2011.

[3]. J. D. Musa, "Software reliability measurement", Journal of Systems and Software, vol. 1, pp. 223–241, 1979.

[4]. I. Iso and N. IEC, "Iso/iec", IEEE International Standard-Systems and software engineering- Vocabulary, pp. 1–541,2017.

[5]. P. D. Singh and A. Chug, "Software defect prediction analysis using machine learning algorithms" in 2017 7th international conference on cloud computing, data science & engineering confluence. IEEE, 2017, pp. 775–781.

[6]. J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases", School of Information Technology and Engineering, University of Ottawa, Canada, 2005. [Online]. Available: http://PROMISEsite.uottawa.ca/SERepository

[7]. T. McCabe, "A complexity measure", IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308–320, December 1976.

[8]. M. Halstead, Elements of Software Science. Elsevier, 1977.

[9]. D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization", IEEE transactions on evolutionary computation, vol. 1, no. 1, pp. 67–82, 1997.

[10]. Y. Zhang, X. Yan, and A. A. Khan, "A kernel density estimation-based variation sampling for class imbalance in defect prediction" in 2020 IEEE Intl Conf. on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). IEEE, 2020, pp. 1058–1065.

[11]. E. Elahi, S. Kanwal, and A. N. Asif, "A new ensemble approach for software fault prediction" in 2020 17th international Bhurban conference on applied sciences and technology (IBCAST). IEEE, 2020, pp. 407–412.

[12]. J. Zheng, X. Wang, D. Wei, B. Chen, and Y. Shao, "A novel imbalanced ensemble learning in software defect predication", IEEE Access, vol. 9, pp. 86 855–86 868, 2021.

[13]. Q. Zha, X. Yan, and Y. Zhou, "Adaptive centre-weighted oversampling for class imbalance in software defect prediction", in 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018, pp. 223–230.

[14]. S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, "An ensemble oversampling model for class imbalance problem in software defect prediction", IEEE access, vol. 6, pp. 24 184–24 195, 2018.

[15]. R. Malhotra, N. Nishant, S. Gurha, and V. Rathi, "Application of particle swarm optimization for software defect prediction using object oriented metrics" in 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence). IEEE, 2021, pp. 88–93.

[16]. Z. Li, X. Zhang, J. Guo, and Y. Shang, "Class imbalance data generation for software defect prediction", in 2019 26th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2019, pp. 276–283.

[17]. S. Ghosh, A. Rana, and V. Kansal, "Combining integrated sampling with nonlinear manifold detection techniques for software defect prediction" in 2018 3rd International Conference on Contemporary Computing and Informatics (IC3I). IEEE, 2018, pp. 147–154.

[18]. S. A. Putri et al., “Combining integreted sampling technique with feature selection for software defect prediction” in 2017 5th International Conference on Cyber and IT Service Management (CITSM). IEEE, 2017, pp. 1–6.

[19]. T. Thaher and N. Arman, “Efficient Multi-Swarm Binary Harris Hawks Optimization as a Feature Selection Approach for Software Fault Prediction” in 2020 11th International conference on information and communication systems (ICICS). IEEE, 2020, pp. 249–254.

[20]. K. Bashir, T. Li, C. W. Yohannese, and Y. Mahama, “Enhancing software defect prediction using supervised-learning based framework” in 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE). IEEE, 2017, pp. 1–6.

[21]. S. S. Rathore, S. S. Chouhan, D. K. Jain, and A. G. Vachhani, “Generative oversampling methods for handling imbalanced data in software fault prediction”, IEEE Transactions on Reliability, vol. 71, no. 2, pp. 747–762, 2022.

[22]. Z. Eivazpour and M. R. Keyvanpour, “Improving performance in software defect prediction using variational autoencoder” in 2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI). IEEE, 2019, pp. 644–649.

[23]. A. Bispo, R. Prudˆencio, and D. VÅLeras, “Instance selection and class balancing techniques for cross project defect prediction” in 2018 7th Brazilian Conference on Intelligent Systems (BRACIS). IEEE, 2018, pp. 552–557.

[24]. K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, “Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction” IEEE Transactions on Software Engineering, vol. 44, no. 6, pp. 534–550, 2017.

[25]. R. Malhotra, R. Kapoor, P. Saxena, and P. Sharma, “Saga: A hybrid technique to handle imbalance data in software defect prediction” in 2021 IEEE 11th IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE). IEEE, 2021, pp. 331–336.

[26]. D. Wang and X. Xiong, “Software defect prediction based on combined sampling and feature selection” in ICMLCA 2021; 2nd International Conference on Machine Learning and Computer Application. VDE, 2021, pp. 1–5.

[27]. Y. Liu, F. Sun, J. Yang, and D. Zhou, “Software defect prediction model based on improved bp neural network” in 2019 6th International Conference on Dependable Systems and Their Applications (DSA). IEEE, 2020, pp. 521–522.

[28]. R. B. Bahaweres, F. Agustian, I. Hermadi, A. I. Suroso, and Y. Arkeman, “Software defect prediction using neural network based smote” in 2020 7th International Conference on Electrical Engineering, Computer Sciences and Informatics (EECSI). IEEE, 2020, pp. 71–76.

[29]. S. Choirunnisa, B. Meidyani, and S. Rochimah, “Software defect prediction using oversampling algorithm: A-suwo” in 2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS). IEEE, 2018, pp. 337– 341.

[30]. W. A. Dipa and W. D. Sunindyo, “Software defect prediction using smote and artificial neural network” in 2021 International Conference on Data and Software Engineering (ICoDSE). IEEE, 2021, pp. 1– 4.

[31]. R. Malhotra, V. Agrawal, V. Pal, and T. Agarwal, “Support vector based oversampling technique for handling class imbalance in software defect prediction” in 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence). IEEE, 2021, pp. 1078–1083.

[32]. L. Gong, S. Jiang, and L. Jiang, “Tackling class imbalance problem in software defect prediction through clusterbased over-sampling with filtering” IEEE Access, vol. 7, pp. 145 725–145 737, 2019.

[33]. R. Malhotra and S. Kamal, “Tool to handle imbalancing problem in software defect prediction using oversampling methods” in 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 2017, pp. 906–912.

[34]. S. K. Pandey and A. K. Tripathi, “Class imbalance issue in software defect prediction models by various machine learning techniques: an empirical study” in 2021 8th International Conference on Smart Computing and Communications (ICSCC). IEEE, 2021, pp. 58–63.

[35]. W. Zhang, Y. Li, M. Wen, and R. He, “Comparative study of ensemble learning methods in just-in-time software defect prediction” in 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRSC), 2023, pp. 83–92.

[36]. X. Yang, S. Wang, Y. Li, and S. Wang, “Does data sampling improve deep learning-based vulnerability detection? yeas! and nays!” in 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), 2023, pp. 2287–2298.

[37]. R. Kumar and A. Chaturvedi, “Software bug prediction using reward-based weighted majority voting ensemble technique”, IEEE Transactions on Reliability, vol. 73, no. 1, pp. 726–740, 2024.

[38]. M. Devi, T. Rajkumar, and D. Balakrishnan, "Prediction of software defects by employing optimized deep learning and oversampling approaches" in 2024 2nd International Conference on Computer, Communication and Control (IC4), 2024, pp. 1–5.

[39]. W. Wei, F. Jiang, X. Yu, and J. Du, "An under- sampling algorithm based on weighted complexity and its application in software defect prediction" in Proceedings of the 2022 5th International Conference on Software Engineering and Information Management, 2022, pp. 38–44.

[40]. G. Abaei, W. Z. Tah, J. Z. W. Toh, and E. S. J. Hor, "Improving software fault prediction in imbalanced datasets using the under-sampling approach" in Proceedings of the 2022 11th International Conference on Software and Computer Applications, 2022, pp. 41–47.

[41]. Z.-W. Zhang, X.-Y. Jing, and T.-J. Wang, "Label propagation based semi-supervised learning for software defect prediction", Automated Software Engineering, vol. 24, pp. 47–69, 2017.

[42]. X. Du, H. Yue, and H. Dong, "Software defect prediction method based on hybrid sampling" in International Conference on Frontiers of Electronics, Information and Computation Technologies, ser. ICFEICT 2021. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3474198.3478215.

[43]. D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost- sensitive boosting approach for cross-project defect prediction", Software Quality Journal, vol. 25, pp. 235–272, 2017.

[44]. L. Zhou, R. Li, S. Zhang, and H. Wang, "Imbalanced data processing model for software defect prediction", Wireless Personal Communications, vol. 102, pp. 937–950, 2018.

[45]. H. He, X. Zhang, Q. Wang, J. Ren, J. Liu, X. Zhao, and Y. Cheng, "Ensemble multiboost based on ripper classifier for prediction of imbalanced software defect data", IEEE Access, vol. 7, pp. 110 333–110 343, 2019.

[46]. C. Zeng, C. Y. Zhou, S. K. Lv, P. He, and J. Huang, "Gcn2defect: Graph convolutional networks for smotetomek based software defect prediction" in 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2021, pp. 69–79.

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

[48]. L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction", Software Quality Journal, vol. 26, pp. 97–125, 2018.

[49]. S. Riaz, A. Arshad, and L. Jiao, "Rough noise-filtered easy ensemble for software fault prediction", IEEE Access, vol. 6, pp. 46 886–46 899, 2018.

[50]. X. Wan, Z. Zheng, and Y. Liu, "Spe2: Self-paced ensemble of ensembles for software defect prediction", IEEE Transactions on Reliability, vol. 71, no. 2, pp. 865–879, 2022.

[51]. G. Menardi and N. Torelli, "Training and assessing classification rules with imbalanced data", Data Mining and Knowledge Discovery, vol. 28, no. 1, pp. 92–122, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10618-012-0295-5.

[52]. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-samplingtechnique", Journal of Artificial Intelligence Research, vol. 16, no. nil, pp. 321–357, 2002. [Online]. Available: http://dx.doi.org/10.1613/jair.953.

[53]. H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning" in 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 6 2008, p. nil. [Online]. Available: http://dx.doi.org/10.1109/IJCNN.2008.4633969.

[54]. G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data", ACM SIGKDD Explorations Newsletter, vol. 6, no. 1, pp. 20–29, 2004. [Online]. Available: http://dx.doi.org/10.1145/1007730.1007735.

[55]. I. Mani and I. Zhang, "knn approach to unbalanced data distributions: a case study involving information extraction" in Proceedings of workshop on learning from imbalanced datasets, vol. 126, no. 1. ICML, 2003, pp. 1–7.

[56]. D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-2, no. 3, pp. 408–421, 1972.

[57]. I. Tomek, "An experiment with the edited nearest- neighbor rule", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-6, no. 6, pp. 448–452, 1976. [Online]. Available: http://dx.doi.org/10.1109/TSMC.1976.4309523.

[58]. B. R. Manju and A. R. Nair, “Classification of Cardiac Arrhythmia of 12 Lead ECG Using Combination of SMOTEENN, XGBoost and Machine Learning Algorithms” in 2019 9th International Symposium on Embedded Computing and System Design (ISED), 2019, pp. 1–7.
[59]. G. E. A. P. A. Batista, A. L. C. Bazzan, and M. C. Monard, “Balancing training data for automated annotation of keywords: a case study” in WOB, 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:1579194
[60]. I. Tomek, “Two modifications of cnn”, IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-6, no. 11, pp. 769–772, 1976.
[61]. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees, ser. Routledge, 2017. [Online]. Available: http://dx.doi.org/10.1201/9781315139470.
[62]. D. A. Cieslak and N. V. Chawla, “Learning decision trees for unbalanced data” in Machine Learning and Knowledge Discovery in Databases, W. Daelemans, B. Goethals, and K. Morik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 241–256.
[63]. E. Fix and J. L. Hodges, “Discriminatory analysis. nonparametric discrimination: Consistency properties”, International Statistical Review / Revue Internationale de Statistique, vol. 57, no. 3, p. 238, 1989. [Online]. Available: http://dx.doi.org/10.2307/1403797.
[64]. L. Breiman, “Random forests”, Machine Learning, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/ 10.1023/A:1010933404324
[65]. Y. Freund and R. E. Schapire, “Experiments with a new boosting algorithm” in Proceedings of the Thirteenth International Conference on International Conference on Machine Learning, ser. ICML’96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, p. 148–156.
[66]. J. H. Friedman, “Stochastic Gradient boosting”, Computational Statistics Data Analysis, vol. 38, no. 4, pp. 367–378, 2002, nonlinear Methods and Data Mining. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S01 67947301000652.
[67]. D. J. Hand and K. Yu, “Idiot’s bayes-not so stupid after all?” International Statistical Review, vol. 69, no. 3, pp. 385–398, 2001. [Online]. Available: http://dx.doi.org/10.1111/j.1751-823.2001.tb00465.x.
[68]. G. E. Hinton, Connectionist learning procedures, ser. Machine Learning. Elsevier, 1990, pp. 555–610. [Online]. Available: http://dx.doi.org/10.1016/B978- 0-08-051055-2.50029-8.
[69]. T. Dyba, V. B. Kampenes, and D. I. Sj.berg, “A systematic review of statistical power in software engineering experiments”, Information and Software Technology, vol. 48, no. 8, pp. 745–755, 2006.
[70]. J. Sánchez-García, “Statistical tests among groups”, [Data set], 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13239734.
[71]. D. S. Moore and G. P. McCabe, Introduction to the practice of statistics. WH Freeman/Times Books/Henry Holt & Co, 1989.
[72]. J. Sánchez-García, “Statistical tests results”, [Data set], 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13240040.
[73]. R. Malhotra and M. Khanna, “Threats to validity in search based predictive modelling for software engineering”, IET Software, vol. 12, no. 4, pp. 293-305, 2018.
[74]. I. Bronshteyn, “Study of defects in a program code in python”, Programming and Computer Software, vol. 39, pp. 279–284, 2013.
[75]. A. Belevantsev, “Multilevel static analysis for improving program quality”, Programming and Computer Software, vol. 43, pp. 321–336, 2017.

Информация об авторах / Information about authors

Ангел Хуан САНЧЕС-ГАРСИЯ имеет степень PhD по искусственному интеллекту, доцент факультета статистики и информатики Университета Веракруса (Мексика). Сфера научных интересов: программометрия, машинное обучение, прогнозирование затрат, эволюционные вычисления.

Ángel Juan SÁNCHEZ-GARCÍA – doctor in artificial intelligence, associate professor at Facultad de Estadística e Informática, Universidad Veracruzana, Mexico (School of Statistics and Informatics, University of Veracruz). Research interests: software measurement, machine learning, effort prediction, evolutionary computation.

Рианьо Гектор Ксавьер ЛИМОН имеет степень PhD по искусственному интеллекту, доцент факультета статистики и информатики Университета Веракруса (Мексика). Сфера научных

интересов: интеллектуальный анализ данных, мультиагентные и распределенные системы, архитектура программного обеспечения.

Riaño Hector Xavier LIMÓN – doctor in artificial intelligence, associate professor at Facultad de Estadística e Informática, Universidad Veracruzana, Mexico (School of Statistics and Informatics, University of Veracruz). Research interests: data mining, multi-agent systems, distributed systems, and software architecture.

Саул ДОМИНГЕС-ИСИДРО имеет степень PhD по искусственному интеллекту, доцент факультета статистики и информатики Университета Веракруса (Мексика). Сфера научных интересов: распределенные системы, разработка программного обеспечения, вычислительный интеллект, машинное обучение.

Saúl DOMÍNGUEZ-ISIDRO – PhD in artificial intelligence, associate professor at Facultad de Estadística e Informática, Universidad Veracruzana, Mexico (School of Statistics and Informatics, University of Veracruz). Research interests: distributed systems, software development, computational intelligence, and machine learning.

Дан Хавьер ОЛВЕРА-ВИЙЕДА имеет степень бакалавра в области программной инженерии, сотрудник факультета статистики и информатики Университета Веракруса (Мексика). Сфера научных интересов: разработка программного обеспечения, вычислительный интеллект, машинное обучение.

Dan Javier OLVERA-VILLEDA – bachelor in software engineering, Facultad de Estadística e Informática, Universidad Veracruzana, Mexico (School of Statistics and Informatics, University of Veracruz). Research interests: software development, machine learning, and computational intelligence.

Хуан Карлос ПЕРЕС-АРРИАГА имеет степень магистра по программированию, доцент факультета статистики и информатики Университета Веракруса (Мексика). Сфера научных интересов: доступность программного обеспечения, безопасность программного обеспечения, конструирование программ.

Juan Carlos PÉREZ-ARRIAGA has a master’s degree in computer science, an associate professor at Facultad de Estadística e Informática, Universidad Veracruzana, Mexico (School of Statistics and Informatics, University of Veracruz). Research interests: software accessibility, software security, and software construction.