e-Informatica Software Engineering Journal, Volume 17, Issue 1, 2023, pages: 230102, DOI: 10.37190/e-Inf230102

The Effect of Dual Hyperparameter Optimization on Software Vulnerability Prediction Models

Deepali Bassi*, Hardeep Singh*
*Department of Computer Science, Guru Nanak Dev University
deepalics.rsh@gndu.ac.in, hardeep.dcse@gndu.ac.in

Abstract

Background: Prediction of software vulnerabilities is a major concern in the field of software security. Many researchers have worked to construct various software vulnerability prediction (SVP) models. The emerging machine learning domain aids in building effective SVP models. The employment of data balancing/resampling techniques and optimal hyperparameters can upgrade their performance. Previous research studies have shown the impact of hyperparameter optimization (HPO) on machine learning algorithms and data balancing techniques.
Aim: The current study aims to analyze the impact of dual hyperparameter optimization on metrics-based SVP models.
Method: This paper has proposed the methodology using the python framework Optuna that optimizes the hyperparameters for both machine learners and data balancing techniques. For the experimentation purpose, we have compared six combinations of five machine learners and five resampling techniques considering default parameters and optimized hyperparameters.
Results: Additionally, the Wilcoxon signed-rank test with the Bonferroni correction method was implied, and observed that dual HPO performs better than HPO on learners and HPO on data balancers. Furthermore, the paper has assessed the impact of data complexity measures and concludes that HPO does not improve the performance of those datasets that exhibit high overlap.
Conclusion: The experimental analysis unveils that dual HPO is 64% effective in enhancing the productivity of SVP models.

Keywords: software vulnerability, hyperparameter optimization, machine learning algorithm, data balancing techniques, data complexity measures

1. Introduction

With the advent of information technology, Software is the main component of devices and systems on which modern life is dependent. Software Vulnerabilities are mistakes, errors, flaws, weaknesses, or loopholes caused during the specification, design, development, or configuration of the software [1]. We can say that a lack of programming practices [2] may lead to software vulnerability which may further provide a gateway for attacks, thereby hampering the confidentiality, integrity, and availability of the information systems.

Submitted: 30 Jun. 2022; Revised: 8 Oct. 2022; Accepted: 18 Nov. 2022; Available online: 5 Dec. 2022

Attackers make use of software vulnerabilities to attack the system. Once access is obtained, security is at stake and any valuable information can be stolen from it, for example, theft of email passwords, debit card information, etc., or the system can be corrupted. To curb the intrusion of attackers, software free from vulnerabilities needs to be developed using security techniques along with secured design principles and software development life cycles [3]. Conventional security techniques include static analysis [4] such as penetration testing [5], fuzz-testing [6], etc., dynamic analysis such as tainted data-flow analysis [7], code inspections [8], etc., and hybrid analysis [9]. Static analysis has the problem of a high false-positive rate [10]. Code reviews are time-consuming and cannot be easily performed on large systems because of time constraints, and scarcity of validation and verification resources. To tackle these limitations to an extent, researchers worked on developing new prediction models based on machine learning algorithms. The time of the developers is reduced by early prediction of the most vulnerable components.

Predictive modeling calculates or predicts future outcomes using historical data and aids in prioritizing the efforts of software testing. The prediction models consist of independent variables (predictors) and dependent variables (outcomes). They are built after collecting historical data for relevant predictors. In the domain of software engineering, various fault prediction and defect prediction models are proposed for predicting faults and defects and enhancing the efficiency of software testing plans [11]. Therefore, to predict vulnerabilities researchers developed SVP models. Although faults and vulnerabilities are different, yet construction of their respective models is almost similar [12]. SVP machine learning-based models classify the software components into vulnerable or non-vulnerable categories depending on different levels of granularity such as file, package, class, or method. They are categorized as metrics-based, text-mining-based, and a combination of both. Metrics-based models are where metrics determine the vulnerable components. Text-mining-based models are where the conversion of source code into tokens and frequencies predicts the vulnerable components. A combination of both predicts the vulnerability by combining the metrics and text features.

Vulnerabilities are hard to find, as it requires attack patterns knowledge and understanding of the source code. Due to security concerns, developers publish a limited number of vulnerabilities compared to faults [13] so, vulnerabilities are subgroups of faults. The class imbalance problem has always been an issue as prediction models favor the majority class (over-represented concept) over the minority class (under-represented concept). Various studies have managed to tackle it by using data-level methods [14–18], algorithm-level methods [19], and ensemble learning methods [20].

1.1. Motivation

The main challenge faced by the SVP models is their performance which is affected by the imbalanced datasets and the hyperparameter settings of machine learning techniques. Therefore, to enhance the efficacy of prediction models, recent studies have used various combinations of resampling techniques and optimized machine learning methods [21–25] in the areas of software engineering. By far, studies have optimized the hyperparameters of machine learners and applied data balancing techniques to balance the dataset but optimization of resamplers has only been explored in a few studies [26–29]. In [26], Six imbalanced datasets from the Keel collection are used. The current study aims to analyze the effect of hyperparameter optimization (HPO) when applied to both machine learners and resamplers called dual HPO on the performance of metrics-based SVP models which

has not been performed on the PHP dataset before. In addition to this, it has been observed that the capability of machine learners is not only hampered by imbalanced datasets but also by the degree of class overlapping, which motivates us to further anatomize the problem with the degree of class overlapping for the cases where efficiency is not improved. Papers [27–29] have worked on bug prediction models and worked on textual data.

1.2. Contributions

In this study, we have replicated the six scenarios of HPO used in [26] on three publicly available datasets PHPMyAdmin, Moodle, and Drupal [30]. We have used five supervised machine learning algorithms and five resampling techniques which are highly used in past studies. This paper revolves around the construction of metrics-based SVP models using

Table 1. Scenarios for hyperparameter optimization

Scenario	Machine Learning Methods	Resampling Techniques
1. Ad + Rn	Default hyperparameters	No Resampling
2. Ao + Rn	Optimized hyperparameters	No Resampling
3. Ad + Rd	Default hyperparameters	Default hyperparameters
4. Ao + Rd	Optimized hyperparameters	Default hyperparameters
5. Ad + Ro	Default hyperparameters	Optimized hyperparameters
6. Ao + Ro	Optimized hyperparameters	Optimized hyperparameters

the six scenarios mentioned in Table 1. The comparative analysis of all the scenarios is performed, and significant improvement is calculated using Wilcoxon signed-rank test, and Bonferroni correction is applied as there are six statistical tests. The effect of hyperparameter tuning and resampling on the prediction models is demonstrated through the outcomes of our experiments performed in the python framework Optuna. The study has the following major contributions:

RQ 1) How much is dual HPO effective in improving the performance of SVP models?
Dual HPO is the optimization of hyperparameters of both machine learners and resampling techniques. Since HPO is known to improve the productivity of SVP models [21–25] the aim is to check whether applying dual HPO increases their performance.

RQ 2) Is dual HPO better than other HPO scenarios?
This question aims to find whether the performance improvements by dual HPO compared with AdRd are better than HPO scenarios AoRd and AdRo when each is compared with AdRd.

RQ 3) How has the degree of class overlapping affected the HPO?
As mentioned in [26] the performance of SVP is also affected by the degree of class overlapping which is measured by the data complexity measures. This paper has used two measures imbalance ratio and maximum Fisher's discriminant ($F1$). The current study is the first attempt to search for the cause of no improvement in the efficiency of models even after applying HPO.

RQ 4) Which resampling technique has performed the best?
The current study has applied five resampling techniques namely SMOTE, Adasyn, Borderline SMOTE, SMOTE + Tomek links, and SMOTE + Edited Nearest Neighbors to balance the datasets. Four scenarios in Table 1 have used resampling with default and optimized parameters respectively. So, the goal of this research question is to find the best resampling technique across all datasets and machine learning techniques.

1.3. Paper organization

The remaining sections of the current study are structured as Section 2 presents the related work, Section 3 provides the methodology, Section 4 describes the experimental design, Section 5 explains the results, Section 6 discusses the findings of the study, Section 7 shows threats to validity, and Section 8 concludes the paper and provides the future scope of the study.

2. Related work

Software vulnerability provides the gateway for attackers to damage the software systems. Therefore, research studies have focused on predicting the software vulnerable components effectively using machine learning algorithms. There exists a large amount of work that exhibits machine-learning techniques for the construction of SVP models. The performance of models is increased by using machine learners but more research needs to be done on the factors affecting them, i.e., hyperparameter tuning, imbalanced datasets, and the degree of class overlapping. In this paper, we are trying to improve machine learners' performance by understanding the role of these factors.

2.1. Class imbalance in prediction models

Ghaffarian and Shahriari [1] have presented a survey paper showing machine learning and data-mining techniques to mitigate the impact of software vulnerability. It has encapsulated various definitions of software vulnerability. Various works related to vulnerability prediction models have been reviewed in this paper. Also, it has been mentioned that the vulnerability datasets are imbalanced and affect the productivity of machine learning algorithms.

Kaya et al. [3] have described the effect of feature types, machine learners, and resamplers on SVP models. It has included three feature types' metrics, text, and a combination of both. It has experimented using seven machine learners namely random forest, linear support vector machine, weighted k-nearest neighbors, adaboost, rusboost, linear discriminant, subspace discriminant and four resamplers such as SMOTE, cluster SMOTE, borderline SMOTE, and adasyn. The datasets used are Drupal, Moodle, and PHPMyAdmin. For experimental evaluation, the performance metrics used are precision, recall, AUC, $F1$-score, and specificity. The paper concludes that random forest has performed the best for smaller datasets Drupal and PHPMyAdmin, and rusboost for larger datasets, i.e., Moodle.

Wang and Yao [15] have used under-sampling techniques, ensemble-learning techniques, and threshold moving techniques on two classifiers (naive bayes and random forest) to address the class imbalance issue in software defect prediction (SDP) models. It has been concluded that balanced random under-sampling shows better defect prediction but lesser than naive bayes. Adaboost has turned out to be the best performer in improving the efficiency of SDP models. The overall performance is assessed using G-mean, AUC, and balance metrics.

Sasada et al. [16] have raised the data complexity issue caused by resampling techniques that affect the accuracy of the predictive model. The performance metrics used are accuracy and f-measure. SMOTE + ENN and SMOTE + TL are proposed for handling the noise and overlap in the dataset. The proposed method is effective in four out of ten datasets. Borowska and Stepaniuk [17] have studied the behavior of under-sampling and over-sampling methods

on imbalanced datasets. The experimental results illustrated that SMOTE + ENN and SMOTE + TL provide good results for datasets with few positive instances and random oversampling (ROS) gives good results when there are more positive instances.

Walden et al. [30] have stated that previous works were done on Java, C++, and C projects so they have proposed the vulnerability datasets based on PHP open-source projects as most of the vulnerabilities are found in web applications. The datasets provided include software metrics and text features. The paper has worked on both software metrics-based and text mining-based SVP models and found that text mining-based models perform better than metrics-based ones. Experiments include random forest machine learner and under-sampling technique for balancing the dataset and the performance metrics used are recall and inspection ratio.

Stuckman et al. [31] have extended the work in [30] by inspecting the impact of dimensionality reduction techniques (feature selection, principal component analysis, and confirmatory factor synthesis) on metrics and text-mining features. It has implemented SMOTE and under-sampling technique for data balancing and found that SMOTE has shown lower recall, lower inspection rate, and higher f-measure so; it is preferred over the under-sampling method. In addition to this, dimensionality reduction techniques worked well for cross-project prediction than within-project prediction.

2.2. Hyperparameter tuning

Hyperparameter tuning in recent studies has been observed to improve the efficiency of prediction models. A lot of studies exist on hyperparameter tuning of defect prediction and bug prediction models which motivated us to explore the same for vulnerability prediction models.

Tantithamthavorn et al. [21] have shown the effect of optimal parameter settings on the defect prediction model (DPM) by using the caret technique (automated parameter optimization technique) and concluded that on optimizing the parameters of classifiers, the performance of DPM has improved by 40% in terms of AUC evaluation metric. The paper has suggested experimenting with different parameter tuning of machine learning classifiers.

Rijn and Hutter [22] have employed a hundred datasets from OpenML to find the important hyperparameters for the random forest, adaboost, and support vector machine algorithms. This paper projects the idea of optimizing important hyperparameters rather than optimizing all the hyperparameters. We have considered some of these important hyperparameters to be tuned as per our research requirement.

Yang and Shami [24] have identified the hyperparameters for machine learning algorithms through their work. The study has discussed the HPO problem in detail and has explained different HPO algorithms and frameworks with their advantages and disadvantages. The range for hyperparameters of (both classifiers and regressors) random forest, k-nearest neighbor, and support vector machine. For our study, we have used classifiers' ranges.

Shu et al. [25] perform parameter tuning of machine learners and data pre-processors to classify bug reports. A comparison of FARSEC and HPO is performed. It is observed that on applying HPO, the recall has improved from 35% to 65% with an increased false-positive rate. Also, optimizing data pre-processors produces better performance results than optimizing machine learners. The paper used five machine learners such as random forest, logistic regression, naïve bayes, k-nearest neighbor, and multilayer perceptron.

Claesen and Moor [32] have discussed the challenges in searching hyperparameters for the machine learning algorithm. Also, current approaches for searching hyperparameters are

mentioned in the paper. Kudjo et al. [33] have discussed the importance of parameter tuning on three PHP datasets Drupal, Moodle, and PHPMyAdmin. The paper has compared the results for random forests with the benchmark study by [30] and found an increase in precision, recall, and accuracy for Drupal and PHPMyAdmin whereas for Moodle only accuracy has increased. The paper has not included balancing techniques. In addition to this, the accuracy metric gives biased results for imbalanced datasets.

Sara et al. [34] use the concept of data balancing and HPO on maintainability or bug prediction models. The paper uses SMOTE and grid search on five machine learning algorithms (k-nearest neighbor, support vector machine, decision tree, naive bayes, and multilayer perceptron). The evaluation metrics involve sensitivity, specificity, accuracy, and precision. Grid search is better than default settings in all datasets but there is no existence of the best machine learning technique for all datasets. However, it concludes that tuning hyperparameters and balanced data helps in obtaining the best productivity of machine learning methods.

Osman et al. [35] have worked on optimizing hyperparameters of two machine learning algorithms (k-nearest neighbor and support vector machine) using five open-source java systems. Hyperparameter tuning improves the accuracy of bug prediction models such as k-nearest neighbor became a better predictive model after HPO. The paper shows how different machine learning algorithms are compared after hyperparameter tuning.

2.3. Class overlapping

Almutairi and Janicki [14] discuss the class imbalance problem and the impact of overlapping on imbalanced data. It has compared and analyzed three machine learning algorithms (decision tree, k-nearest neighbor, and support vector machine) using six combinations of overlapping and imbalance. For the evaluation criteria accuracy, precision, and recall are used. The findings show that imbalanced data has less overlap than balanced data. It further concluded that the performance of the predictive model not only depends on resampling techniques but also on the degree of overlapping in the datasets.

Barella et al. [36] have explained the class overlapping issue in imbalanced binary classification. It states that various research studies have focused on balancing methods but that works well when the classes are linearly separable. The class overlap is measured by various data complexity measures mentioned in Ho and Basu [37] and Sotoca et al. [38] that divide the data complexity measures into three categories: (i) measures of feature overlap, (ii) class separability measures, and (iii) measures of geometry and topology.

Furthermore, Lorena et al. [39] have divided these measures into the following feature-based measures linearity measures, neighborhood measures, network measures, dimensionality measures, and class imbalance measures. Also, it has mentioned the application areas for data complexity measures such as data analysis, data pre-processing tasks, learning algorithms, and meta-learning. For our study, since we need to deal with data imbalance which is one of the data pre-processing tasks, therefore, data complexity measures are applied to see their impact on the classification of imbalanced datasets.

2.4. Dual hyperparameter optimization

Shu et al. [27] extended the paper [25] by incorporating dual optimization approach (SWIFT) for bug prediction. The research paper concludes that SWIFT is better than optimizing data pre-processor and learners individually. Kong et al. [26] show the impact

Table 2. Comparisons with existing works

Research work	Machine learning techniques used	Evaluation metrics used	Resampling	HPO	Dual HPO	Data complexity measures	Datasets
Kaya et al. [3]	RF, AB, Linear Discriminant, Linear SVM, Weighted KNN, Subspace Discriminant, Rusboost	*AUC*, Precision, Recall, *F*-Score, Specificity	SMOTE, Adasyn, ClusterSMOTE, BLSMOTE	NO	NO	NO	PHP
Shu et al. [25]	RF, NB, Logistic Regression, Multilayer Perceptron, *k*-nearest Neighbors	Precision, Recall	SMOTE	YES	NO	NO	Chromium, Apache projects (Ambari, Wicket, Camel, Derby)
Walden et al. [30]	RF	Recall, Inspection Ratio	Under-sampling	NO	NO	NO	PHP
Stuckman et al. [31]	RF	*Recall*, *F*1-Score, Inspection Ratio	Under-sampling, SMOTE	NO	NO	NO	PHP
Kudjo et al. [33]	RF, KNN, SVM, Decision Tree	Precision, Recall, Accuracy	No Resampling	YES	NO	NO	PHP
Zhang et al. [40]	RF, NB, Decision Tree	Recall, Precision, Accuracy	No Resampling	NO	NO	NO	PHP
Abunadi et al. [41]	NB, Logistic Regression, SVM, RF, Decision Tree	Precision, Recall, F-Measure	No Resampling	NO	NO	NO	PHP
Khalid et al. [42]	RF, NB, Decision Tree	*Accuracy*, *Precision*, *Recall*, *F*1-Score	SMOTE	NO	NO	NO	PHP
Catal et al. [43]	Averaged Perceptron, Bayes point machine, Boosted Decision Tree, Decision Forest, Decision jungle, Deep SVM, SVM, Logistic Regression, Multilayer Perceptron	*AUC*	No Resampling	NO	NO	NO	PHP
Shu et al. [27]	RF, NB, Logistic Regression, Multilayer Perceptron, *k*-nearest Neighbors	*Precision*, *Recall*, *F*1-score, False alarms, G-mean	SMOTE	YES	YES	NO	Chromium, Apache projects (Ambari, Wicket, Camel, Derby)
Kong et al. [26]	RF, SVM	*AUC*	SMOTE, Adasyn, SMOTE + Tomek links, SMOTE + Edited Nearest Neighbor	YES	YES	YES (maximum Fisher's discriminant)	Keel-Collection
Proposed Work	RF, AB, NB, KNN, SVM	*AUC*, *F*1-Score	SMOTE, Adasyn, BLSMOTE, SMOTE + ENN, SMOTE + TL	YES	YES	YES (maximum Fisher's discriminant and imbalance ratio)	PHP

of resampling and HPO on the performance of machine learning algorithms. Also, tuning the hyperparameters of both resamplers and learners is emphasized. The area under the ROC curve is used for performance evaluation. The experiments are performed on two machine learning algorithms (random forest and support vector machine) that consider six combinations of HPO (learners and resamplers). Also, data complexity measures show that hyperparameter optimization works for datasets with low overlap than datasets with high overlap. Existing works have encouraged us to propose a methodology that assimilates data imbalance, hyperparameter tuning, and class overlapping areas to increase the efficacy of SVP models.

Agrawal et al. [28] have also applied HPO on pre-processor and machine learners for defect prediction models where the "dodge" technique is applied to reduce the CPU cost caused by HPO. It eliminates duplicate hyperparameter tunings but is efficient for data with low dimensionality; therefore Agrawal et al. [29] extended the work for high-dimensionality data. In [28] 10 SE defect prediction datasets and 6 SE issue tracking datasets were used. In addition to this, [29] has included 63 SE datasets that explore Github issue close time, 4 SE datasets for bad smell detection, and 37 non-SE problems from the UCI repository hence considering datasets with high dimensionality.

2.5. Comparisons with existing works

Table 2 has compared our work with the research studies based on machine learning techniques used, resampling techniques, evaluation metrics, whether HPO exists, dual HPO exists, and whether data complexity measure exists. It has been inferred from Table 2 that most of the studies lack HPO, dual HPO, and data complexity measures. In [27], only data complexity measures are missing. Since [26] has included all the factors but on the Keel-collection datasets, only two machine learning techniques, four resampling techniques, and one data complexity measure. In addition to this, only (Ao + Ro) is compared with (Ad + Rd).

The proposed work has replicated this idea on the PHP dataset with three added machine learning algorithms, i.e., gaussian naïve bayes, adaboost, and k-nearest neighbor, one added resampler namely Borderline SMOTE, and one added data complexity measure such as imbalance ratio. Furthermore, six scenarios are compared and mentioned in Section 4.2.

3. Research methodology

This section explains the research methodology stating the datasets used (Section 3.1), machine learning methods (Section 3.2), resampling techniques (Section 3.3), hyperparameter optimization (Section 3.4), performance evaluation metrics (Section 3.5), and data complexity measures (Section 3.6) used in the study.

3.1. Experimental datasets

The experiments are performed on three open-source publicly available datasets[1], namely Drupal is a content management system, PHPMyAdmin is an open-source administration for MySQL, and Moodleis a learning management system. These datasets have been used

[1]http://seam.cs.umd.edu/webvuldata

in recent studies mentioned in the related work section. The level of granularity of datasets is file and each file is labeled with "no" (no vulnerability exists) or "yes" (at least one vulnerability exists from vulnerability type). There exist vulnerable types such as code injection, cross-script request forgery (CSRF), cross-site scripting (XSS), path disclosure, authorization issues, and others related to phishing or man-in-the-middle vulnerabilities. The current paper focuses on binary classification as the dataset available are labeled with two values "NO" and "YES". There exist both metrics and text mining datasets1. For this work, only metrics-based datasets are considered. The datasets downloaded have comma-separated values which are further preprocessed and saved. Table 3 shows the Drupal vulnerability prediction dataset after preprocessing. Each column header is the software metric of the dataset. There are 13 software metrics (Independent variable) and the column "IsVulnerable" (dependent variable) shows the labeling of each file:

– "nonecholoc": non HTML lines of code;
– "loc": total number of lines of code in a PHP file;
– "nmethods": total number of functions in the file;
– "ccomdeep" and "ccom": cyclomatic complexity, i.e., the number of independent paths. Since these two metrics have the same values so can be considered as one metric;
– "nest": maximum nesting complexity, i.e., maximum depth of the nested loops;

Table 3. Example of Drupal dataset

non echoloc	loc	nmethods	ccom deep	ccom	nest	hvol	nIncoming Calls	nIncoming CallsUniq	nOutgoing InternCalls	nOutgoing ExternFlsCalled	nOutgoingExtern FlsCalledUniq	nOutgoingExtern CallsUniq	Is Vulnerable
4	4	0	1	1	0	3.29	0	0	0	2	2	2	0
126	126	9	26	26	4	1402.86	18	6	2	18	8	35	1
168	168	10	29	29	3	1455.88	4	4	8	19	9	32	0
412	412	35	77	77	5	4929.76	361	119	23	30	12	30	1
10	10	3	1	1	0	29.93	54	33	0	0	0	0	0
53	53	3	16	16	4	583.07	53	32	1	14	6	24	0
1355	1355	92	251	251	8	17945.97	698	147	38	62	19	137	1
162	162	13	31	31	3	1702.981	59	34	4	24	10	16	1
172	172	21	29	29	2	1854.86	113	77	7	17	5	35	0
131	131	19	14	14	1	1455.23	176	79	3	16	6	25	0
135	135	19	14	14	1	1510.92	176	79	3	16	6	25	0
328	328	42	51	51	2	4440.29	286	84	15	27	6	67	0
381	381	27	84	84	6	5256.54	48	15	16	24	9	82	1
896	896	70	197	197	6	13130.98	107	51	27	38	11	108	1
107	107	9	18	18	3	941.85	1	1	3	4	4	14	0
77	77	9	13	13	3	917.13	8	4	3	4	3	4	0
361	361	22	86	86	6	4169.19	33	9	8	26	10	39	0
60	60	2	11	11	1	360.58	0	0	1	2	2	16	0
59	59	2	10	10	1	413.19	0	0	1	2	2	16	0
74	74	2	16	16	1	473.45	0	0	1	2	2	18	0

– "hvol": Halstead's volume is calculated using the number of total operands and operators with the number of unique operators and operands;
– "nIncomingCalls": fan-in, i.e., number of files that call the function from the measured file;
– "nIncomingCallsUniq": internal methods that are called by the statement in the measured file;
– "nOutgoingInternCalls": fan-out is the number of files called from the measured file;
– "nOutgoingExternFlsCalled": total external calls are the number of methods from other files called in the measured file;
– "nOutgoingExternFlsCalledUniq": external methods called are the number of functions called from the measured file and also included in other files;
– "nOutgoingExternCalls": external calls to methods are the number of files that calls methods in the measured file.

Table 4 describes the version of the project, total files, vulnerable files, Imbalance Ratio (IR), no of vulnerabilities, Maximum Fisher's Discriminant Ratio ($F1$), and text features. As per equation (8) mentioned in Section 3.6, IR for each dataset is calculated and it is observed that all three datasets are imbalanced, Moodle being highly imbalanced with IR 120.8. Drupal and PHPMyAdmin are imbalanced with IR 2.25 and 10.92, respectively. Equation (7) in Section 3.6 calculates the $F1$ values of the datasets. Moodle is less overlapped with the $F1$ value of 0.8098. Drupal is moderately overlapped, having a 0.6229 $F1$ value, and PHPMyAdmin is highly overlapped with an $F1$ of 0.3195.

Table 4. Dataset descriptions

Dataset	Version	Total Files	Vulnerable Files	IR	Vulnerabilities	$F1$	Text Features
Drupal	6.0	202	62	2.25	97	0.6229	3886
PHPMyAdmin	3.3.0	322	27	10.92	75	0.3195	5232
Moodle	2.0.0	2924	24	120.8	51	0.8098	18306

3.2. Machine learning methods

Machine learning methods consist of supervised and unsupervised algorithms. Supervised machine learning methods include those machine learning algorithms where input features are mapped to the target using labeled data. These include Linear Regression, Logistic Regression, Naive Bayes (NB), k-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), Support Vector Machines (SVM), etc. Various ensemble learning methods are Bagging, Voting, Adaboost, etc. In this paper, five supervised machine learning methods; RF, AB, Gaussian NB, SVM, and KNN are used. In paper [44], we have studied the effect of hyperparameter optimization on eight machine learning algorithms that are widely used in research studies as per Table 2 and have different training strategies. Further, we have considered the top five machine learners with high AUC values and extended our work on those techniques.

3.2.1. Random forest (RF)

Most studies use the RF algorithm as mentioned in Table 2; therefore, it is selected for the current study. It is a supervised learning technique based on the concept of ensemble

learning. By counting the votes (classification output) of multiple decision trees constructed from randomly generated subsets in the forest, the class with the majority votes is selected to be the classification output of the RF classifier. It can perform classification and regression tasks. It can handle large datasets and prevent the over-fitting issue [45].

3.2.2. Adaboost (AB)

Adaboost, called adaptive boosting, is the boosting algorithm to build strong classifiers using several weak classifiers. Weak learners are sequentially added and trained by weighted training data. It predicts the classification output by calculating the weighted mean of the weak classifiers. It boosts the efficiency of the prediction model in binary classification problems. It can use different base learners to improve its performance. Noisy data and outliers highly affect the AB algorithm [46–48].

3.2.3. Naive Bayes (NB)

Naive Bayes is the supervised machine learning algorithm based on Bayes' theorem, assuming there is independence among the features of the class. NB models are of four types: Gaussian NB (GNB), Multinomial NB (MNB), Bernoulli NB (BNB), and Complement NB (CNB) [49] and [50].

- In GNB, the predictors follow the gaussian distribution. MNB is implemented for multinomial distributed data and used for text classification.
- BNB is used for multivariate Bernoulli distributed data where multiple features exist, with each feature to be assumed as binary-valued. CNB is a variant of MNB and is suitable for imbalanced datasets. NB is easy to implement and consumes less time but has the limitation of independence among predictors, which in real-life cases can affect the performance of the classifier.
- We used GNB in this paper as compared to CNB because the former has performed better than the latter [51].

3.2.4. Support vector machine (SVM)

SVM is a supervised machine learning algorithm used to construct a hyperplane (best decision boundary) that separates n-dimensional space into classes to put new data points in the correct category for the future. SVM are linear and non-linear used for linearly separable and non-linearly separable data, respectively. It is effective for high dimensional space, is memory efficient, and consists of various kernel functions (linear, polynomial, radial basis function, and sigmoid) used for decision function. It has the limitation of overfitting, which arises when the number of samples is much smaller than the number of features [52, 53].

3.2.5. k-nearest neighbors (KNN)

KNN is a supervised machine learning algorithm that classifies the data points by calculating the distance between them. It classifies the new data point based on the similarity with the stored data. It is crucial to determine the value of k as a small value may lead to underfitting and a large value to overfitting [54, 55].

3.3. Resampling techniques

Resampling is an approach to yielding a balanced dataset (training dataset) from an imbalanced dataset (training dataset). Resampling can be done in three ways: Under-sampling, Over-sampling, and Hybrid resampling. Under-sampling removes the majority of samples leading to loss of data and degrading the performance of the classifiers. For example, random under-sampling (RUS). Over-sampling replicates the minority samples which leads to over-fitting. It should be ensured that over-sampling is restricted only to the training dataset to avoid over-fitting issues. Examples are: random oversampling (ROS), synthetic minority oversampling technique (SMOTE), adaptive synthetic sampling approach (ADASYN), borderline SMOTE (BL SMOTE), cluster SMOTE, etc [56]. Hybrid sampling is the combination of under-sampling and over-sampling techniques such as SMOTE + edited nearest neighbor (SMOTE + ENN), SMOTE + Tomek links (SMOTE + TL), and condensed nearest neighbors + Tomek links (CNN + TL).

In this study, the main focus is on over-sampling and hybrid sampling techniques so; we have used SMOTE, ADASYN, BL SMOTE, SMOTE + ENN, and SMOTE + TL for our research.

3.3.1. SMOTE

SMOTE generates synthetic samples by creating new instances rather than duplicating the existing ones. The oversampling process occurs in the feature space. The synthetic samples are yielded along with the line segments that join k-nearest neighbors of the minority samples. It prevents over-fitting and increases the performance capabilities of the classifiers by generalizing the decision boundaries [57].

3.3.2. ADASYN

ADASYN produces synthetic samples by giving importance to minority samples that are hard to learn and minority classes having fewer samples. Density distribution is considered in ADASYN. It promotes adaptive learning and reduces learning bias [58].

3.3.3. BL SMOTE

BL SMOTE is an extended version of SMOTE. In this approach, synthetic samples are generated by selecting misclassified instances of the minority class. The instances near and on the borderline are misclassified than instances far from the borderline [59].

3.3.4. SMOTE + TL

SMOTE + TL is a hybrid sampling technique that integrates SMOTE and Tomek links to reduce the occurrence of overlap. SMOTE oversamples the minority class, and Tomek links are removed from the oversampled samples. A clear decision boundary is formed by removing the instances in the overlapping region [16].

3.3.5. SMOTE + ENN

SMOTE + ENN combines the SMOTE technique with the ENN to reduce the noise. SMOTE oversamples the minority class, and ENN removes the noisy samples. It removes

misclassified examples by its three nearest neighbors. It deeply cleanses the data more than SMOTE + TL [16].

3.4. Hyperparameter optimization

Optimal hyperparameter settings are required to improve the potential of SVP models [33]. The data learning process initializes and updates the parameters, known as model parameters, but we cannot estimate hyperparameters from this process. Hyperparameters are set before the model's training, as they configure prediction models and minimize the loss function. Hyperparameter tuning can either be done: manually or automatically.

In the case of manual tuning, various machine learning algorithms require a deep knowledge of hyperparameters. It is laborious and time-consuming for algorithms with larger hyperparameters. Due to the above limitations, the optimization of hyperparameters is automated called hyperparameter optimization (HPO). It is time-efficient, reduces human effort, aids in comparing machine learning algorithms, and determines the suitable prediction model for a particular problem [32].

There exist various HPO techniques, and selecting the apt technique is crucial. Grid search (GS), random search (RS), bayesian optimization (gaussian process, SMAC, tree-structured Parzen estimator), gradient-based optimization, multi-fidelity optimization algorithms (successive halving, hyperband, bayesian optimization hyperband), and metaheuristic algorithms (genetic algorithm, particle swarm optimization) are the HPO techniques [24]. These are applied depending on the hyperparameters such as continuous, conditional, categorical, and discrete.

Some open-source libraries to handle the HPO problems are sklearn, spearmint, bayesopt, hyperopt, optunity, and optuna [24]. For this paper, Optuna [60] is used instead of hyperopt [26] due to its advantages that search space is dynamically constructed; searching and pruning algorithms are efficient, scalable, lightweight, and distributed.

Different machine learning algorithms have different hyperparameters to be tuned. Search spaces are selected based on recent studies. Table 5 describes the hyperparameters for each machine learning algorithm, default values, and their ranges, respectively. We have chosen hyperparameters suitable for our study.

- For RF, we have taken n_estimators, max-depth, max_features, and criterion where n_estimators means the number of trees in the table, max_depth is the maximum number of trees, max_features are the maximum features to consider when searching for the best split, and criterion is the function that measures the quality of the split.
- In the case of AB, the hyperparameters chosen are: n_estimators are the maximum number of estimators where boosting is terminated, learning_rate is the weight applied to each machine learner at boosting iteration, and the algorithm is the real boosting algorithm to be used.
- For GNB, we use var_smoothing, the part of the largest variance of all features added to variances for calculation stability.
- In the case of SVM, we have used the "c" regularization parameter and kernel that describes the kernel type of the algorithm.
- For KNN, n_neighbours is the number of neighbors, weights are the weight function that describes the weights of neighbors, and leaf_size is passed to the algorithms like BallTree or KDTree.
- For SMOTE, Adasyn, and BL-SMOTE, k-neighbors are the number of neighbors, and sampling_strategy to resample the dataset are the hyperparameters to be used.

Table 5. Hyperparameter search space

Machine Learning Algorithms	Hyperparameters	Default	Range
Random Forest(RF)	n_estimators	100	[10–150]
	max_depth	None	[5–50]
	max_features	None	[0.01–1.0]
	criterion	"gini"	["gini", "entropy"]
AdaBoost(AB)	n_estimators	50	[50–500]
	learning_rate	1.0	[0.01–2.0]
	algorithm	"SAMME.R"	["SAMME", "SAMME.R"]
Gaussian Naive Bayes(GNB)	var_smoothing	1e-09	[0.0–1.0]
Support Vector Machine (SVM)	c	1.0	[1e-6, 100.0]
	kernel	"rbf"	["linear", "poly", "rbf", "sigmoid"]
k-Nearest Neighbour (KNN)	n_neighbours	5	[1–20]
	leaf_size	30	[10–100]
	weights	"uniform"	["uniform", "distance"]
SMOTE, ADASYN, BL-SMOTE	k_neighbours	5	[1–12]
	sampling_strategy	"not majority"	["minority", "not minority", "not majority", "all"]
Tomek Links(TL), Edited Nearest Neighbor(ENN)	sampling_strategy	"not majority"	["majority", "not minority", "not majority", "all"]
SMOTE + TL	sampling_strategy	"not majority",	["minority", "not minority", "not majority", "all"]
	SMOTE	None,	[1, 12], ["minority", "not minority", "not majority", "all"]
	TL	None	["majority", "not minority", "not majority", "all"]
SMOTE + ENN	sampling_strategy	"not majority",	["minority", "not minority", "not majority", "all"]
	SMOTE	None,	[1, 12], ["minority", "not minority", "not majority", "all"]
	ENN	None	["majority", "not minority", "not majority", "all"]

– In the case of SMOTE + ENN, the hyperparameters are the SMOTE object, ENN object, and sampling_strategy.
– Further, SMOTE + TL uses SMOTE object, Tomek links object, and sampling_strategy as hyperparameters.

HPO problem is defined as [61]:

$$x' = \arg\min f(x), \quad x \in \chi \tag{1}$$

where $f(x)$ is the objective function, χ is the hyperparameter search space, x is the set of best hyperparameters, and x can choose any hyperparameter from χ. HPO process is depicted in Figure 1. This paper has considered a single objective function to be minimized therefore only one performance metric is considered. To optimize more metrics multi-objective function is used.

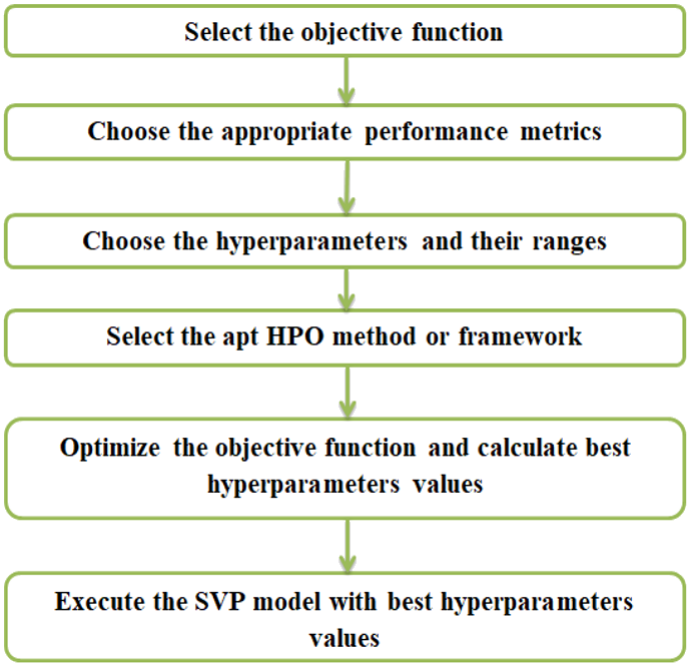

Figure 1. HPO process

3.5. Performance evaluation metrics

The accuracy metric fails in evaluating the SVP models in the imbalanced domain. Hence, we have used the area under the Receiver Operator Characteristic (ROC) curve (AUC) and $F1$-Score as the performance metric for the current study. ROC is a probability curve that plots the true positive rate ($TPrate$) against the false positive rate ($FPrate$). AUC gives the probability that the positive sample is ranked higher than the negative sample by the classifiers and is described in Figure 2. $TPrate$ is defined as the probability that the actual positive samples are correctly tested as positive whereas $FPrate$ is defined as the probability that the negative samples are tested as positive. AUC can be measured in terms of true positive rate ($TPrate$) and false positive rate ($FPrate$) as [62]:

$$Recall = TPrate = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{2}$$

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

$$FPrate = \frac{FP}{FP + TN} = \frac{FP}{N} \tag{4}$$

$$AUC = \frac{1 + TPrate - FPrate}{2} \tag{5}$$

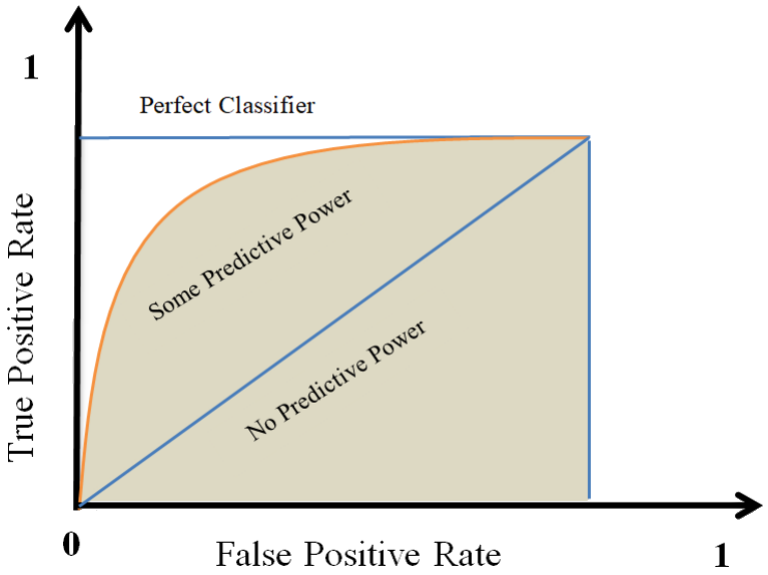

Figure 2. Area under *ROC* curve

The performance values of AUC range from 0 to 1 and these values figure out how well are classifiers at distinguishing between positive and negative classes. The high-performance model has AUC close to 1, whereas the low-performance model has AUC close to 0.5 [63]. $F1$-Score is defined as the harmonic mean of precision and recall each weighted equally.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6}$$

3.6. Data complexity measures

It is observed that the efficiency of the machine learning algorithms is not only degraded by imbalanced datasets but also by the degree of class overlapping. The degree of class overlapping is calculated by the data complexity measures [36] and [37].

The current study has focused on the feature-based measure (maximum Fisher's discriminant ratio ($F1$) and class imbalance measure (imbalance ratio). $F1$ measures the maximum discriminative power of all features in different classes and is computed as [38]:

$$F1 = \frac{(\mu c1 - \mu c2)^2}{\sigma c1^2 + \sigma c2^2} \tag{7}$$

where μci and σci are the mean and variance of values of the class i feature, respectively. The higher values of $F1$ indicate lower complexity; henceforth the classification problem is simple. The lower value of $F1$ shows that classes are highly overlapped. The imbalance ratio (IR) in the case of binary classification is calculated as [64]:

$$IR = \frac{No.\ of\ majority\ class\ instances}{No.\ of\ minority\ class\ instances} \tag{8}$$

4. Experimental framework

This section includes an Experimental procedure (Section 4.1) and a statistical test (Section 4.2). Figure 3 shows the experimental framework that illustrates the working of the model [65].

4.1. Experimental procedure

The experimental procedure (see Table 6) is based on the pseudo-code presented in [3]. The experimental methodology is illustrated through Algorithm1 and Subalgorithm.

Table 6. Summary of the information regarding the experimental setup of the current study

Datasets	Machine Learning Techniques	Resampling Techniques	Hyperparameter Optimization Method	Performance Metrics	Data Complexity Measures
Drupal, Moodle, PHPMyAdmin	RF, AB, GNB, KNN, SVM	SMOTE, ADASYN, BL SMOTE, SMOTE + TL, SMOTE + ENN	Optuna	AUC and $F1$-Score	$F1$ and IR

- Algorithm1 calculates the AUC value and $F1$-Score of each scenario for HPO explained in Table 1. The experiments are performed on three datasets given in Table 4.
- The methodology uses three-fold cross-validation repeated 50 times to maintain the random order construed in Subalgorithm. The three-fold cross-validation splits the dataset into three parts (2:1) where two parts are used for training and one part is used as the testing dataset. In other words, the training dataset is 66.66% of the entire dataset and the testing dataset is 33.33% of the whole dataset. In Algorithm1, line 5–8 calculates evaluation metrics for (Ad + Rn) and (Ao + Rn) scenarios and line 9–12 calculates evaluation metrics for (Ad + Rd), (Ao + Rd), (Ad + Ro), and (Ao + Ro) scenarios.
- There are two arrays default[] and optimized[] which stores the hyperparameters of machine learners and resamplers. Default[] indicates that the machine learners will run in their default settings whereas the optimized[] array is calculated by performing hyperparameter tuning mentioned in Section 3.4.
- For hyperparameter tuning, there is a need for a validation dataset. The training dataset (66.66%) is further split into two parts, i.e., new training dataset and one part of the validation dataset which means 44.435% is the new training dataset and 22.217% is the validation dataset. The training dataset trains the classifier with chosen hyperparameters and their ranges. An objective function is optimized (AUC in our

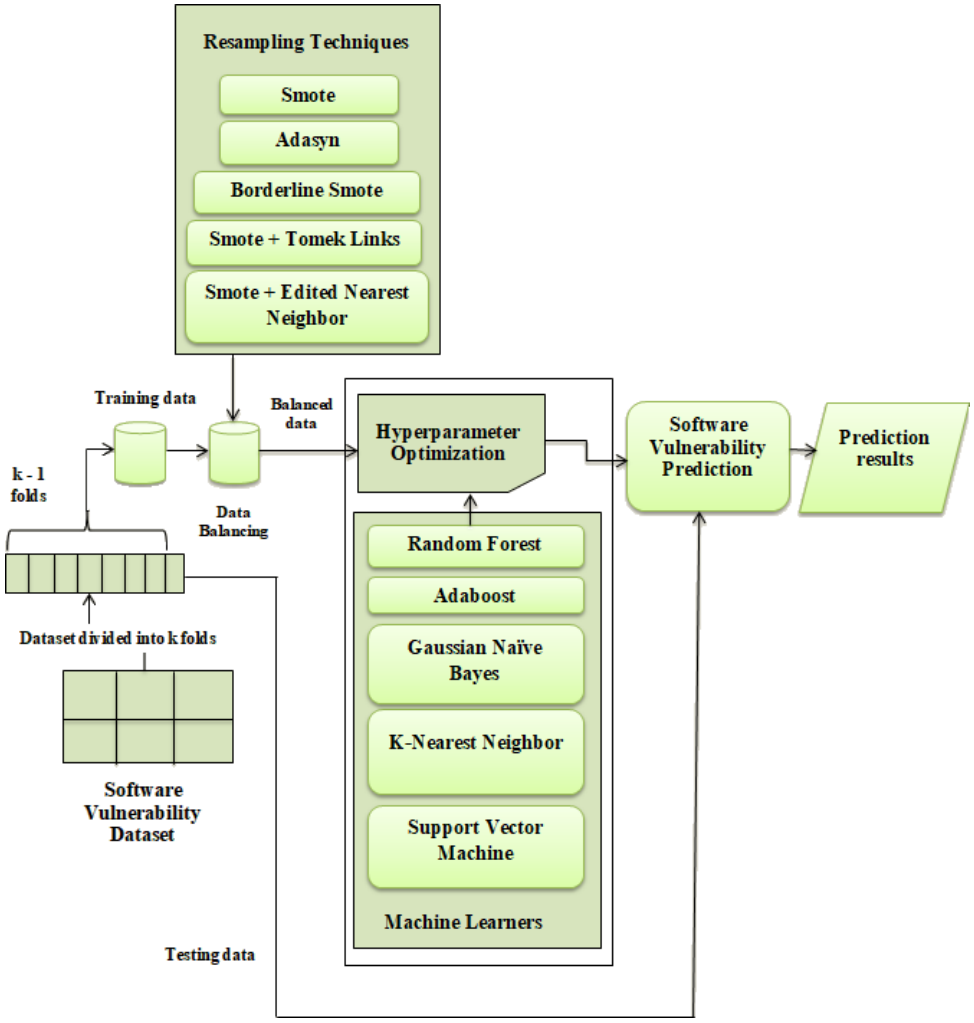

Figure 3. Working of proposed methodology

```
In [3]: study = optuna.create_study(direction='maximize')
        study.optimize(objective, n_trials=100)
        [I 2022-09-12 20:45:51,610] Trial 91 finished with value: 0.840579208548848 and parameters: {'n_estimators': 133, 'max_dept
        h': 41, 'max_features': 0.9009689339823158, 'criterion': 'entropy'}. Best is trial 88 with value: 0.8593560434926659.
        [I 2022-09-12 20:45:52,312] Trial 92 finished with value: 0.8415460651513782 and parameters: {'n_estimators': 129, 'max_dept
        h': 47, 'max_features': 0.8534884394041747, 'criterion': 'entropy'}. Best is trial 88 with value: 0.8593560434926659.
        [I 2022-09-12 20:45:53,172] Trial 93 finished with value: 0.8435169623404918 and parameters: {'n_estimators': 141, 'max_dept
        h': 46, 'max_features': 0.978168098981247, 'criterion': 'entropy'}. Best is trial 88 with value: 0.8593560434926659.
        [I 2022-09-12 20:45:53,898] Trial 94 finished with value: 0.8423431933868365 and parameters: {'n_estimators': 122, 'max_dept
        h': 45, 'max_features': 0.945286260751885, 'criterion': 'entropy'}. Best is trial 88 with value: 0.8593560434926659.
        [I 2022-09-12 20:45:54,317] Trial 95 finished with value: 0.8383656844947168 and parameters: {'n_estimators': 70, 'max_dept
        h': 37, 'max_features': 0.8398878590212185, 'criterion': 'entropy'}. Best is trial 88 with value: 0.8593560434926659.
        [I 2022-09-12 20:45:55,094] Trial 96 finished with value: 0.8453117056532616 and parameters: {'n_estimators': 147, 'max_dept
        h': 25, 'max_features': 0.5457116291238912, 'criterion': 'gini'}. Best is trial 88 with value: 0.8593560434926659.
        [I 2022-09-12 20:45:55,736] Trial 97 finished with value: 0.8226634739917472 and parameters: {'n_estimators': 133, 'max_dept
        h': 6, 'max_features': 0.4952815803481252, 'criterion': 'entropy'}. Best is trial 88 with value: 0.8593560434926659.

        [I 2022-09-12 20:45:56,074] Trial 98 finished with value: 0.8344033357885351 and parameters: {'n_estimators': 62, 'max_dept
        h': 10, 'max_features': 0.371176569734877, 'criterion': 'entropy'}. Best is trial 88 with value: 0.8593560434926659.
        [I 2022-09-12 20:45:56,719] Trial 99 finished with value: 0.8513266121046007 and parameters: {'n_estimators': 126, 'max_dept
        h': 42, 'max_features': 0.42050213915806467, 'criterion': 'entropy'}. Best is trial 88 with value: 0.8593560434926659.

In [4]: trial = study.best_trial
        print('AUC: {}'.format(trial.value))

        AUC: 0.8593560434926659

In [5]: print("Best hyperparameters: {}".format(trial.params))

        Best hyperparameters: {'n_estimators': 133, 'max_depth': 43, 'max_features': 0.8915329694493298, 'criterion': 'entropy'}
```

Figure 4. Example of the result of RF hyperparameter tuning on Drupal dataset

Algorithm 1: Experimental Methodology

Input: D: A set of software vulnerability datasets, D = {PHPMyAdmin, Moodle, and Drupal}
R: A set of resampling techniques, R = {unbalanced, Smote, Adasyn, BLSmote, SmoteTL, SmoteENN}
L: A set of machine learning methods, L = {RF, AB, GNB, KNN, SVM}
RC:A set of resampling conditions, RC = {none, default, optimized}
HP:A set of arrays containing default hyperparameters and optimized hyperparameters, HP = {default [], optimized []}
Output: AUC and F1-Score performance of a combination of learners and resamplers
Begin:

```
1. for each data in D do
2.     for each l in L do
3.         for each r in R do
4.             for each rc in RC do
5.                 if(balancer == unbalanced && rc == none)
6.                     M1= Subalgorithm (HP = default [], l, data, r, rc)
7.                     M2= Subalgorithm (HP = optimized [], l, data, r, rc)
8.                 end if
9.                 if(balancer!=unbalanced && ((rc == default || rc == optimized))
10.                    M1= Subalgorithm (HP = default [], l, data, r, rc)
11.                    M2= Subalgorithm (HP = optimized [], l, data, r, rc)
12.                end if
13.            end for
14.        end for
15.    end for
16. end for
```

End

Subalgorithm: learners and resamplers performing with default or optimized hyperparameters

Input: HP = default [] or optimized [] , l: l ϵ L, data: data ϵ D, r: r ϵ R, rc: rc ϵ RC
M = 50, N = 3 // N fold cross-validation M times
Output: mean AUC metric value and F1-Score
Begin:

```
1. repeat (M times)
2.     randomized order from data
3.     generate k bins from data
4.     for each k in N do
5.         testData = bin (k)
6.         trainingData = data – testData
7.         new_data = r (trainingData)                    //data resampling
8.         predictor = l (new_data)
9.         apply predictor to testData
10.        return mean AUC metric value and F1-Score
11.    end for
12. end repeat
```

End

case), and on the validation part, 100 iterations are performed to achieve the best hyperparameters. Figure 4 describes the example of hyperparameter tuning of RF on the Drupal dataset for scenario 2.

- Further, the best hyperparameters are evaluated on the testing dataset using subalgorithm which calculates the AUC and $F1$-Score of each scenario. The final ratio of the training, validation, and testing part is 1.33:0.66:1.
- This paper focuses on optimizing a single objective function which means we are focusing on maximizing the AUC performance metric using the best hyperparameters configuration.
- Subalgorithm takes the dataset, hyperparameters (default or optimized), machine learning algorithm, resampling technique, and resampling condition as input and returns the AUC metric and $F1$-score as output to Algorithm 1.

4.2. Statistical tests

The current study has applied Wilcoxon signed-rank test [66, 67] for statistical analysis. The Wilcoxon signed-rank test is a non-parametric significance test, usually applied when the

readings are not normally distributed. It is used to test the differences in the performance (AUC) of six scenarios mentioned in Table 1. The p-value indicates a significant difference in the readings. Since six statistical tests are being performed so the p-value will now be changed to $0.05/6 = 0.0083$ as per the Bonferroni correction method [68, 69]. To conduct this test, the SPSS tool is used. The null hypothesis (Ho) describes that the performance values are equal and the alternate hypothesis (Ha) indicates that performance value differs.

To answer the research questions comparisons of the following cases are required:

1. (AdRn) & (AoRn): Default learner parameters with no resampling and optimized hyperparameters of learner with no resampling.
2. (AdRn) & (AdRd): Default learner parameters with no resampling and Default learner parameters with default resampler parameters.
3. (AdRd) & (AoRd): Default learner parameters with default resampling and optimized hyperparameters of learner with default resampler parameters.
4. (AdRd) & (AdRo): Default learner parameters with default resampling and default learner parameters with optimized hyperparameters of resampler.
5. (AdRn) & (AoRo): Default learner parameters with no resampling and optimized hyperparameters of both learner and resampler.
6. (AdRd) & (AoRo): Default learner parameters with default resampler parameters and optimized hyperparameters of both learner and resampler.

5. Results and analysis

This section explains the experimental results for the six scenarios of HPO. Further, the analysis of the results is performed to check whether each scenario is statistically significant. The results are based on the experimental procedure mentioned in Section 4. The goal of this paper is not to find the best resampler or best machine learner but to find the impact of dual HPO on SVP models.

5.1. AUC results

Tables 7–11 present the AUC values for each HPO scenario explained in Table 1. The blue-shaded cells indicate the highest AUC value per row. The yellow-shaded cell indicates the highest $F1$-Score among all scenarios. The bold + shaded cell indicates the highest AUC per dataset for each machine learner. It should be noted that our study works on the single objective function which has optimized the AUC metric and we are measuring AUC improvements. Although we have shown $F1$-Score in the results HPO and dual HPO will affect this metric mainly when used in the objective function which is a multi-objective problem that is out of the scope of this paper. The paper has still presented the effect of HPO and dual HPO on $F1$-Score when the single-objective function is optimized.

In Table 7, it has been observed that:

– In the case of RF, for Drupal, scenario 6 has achieved the highest AUC value of 0.8461 with SMOTE resampling technique.
– For Moodle dataset, BL SMOTE for scenario 6 has shown the highest AUC of 0.7783.
– For PHPMyAdmin, scenario 3 with default resampling results in a maximum AUC value of 0.7081.

– In the case of Drupal $F1$-Score has the highest value of 0.7088 in scenario 6 of Adasyn. For Moodle, $F1$-Score has the highest value of 0.0664 in scenario 6 of SMOTE. In PHPMyAdmin, the highest value of 0.2594 is attained in scenario 1.

In Table 8,

– For AB, it is observed that BL SMOTE for scenario 6 has performed the best for Drupal, with an AUC value of 0.8524.
– For Moodle, Adasyn for scenario 6 has resulted best, with an AUC value of 0.8402.
– In the case of PHPMyAdmin, scenario 5 has performed the best with an AUC value of 0.7351.
– $F1$-Score in the case of Drupal is highest with a value of 0.7566 for BL SMOTE scenario 6. Moodle has the highest $F1$-Score with a value of 0.0778 in SMOTE + TL scenario 6. PHPMyAdmin has the highest value of $F1$-Score 0.2881 in scenario 4 of SMOTE + TL.

Table 9 gives the AUC results for the GNB algorithm,

– SMOTE + TL with scenario 6 has provided the highest AUC value of 0.8777 for the Drupal dataset.
– For Moodle, SMOTE + ENN in scenario 6 has given the highest AUC value of 0.8909.
– In the case of PHPMyAdmin, Adasyn in scenario 4 has performed the best with the maximum AUC value of 0.7561.
– $F1$-Score in the case of Drupal is highest with a value of 0.5777 for SMOTE + ENN scenario 5. Moodle has the highest $F1$-Score with a value of 0.1679 in SMOTE scenario 6. PHPMyAdmin has the highest value of $F1$-Score 0.2648 in scenario 1.

Table 10 presents the AUC values for the KNN algorithm,

– For the Drupal dataset, SMOTE in scenario 6 has performed the best, with an AUC value of 0.8652.
– In the case of Moodle, BL SMOTE in scenario 6 has performed the best, with an AUC value of 0.7997.
– For PHPMyAdmin, the highest AUC value of 0.7001 is achieved by SMOTE in scenario 4.
– $F1$-Score in the case of Drupal is highest with a value of 0.6571 for SMOTE + ENN scenario 5. Moodle has the highest $F1$-Score with a value of 0.0743 in Adasyn scenario 5. PHPMyAdmin has the highest value of $F1$-Score 0.2683 in BL SMOTE scenario 6.

Table 11 gives the AUC performance value measure for the SVM algorithm,

– For the Drupal dataset, SMOTE + TL in scenario 4 has performed the best, with an AUC value of 0.8825.
– In the case of Moodle, BL SMOTE in scenario 6 has the best AUC value of 0.8492.
– For PHPMyAdmin, SMOTE + TL in scenario 4 has given the best AUC value of 0.7101.
– $F1$-Score in the case of Drupal is highest with a value of 0.6155 for BL SMOTE scenario 5. Moodle has the highest $F1$-Score with a value of 0.1133 in SMOTE scenario 4. PHPMyAdmin has the highest value of $F1$-Score 0.3872 in scenario 4.

It has been found that there lies a slight difference among the results of resampling techniques per HPO scenario; therefore we have not compared them in the study. Figures 5–7 describe the average AUC performance values of each HPO scenario calculated column-wise for Drupal, Moodle, and PHPMyAdmin respectively.

Table 7. Performance values for RF algorithm

Resampling techniques	Dataset	Scenarios											
		$A_d + R_n$		$A_o + R_n$		$A_d + R_d$		$A_o + R_d$		$A_d + R_o$		$A_o + R_o$	
		AUC	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score
NONE	PHP MyAdmin	0.6879	0.2594	0.6274	0.2327								
SMOTE						0.7025	0.2308	0.7044	0.1054	0.6741	0.09	0.6460	0.1269
ADASYN						0.7005	0.2389	0.6568	0.1658	0.6964	0.1421	0.6286	0.2208
BL SMOTE						0.7036	0.2323	0.6433	0.1299	0.6084	0.0791	0.6159	0.1365
SMOTE + TL						**0.7081**	0.2329	0.6302	0.1228	0.7029	0.0282	0.6677	0.0083
SMOTE + ENN						0.6515	0.2068	0.6965	0.0730	0.6164	0.0197	0.6863	0.1196
NONE	Moodle	0.6341	0.0	0.5644	0.0								
SMOTE						0.7264	0.0289	0.7057	0.0594	0.7169	0.0181	0.7374	0.0664
ADASYN						0.7358	0.0295	0.7019	0.0310	0.7042	0.0188	0.7559	0.0414
BL SMOTE						0.6641	0.0191	0.7033	0.0302	0.7149	0.0012	0.7316	0.0275
SMOTE + TL						0.7331	0.0286	0.7252	0.0032	0.7043	0.0057	**0.7783**	0.0244
SMOTE + ENN						0.7401	0.0281	0.6709	0.0222	0.7159	0.0574	0.7622	0.0263
NONE	Drupal	0.8131	0.5684	0.8145	0.4108								
SMOTE						0.8024	0.6087	0.7966	0.5527	0.8302	0.6458	0.8416	0.5742
ADASYN						0.7994	0.6082	0.8011	0.5427	0.8021	0.6141	0.8190	0.7088
BL SMOTE						0.7967	0.6022	0.8221	0.6388	0.8037	0.6328	0.8353	0.6085
SMOTE + TL						0.8019	0.6166	0.8157	0.6032	0.8277	0.5761	0.8236	0.5239
SMOTE + ENN						0.7961	0.6093	0.8196	0.5568	0.8131	0.5537	**0.8461**	0.4776

Table 8. Performance values for AB algorithm

Resampling techniques	Dataset	Scenarios											
		$A_d + R_n$		$A_o + R_n$		$A_d + R_d$		$A_o + R_d$		$A_d + R_o$		$A_o + R_o$	
		AUC	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score
NONE	PHPMyAdmin	0.6357	0.2799	0.6331	0.0484								
SMOTE						0.6095	0.1863	0.6895	0.1346	0.7189	0.1785	0.6882	0.0743
ADASYN						0.6136	0.1777	0.6825	0.2451	0.6769	0.2072	0.6206	0.1968
BL SMOTE						0.6113	0.1974	0.6311	0.241	0.7121	0.3295	0.6282	0.0287
SMOTE + TL						0.6107	0.1844	0.6584	0.2881	0.6523	0.1226	0.6145	0.0427
SMOTE + ENN						0.6081	0.1829	0.6621	0.2751	**0.7351**	0.1922	0.6419	0.1829
NONE	Moodle	0.7307	0.0009	0.8346	0.0								
SMOTE						0.7403	0.0381	0.7773	0.0261	0.8153	0.0391	0.7941	0.0434
ADASYN						0.7424	0.2469	0.8191	0.0391	0.7721	0.0353	**0.8402**	0.0402
BL SMOTE						0.7199	0.0514	0.8085	0.0209	0.8022	0.0318	0.8177	0.0407
SMOTE + TL						0.7434	0.0358	0.8239	0.0610	0.7746	0.0385	0.7611	0.0778
SMOTE + ENN						0.7521	0.0411	0.7829	0.0118	0.7483	0.0287	0.7737	0.0838
NONE	Drupal	0.7729	0.5177	0.8082	0.6460								
SMOTE						0.7655	0.5561	0.8355	0.5259	0.8065	0.6156	0.8175	0.6768
ADASYN						0.7643	0.5573	0.8239	0.6303	0.7609	0.4338	0.8511	0.7042
BL SMOTE						0.7588	0.5575	0.8083	0.5934	0.7962	0.5428	**0.8524**	0.7566
SMOTE + TL						0.7718	0.5739	0.7909	0.5875	0.7995	0.5291	0.8501	0.6203
SMOTE + ENN						0.7515	0.6043	0.7631	0.6561	0.7432	0.4527	0.8299	0.6324

Table 9. Performance values for GNB algorithm

Resampling techniques	Dataset	Scenarios											
		$A_d + R_n$		$A_o + R_n$		$A_d + R_d$		$A_o + R_d$		$A_d + R_o$		$A_o + R_o$	
		AUC	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score
NONE	PHPMyAdmin	0.6941	0.2648	0.7029	0.2206								
SMOTE						0.6851	0.2482	0.7265	0.0779	0.6221	0.2392	0.6975	0.1581
ADASYN						0.6842	0.2442	**0.7561**	0.2265	0.6582	0.2133	0.6005	0.1152
BL SMOTE						0.6525	0.2421	0.7007	0.0759	0.6268	0.2397	0.6671	0.0927
SMOTE + TL						0.6847	0.2547	0.7538	0.0762	0.6873	0.2869	0.6752	0.1522
SMOTE + ENN						0.7006	0.2452	0.7345	0.0545	0.6571	0.0605	0.6043	0.2357
NONE	Moodle	0.8151	0.0782	0.8453	0.0128								
SMOTE						0.8075	0.0606	0.8310	0.0721	0.7864	0.0291	0.8775	0.1679
ADASYN						0.8039	0.0602	0.8412	0.0356	0.7329	0.0182	0.8833	0.1114
BL SMOTE						0.8024	0.0883	0.8167	0.0410	0.7748	0.1454	0.8691	0.1271
SMOTE + TL						0.8094	0.0629	0.8313	0.0756	0.7668	0.0622	0.8553	0.0721
SMOTE + ENN						0.8040	0.0593	0.8404	0.0274	0.7962	0.0273	**0.8909**	0.1261
NONE	Drupal	0.7756	0.4924	0.8322	0.2132								
SMOTE						0.7781	0.4957	0.8456	0.5164	0.7926	0.5247	0.8735	0.4383
ADASYN						0.7723	0.5060	0.8596	0.4924	0.7533	0.4617	0.8482	0.5325
BL SMOTE						0.7731	0.5121	0.8236	0.5616	0.8226	0.5777	0.8551	0.4299
SMOTE + TL						0.7795	0.5034	0.8194	0.4556	0.7594	0.5009	**0.8777**	0.5396
SMOTE + ENN						0.7591	0.5626	0.8198	0.5231	0.7931	0.6211	0.8721	0.5521

Table 10. Performance values for KNN algorithm

Resampling techniques	Dataset	Scenarios											
		$A_d + R_n$		$A_o + R_n$		$A_d + R_d$		$A_o + R_d$		$A_d + R_o$		$A_o + R_o$	
		AUC	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score
NONE	PHPMyAdmin	0.6626	0.2099	0.6274	0.2327								
SMOTE						0.6676	0.2010	0.6746	0.157	0.6413	0.1956	0.6367	0.074
ADASYN						0.6504	0.2044	**0.7001**	0.1866	0.6194	0.1763	0.6194	0.2077
BL SMOTE						0.6508	0.1931	0.6674	0.084	0.6596	0.2455	0.6784	0.2683
SMOTE + TL						0.6598	0.2046	0.6657	0.2097	0.6764	0.2616	0.6661	0.1829
SMOTE + ENN						0.6676	0.1946	0.6561	0.2029	0.6561	0.1772	0.6432	0.1993
NONE	Moodle	0.5199	0.0	0.5644	0.0								
SMOTE						0.6917	0.0411	0.5683	0.023	0.7219	0.0372	0.7371	0.0566
ADASYN						0.6972	0.0389	0.6445	0.0025	0.7441	0.0743	0.7364	0.0431
BL SMOTE						0.6112	0.0689	0.5842	0.002	0.7133	0.0267	**0.7997**	0.0726
SMOTE + TL						0.7058	0.0411	0.5862	0.037	0.5328	0.0247	0.7458	0.043
SMOTE + ENN						0.6901	0.0382	0.5972	0.0132	0.7306	0.0646	0.7206	0.0563
NONE	Drupal	0.7679	0.5193	0.8145	0.4108								
SMOTE						0.7619	0.5938	0.7276	0.5709	0.8128	0.4338	0.8351	0.6154
ADASYN						0.7596	0.6035	0.7984	0.6258	0.8067	0.6055	0.8652	0.4401
BL SMOTE						0.7512	0.6034	0.7742	0.5148	0.7561	0.4055	0.8624	0.6039
SMOTE + TL						0.7582	0.5958	0.8053	0.5483	0.7729	0.7082	0.8124	0.5026
SMOTE + ENN						0.7501	0.6151	0.8055	0.6413	0.8143	0.6751	0.7626	0.5665

Table 11. Performance values for SVM algorithm

Resampling techniques	Dataset	Scenarios											
		$A_d + R_n$		$A_o + R_n$		$A_d + R_d$		$A_o + R_d$		$A_d + R_o$		$A_o + R_o$	
		AUC	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score	*AUC*	*F*1-Score
NONE	PHPMyAdmin	0.6354	0.2999	0.6629	0.2215								
SMOTE						0.6973	0.2444	0.6536	0.2738	0.6251	0.1862	0.6284	0.0
ADASYN						0.6905	0.2311	0.6572	0.2120	0.6951	0.0	0.6068	0.2292
BL SMOTE						0.6881	0.2465	0.6535	0.3872	0.6321	0.0	0.6108	0.0019
SMOTE + TL						0.6934	0.2474	**0.7101**	0.3218	0.6113	0.0	0.6625	0.1988
SMOTE + ENN						0.6267	0.1836	0.6311	0.1048	0.6923	0.3553	0.6049	0.1678
NONE	Moodle	0.4028	0.0	0.7506	0.0								
SMOTE						0.7303	0.0386	0.7923	0.1133	0.8415	0.0224	0.8236	0.1129
ADASYN						0.7379	0.0379	0.8265	0.0161	0.7885	0.0223	0.8202	0.0099
BL SMOTE						0.7721	0.0622	0.8215	0.0146	0.7628	0.0512	**0.8492**	0.0598
SMOTE + TL						0.7536	0.0379	0.8210	0.0538	0.8086	0.0176	0.7810	0.0311
SMOTE + ENN						0.7557	0.0365	0.7765	0.0091	0.7505	0.0652	0.8392	0.0801
NONE	Drupal	0.6955	0.4423	0.8001	0.1112								
SMOTE						0.8055	0.5681	0.8107	0.5831	0.8218	0.5721	0.8126	0.3564
ADASYN						0.7942	0.5805	0.8369	0.2854	0.8141	0.5803	0.8704	0.4923
BL SMOTE						0.7732	0.5934	0.8396	0.1655	0.8217	0.6155	0.8553	0.2401
SMOTE + TL						0.8001	0.5611	0.7744	0.4553	0.7976	0.4847	**0.8825**	0.3454
SMOTE + ENN						0.7931	0.5592	0.7859	0.5776	0.8052	0.5432	0.8518	0.4882

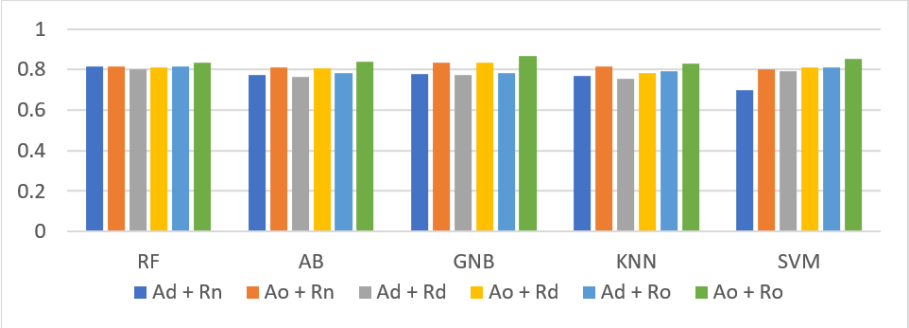

Figure 5. Average *AUC* performance values of each HPO scenario for the Drupal dataset

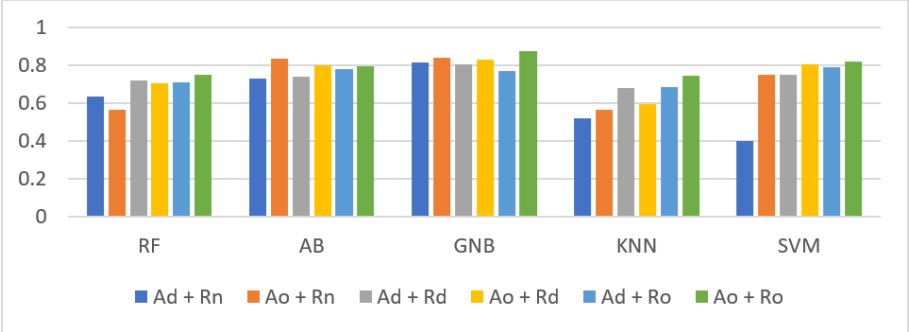

Figure 6. Average *AUC* performance values of each HPO scenario for the Moodle dataset

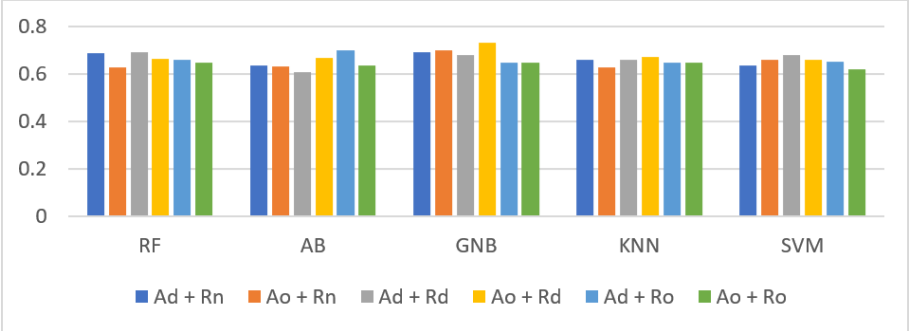

Figure 7. Average *AUC* performance values of each HPO scenario for the PHPMyAdmin dataset

5.2. Statistical results

Tables 12–14 show the results of the Wilcoxon signed-rank test applied to six scenarios for Drupal, Moodle, and PHPMyAdmin, respectively. The grey shaded area shows significant improvement, and "*" indicates the *AUC* value has decreased.

Table 12 shows the *p*-values, calculated after comparing the six cases mentioned above, for the Drupal dataset. The following observations depict the significant improvements of each machine learning algorithm.

– RF shows improvement in case 4 SMOTE, case 5 SMOTE, BL SMOTE, and SMOTE + ENN, and in case 6.

Table 12. Statistical comparison results among all five machine learners for Drupal dataset

		Resampling techniques					
Scenarios	Machine learning algorithm	NONE	SMOTE	ADASYN	BL SMOTE	SMOTE + TL	SMOTE + ENN
(A_dR_n) & (A_oR_n)	RF	0.211					
(A_dR_n) & (A_dR_d)			*	*	*	*	*
(A_dR_n) & (A_oR_d)			*	*	0.059	0.331	0.173
(A_dR_n) & (A_dR_o)			0.066	*	*	0.022	0.873
(A_dR_n) & (A_oR_o)			<0.001	0.511	0.002	0.039	<0.001
(A_dR_d) & (A_oR_o)			<0.001	0.005	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_n)	AB	<0.001					
(A_dR_n) & (A_dR_d)			*	*	*	*	*
(A_dR_n) & (A_oR_d)			<0.001	<0.001	<0.001	0.057	*
(A_dR_n) & (A_dR_o)			<0.001	*	0.004	<0.001	*
(A_dR_n) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_d) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_n)	GNB	<0.001					
(A_dR_n) & (A_dR_d)			0.930	*	*	0.360	*
(A_dR_n) & (A_oR_d)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_dR_o)			0.023	*	<0.001	*	0.018
(A_dR_n) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_d) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_n)	KNN	<0.001					
(A_dR_n) & (A_dR_d)			*	*	*	*	*
(A_dR_n) & (A_oR_d)			<0.001	0.002	0.199	<0.001	<0.001
(A_dR_n) & (A_dR_o)			0.023	<0.001	*	0.705	<0.001
(A_dR_n) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	0.520
(A_dR_d) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.036
(A_dR_n) & (A_oR_n)	SVM	<0.001					
(A_dR_n) & (A_dR_d)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_d)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_dR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_d) & (A_oR_o)			0.037	<0.001	<0.001	<0.001	<0.001

- For AB, case 1, case 3 SMOTE, Adasyn, and BL SMOTE, case 4 SMOTE, BL SMOTE, SMOTE + TL, case 5, and 6 shows improvement.
- GNB shows improvement for scenarios 1, 3, 5, and 6 and BL SMOTE in case 4.
- KNN has complete improvements for case 1, and case 3 except for BL SMOTE, case 4 Adasyn and SMOTE + ENN, case 5 and 6 except SMOTE + ENN.
- SVM shows improvement in all the cases except for case 6 of SMOTE.

Table 13 shows the p-values for the Moodle dataset. The observations are as follows:

- RF has shown improvement in cases 2, 4, 5, case 3 except SMOTE + ENN, and case 6 with BL SMOTE and SMOTE + TL.
- AB shows improvement for case 1, case 5, case 2 SMOTE + ENN, case 3 except SMOTE + ENN, case 4 except SMOTE + ENN, and case 6 except SMOTE + TL and SMOTE + ENN.
- GNB shows improvement for case 1, case 5, case 6, for case 3 SMOTE + ENN.
- KNN shows improvement in case 1, case 2, case 5, case 3 except Adasyn and SMOTE + ENN, case 4 except SMOTE + TL, and case 6 except Adasyn and SMOTE + ENN.
- SVM shows improvement in all cases except SMOTE + ENN of the case 6.

Table 13. Statistical comparison results among all five machine learners for Moodle dataset

Scenarios	Machine learning algorithm	Resampling techniques					
		NONE	SMOTE	ADASYN	BL SMOTE	SMOTE + TL	SMOTE + ENN
(A_dR_n) & (A_oR_n)	RF	*					
(A_dR_n) & (A_dR_d)			<0.001	<0.001	0.003	<0.001	<0.001
(A_dR_n) & (A_oR_d)			<0.001	<0.001	<0.001	0.002	0.047
(A_dR_n) & (A_dR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_d) & (A_oR_o)			0.071	0.036	<0.001	0.004	0.709
(A_dR_n) & (A_oR_n)	AB	<0.001					
(A_dR_n) & (A_dR_d)			0.183	0.172	*	0.159	0.004
(A_dR_n) & (A_oR_d)			0.007	<0.001	<0.001	<0.001	0.028
(A_dR_n) & (A_dR_o)			0.003	0.001	<0.001	0.002	0.023
(A_dR_n) & (A_oR_o)			<0.001	<0.001	0.001	<0.001	<0.001
(A_dR_d) & (A_oR_o)			<0.001	<0.001	<0.001	0.025	0.037
(A_dR_n) & (A_oR_n)	GNB	<0.001					
(A_dR_n) & (A_dR_d)			*	*	*	*	*
(A_dR_n) & (A_oR_d)			0.015	0.229	0.015	0.150	<0.001
(A_dR_n) & (A_dR_o)			*	*	*	*	*
(A_dR_n) & (A_oR_o)			<0.001	<0.001	<0.001	0.004	<0.001
(A_dR_d) & (A_oR_o)			<0.001	<0.001	<0.001	0.002	<0.001
(A_dR_n) & (A_oR_n)	KNN	<0.001					
(A_dR_n) & (A_dR_d)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_d)			<0.001	0.031	<0.001	<0.001	0.0153
(A_dR_n) & (A_dR_o)			0.004	<0.001	<0.001	0.785	<0.001
(A_dR_n) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_d) & (A_oR_o)			<0.001	0.022	<0.001	0.004	<0.013
(A_dR_n) & (A_oR_n)	SVM	<0.001					
(A_dR_n) & (A_dR_d)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_d)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_dR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_n) & (A_oR_o)			<0.001	<0.001	<0.001	<0.001	<0.001
(A_dR_d) & (A_oR_o)			0.022	<0.001	0.984	0.069	<0.001

Table 14 presents the p-values for the PHPMyAdmin dataset. The statistical results are:

- RF has shown no significant improvement.
- AB has shown significant improvement for case 3 Adasyn, case 4 except SMOTE + TL, SMOTE for case 5 and case 6.
- GNB has shown significant improvement for case 3 Adasyn
- KNN has shown significant improvement for case 1 and case 2 SMOTE + ENN.
- SVM shows a significant improvement in case 2 except SMOTE + ENN, case 3 SMOTE + TL, and case 4 Adasyn.

5.3. Illustration of research questions

Results reported in Tables 7–14 contribute to the answers to research questions.
RQ 1: How much is dual HPO effective in improving the performance of SVP models? The statistical comparisons of AdRn & AoRo (case 5 of Section 4.2) show the effectiveness of dual HPO on SVP models. There exist a total of 75 instances out of which 48 instances

Table 14. Statistical comparison results among all five machine learners for PHPMyAdmin dataset

Scenarios	Machine learning algorithm	Resampling techniques					
		NONE	SMOTE	ADASYN	BL SMOTE	SMOTE + TL	SMOTE + ENN
(A_dR_n) & (A_oR_n)	RF	*					
(A_dR_n) & (A_dR_d)			0.100	0.159	0.089	0.041	*
(A_dR_n) & (A_oR_d)			0.638	*	*	*	0.985
(A_dR_n) & (A_dR_o)			*	0.558	*	0.107	*
(A_dR_n) & (A_oR_o)			*	*	*	*	*
(A_dR_d) & (A_oR_o)			*	*	*	*	0.009
(A_dR_n) & (A_oR_n)	AB	*					
(A_dR_n) & (A_dR_d)			*	*	*	*	*
(A_dR_n) & (A_oR_d)			0.020	0.005	*	0.181	0.033
(A_dR_n) & (A_dR_o)			0.004	0.006	<0.001	0.517	<0.001
(A_dR_n) & (A_oR_o)			<0.001	*	*	0.237	0.742
(A_dR_d) & (A_oR_o)			<0.001	0.812	0.325	0.683	0.063
(A_dR_n) & (A_oR_n)	GNB	0.361					
(A_dR_n) & (A_dR_d)			*	*	*	*	0.541
(A_dR_n) & (A_oR_d)			0.059	<0.001	0.361	0.203	0.031
(A_dR_n) & (A_dR_o)			*	*	*	*	*
(A_dR_n) & (A_oR_o)			*	*	*	*	*
(A_dR_d) & (A_oR_o)			0.042	*	0.095	*	*
(A_dR_n) & (A_oR_n)	KNN	0.007					
(A_dR_n) & (A_dR_d)			0.512	*	*	*	<0.001
(A_dR_n) & (A_oR_d)			0.381	0.041	0.713	0.835	*
(A_dR_n) & (A_dR_o)			*	*	*	0.406	*
(A_dR_n) & (A_oR_o)			*	*	0.173	0.492	*
(A_dR_d) & (A_oR_o)			*	*	0.056	0.443	*
(A_dR_n) & (A_oR_n)	SVM	0.031					
(A_dR_n) & (A_dR_d)			<0.001	<0.001	<0.001	<0.001	0.567
(A_dR_n) & (A_oR_d)			0.195	0.080	0.207	<0.001	*
(A_dR_n) & (A_dR_o)			0.661	0.005	0.695	0.447	*
(A_dR_n) & (A_oR_o)			*	0.354	*	*	0.422
(A_dR_d) & (A_oR_o)			*	*	*	0.272	*

significantly improved the *AUC* performance value. Therefore, dual HPO is 64% effective in enhancing the productivity of SVP models.

RQ 2: Is dual HPO better than other HPO scenarios?

Wilcoxon signed-rank test results mentioned in Tables 11–13 show cases of significant improvements. To check whether dual HPO is better than other HPO scenarios, we compare the number of instances of significant improvement of dual HPO with others. AdRn & AdRd (case 2) shows 26 significant improvements out of 75 instances, AdRn & AoRd (case 3 of Section 4.2) shows 38 significant improvements, and AdRn & AdRo (case 4 of Section 4.2) results in 34 instances of significant improvement. Dual HPO has 48 instances of improvement; therefore it is better than other HPO scenarios.

RQ 3: How has the degree of class overlapping affected the HPO?

There are 100 instances (consider cases 3–6 of Section 4.2) of HPO comparisons per dataset. Drupal has shown improvement in 61 instances, Moodle in 75 instances, and PHPMyAdmin in 10 instances. HPO depends on data complexity measures mentioned in Section 3.6. If the overlap among the classes is low, then HPO performs efficiently. As per the *F*1 values for Drupal, Moodle, and PHPMyAdmin in Table 4, PHPMyAdmin has the highest overlap, and Moodle has the lowest. Therefore, for PHPMyAdmin, HPO has not improved

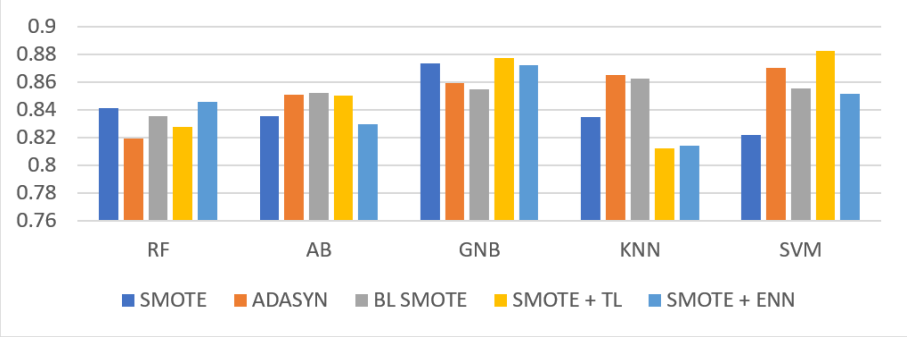

Figure 8. Comparison of resampling techniques in Drupal dataset

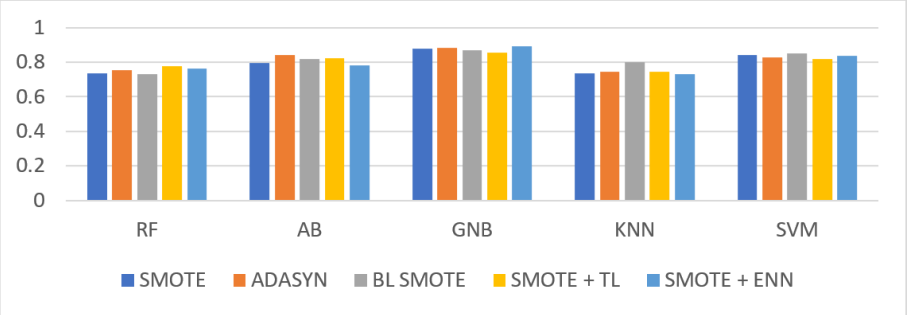

Figure 9. Comparison of resampling techniques in Moodle dataset

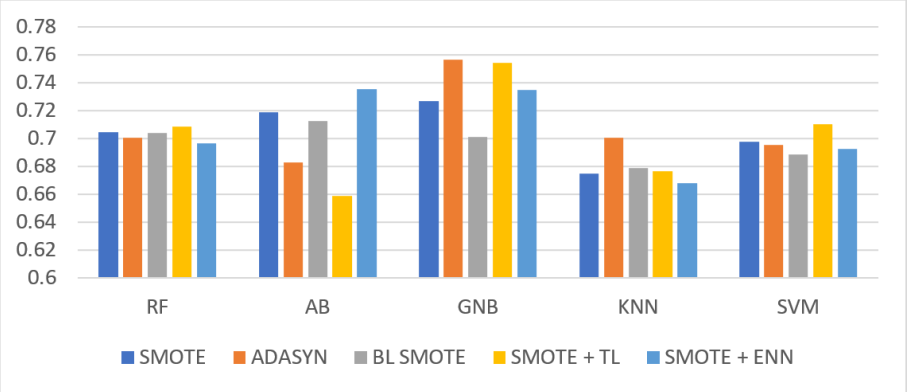

Figure 10. Comparison of resampling techniques in PHPMyAdmin dataset

the performance and, for Moodle, which is highly imbalanced, HPO has worked well. For Drupal, HPO has given mixed results.

RQ 4: Which resampling technique has performed the best?

Figures 8–10 show the highest *AUC* value among four scenarios (AdRd, AoRd, AdRo, and AoRo) resampling techniques for each machine learning algorithm.

It has been observed that there is not one technique that performs well for all algorithms and all datasets. Different machine learning algorithms and datasets have distinct highest-performing resamplers. For instance, in the case of the Drupal dataset, SMOTE + ENN has performed highest in RF, BL SMOTE in AB, SMOTE + TL in GNB and SVM, and finally Adasyn in KNN. For Moodle, SMOTE + TL in RF, Adasyn in AB, SMOTE + ENN in GNB, BL SMOTE in KNN and SVM. For PHPMyAdmin, SMOTE + TL in RF and SVM, SMOTE + ENN in AB, Adasyn in GNB and KNN.

6. Discussions

This section discusses the findings of our study. Imbalanced datasets and lack of hyperparameter tuning may impact the productivity of SVP models. HPO and resampling techniques are known to improve the performance of the prediction models. The current study aims to analyze the effect of dual HPO in software vulnerability prediction. By dual HPO, it means that hyperparameters of both machine learning algorithms and resampling techniques are optimized. We have not only checked the effect of dual HPO but additionally, analyzed whether dual HPO is better than single HPO scenarios where only one of the two factors (machine learning and resampling) is optimized. Furthermore, the inclusion of data complexity measures helps in finding why HPO is not producing results for the classes with high overlap.

Our study will help researchers in improving the performance of their work on prediction models and provide open research for multi-objective optimization, text-mining-based SVP models, cost and time complexity measures, and deep learning approaches.

7. Threats to validity

The current paper deals with the following threats to validity:

- Construct Validity: The datasets used in this paper are PHP web application projects. This paper has used a metrics-based dataset to carry out the experiments. If a different type of feature such as text-mining is considered, then the results may vary.
- Internal validity: The selection of machine learning algorithms is based on past research performed. We have confined our experiments to five machine-learning algorithms. Also, the resampling techniques used are either oversampling or a combination of oversampling and under-sampling. Under-sampling techniques are not used because information loss may occur. The hyperparameter search spaces are taken from past research, and results may vary for considering different search spaces. The paper takes care of optimizing the single-objective function which works on increasing the value of the particular metric AUC in our case, a multi-objective function is for the future scope.
- External validity: Generalization of the work in an empirical study is always limited, and it is difficult to conclude. The generalizability of our results is out of the scope of the current study. The performance varies for different programming languages as the features and granularity levels may be different. The study aims to see the impact of dual HPO on various machine learners and to check whether it improves the efficiency of metrics-based SVP models. It uses the Wilcoxon signed-rank test with Bonferroni correction for performing significant comparisons. This test is based on the data sample in hand.
- Conclusion validity: AUC is the performance metric used in the current study. There exist other parameters for the evaluation of imbalanced datasets such as Geometric mean, Matthews's correlation coefficient (MCC), etc. HPO comes with time and cost overhead. We have kept these overheads out of the scope of this paper.

8. Conclusions and future scope

The current study gives an insight into HPO in the machine learning area. Previous studies have used HPO in bug prediction, defect prediction, and even vulnerability prediction

models. In this work, we have analyzed the effectiveness of dual HPO on the capability of machine learners in software vulnerability prediction. Further, whether dual HPO performs better than other HPO scenarios is examined. In addition, it is further studied why HPO performance is degraded for some datasets, i.e., has class overlapping affected the ability of HPO in improving the efficacy of machine learners. The study proposed the experimental methodology based on the python framework "Optuna" that evaluates the six HPO scenarios depicted in Table 1. The paper uses five machine learning algorithms and five resampling techniques for three open-source software vulnerability datasets Drupal, Moodle, and PHPMyAdmin. The best hyperparameters are found for learners and resamplers to optimize the SVP model. In addition to this, the Wilcoxon signed-rank test with Bonferroni correction is applied for statistical comparison to know which HPO scenario has performed significantly. The experimental results are concluded as:

- The results state that dual HPO has shown 64% effectiveness in amplifying the efficiency of SVP models.
- Dual HPO shows 64% effectiveness whereas HPO when applied on machine learners shows 51% and HPO when applied on resamplers obtains 45.33% effectiveness. Therefore, it can be concluded that Dual HPO performs better than other HPO scenarios. We have not compared with the scenarios that do not involve resampling. Although AoRn has shown 9 significant improvements out of 15, we cannot compare them as they may be biased since applied to unbalanced datasets.
- The efficiency of HPO is affected by class overlapping. One of the data complexity measures; is the maximum Fisher's discriminant ratio ($F1$) which calculates the overlap among the classes. The datasets with high overlapping classes result in the poor performance of HPO. PHPMyAdmin is highly overlapped resulting in 10 improvements out of 100; therefore HPO has not performed well for it. Moodle being low overlapped gives the maximum significant results for HPO 75 out of 100.
- Resampling Techniques are analyzed and it is found that they perform differently on distinct datasets and with different learners. Hence, we cannot find the best resampler that fits all the datasets and machine learners.

Future work emphasizes the use of HPO scenarios for text-based datasets. We can consider time and cost complexity measures with more data complexity measures in the future. The impact of HPO on deep learning approaches can be studied. Other programming languages can be explored. Furthermore, the multi-objective problem can be explored.

References

[1] S.M. Ghaffarian and H.R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Computing Surveys (CSUR)*, Vol. 50, No. 4, 2017, pp. 1–36.

[2] W.R.J. Freitez, A. Mammar, and A.R. Cavalli, "Software vulnerabilities, prevention and detection methods: A review," in *Security in Model Driven Architecture (SEC-MDA)*, A. Bagnato, Ed., 2009, pp. 6–13.

[3] A. Kaya, A.S. Keceli, C. Catal, and B. Tekinerdogan, "The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models," *Journal of Software: Evolution and Process*, Vol. 31, No. 9, 2019, p. e2164.

[4] J. Morgenthaler and J. Penix, "Software development tools using static analysis to find bugs," *Development*, 2008.

[5] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," *IEEE Security and Privacy*, Vol. 3, No. 1, 2005, pp. 84–87.

[6] P. Godefroid, "Random testing for security: Blackbox vs. whitebox fuzzing," in *Proceedings of the 2nd International Workshop on Random Testing: Co-Located With the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, 2007, pp. 1–1.
[7] D. Evans and D. Larochelle, "Improving security using extensible lightweight static analysis," *IEEE Software*, Vol. 19, No. 1, 2002, pp. 42–51.
[8] M. Fagan, "Design and code inspections to reduce errors in program development," in *Software Pioneers*, M. Broy and E. Denert, Eds. Springer, 2002, pp. 575–607.
[9] H. Shahriar and M. Zulkernine, "Mitigating program security vulnerabilities: Approaches and challenges," *ACM Computing Surveys (CSUR)*, Vol. 44, No. 3, 2012, pp. 1–46.
[10] M. Jimenez, R. Rwemalika, M. Papadakis, F. Sarro, Y. Le Traon et al., "The importance of accounting for real-world labelling when predicting software vulnerabilities," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 695–705.
[11] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Software Engineering*, Vol. 18, No. 1, 2013, pp. 25–59.
[12] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *Third International Conference on Software Testing, Verification and Validation*. IEEE, 2010, pp. 421–428.
[13] H. Alves, B. Fonseca, and N. Antunes, "Experimenting machine learning techniques to predict vulnerabilities," in *Seventh Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 2016, pp. 151–156.
[14] W. Almutairi and R. Janicki, "On relationships between imbalance and overlapping of datasets," in *Proceedings of 35th International Conference on Computers and Their Applications*, 2020, pp. 141–150.
[15] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, Vol. 62, No. 2, 2013, pp. 434–443.
[16] T. Sasada, Z. Liu, T. Baba, K. Hatano, and Y. Kimura, "A resampling method for imbalanced datasets considering noise and overlap," *Procedia Computer Science*, Vol. 176, 2020, pp. 420–429.
[17] K. Borowska and J. Stepaniuk, "Imbalanced data classification: A novel re-sampling approach combining versatile improved SMOTE and rough sets," in *IFIP International Conference on Computer Information Systems and Industrial Management*. Springer, 2016, pp. 31–42.
[18] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue et al., "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems With Applications*, Vol. 73, 2017, pp. 220–239.
[19] G. Douzas, F. Bacao, and F. Last, "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE," *Information Sciences*, Vol. 465, 2018, pp. 1–20.
[20] C. Seiffert, T.M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, Vol. 40, No. 1, 2009, pp. 185–197.
[21] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 321–332.
[22] J.N. Van Rijn and F. Hutter, "Hyperparameter importance across datasets," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 2367–2376.
[23] H.J. Weerts, A.C. Mueller, and J. Vanschoren, "Importance of tuning hyperparameters of machine learning algorithms," *arXiv preprint arXiv:2007.07588*, 2020.
[24] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, Vol. 415, 2020, pp. 295–316. [Online]. https://www.sciencedirect.com/science/article/pii/S0925231220311693
[25] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," *arXiv preprint arXiv:1905.06872*, 2019.

[26] J. Kong, W. Kowalczyk, D.A. Nguyen, T. Bäck, and S. Menzel, "Hyperparameter optimisation for improving classification under class imbalance," in *Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2019, pp. 3072–3078.
[27] R. Shu, T. Xia, J. Chen, L. Williams, and T. Menzies, "How to better distinguish security bug reports (using dual hyperparameter optimization)," *Empirical Software Engineering*, Vol. 26, No. 3, 2021, pp. 1–37.
[28] A. Agrawal, W. Fu, D. Chen, X. Shen, and T. Menzies, "How to "dodge" complex software analytics," *IEEE Transactions on Software Engineering*, Vol. 47, No. 10, 2019, pp. 2182–2194.
[29] A. Agrawal, X. Yang, R. Agrawal, R. Yedida, X. Shen et al., "Simpler hyperparameter optimization for software analytics: Why, how, when," *IEEE Transactions on Software Engineering*, Vol. 48, No. 8, 2021, pp. 2939–2954.
[30] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 23–33.
[31] J. Stuckman, J. Walden, and R. Scandariato, "The effect of dimensionality reduction on software vulnerability prediction models," *IEEE Transactions on Reliability*, Vol. 66, No. 1, 2016, pp. 17–37.
[32] M. Claesen and B. De Moor, "Hyperparameter search in machine learning," 2015. [Online]. https://arxiv.org/abs/1502.02127
[33] P.K. Kudjo, S.B. Aformaley, S. Mensah, and J. Chen, "The significant effect of parameter tuning on software vulnerability prediction models," in *19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2019, pp. 526–527.
[34] E. Sara, C. Laila, and I. Ali, "The impact of SMOTE and grid search on maintainability prediction models," in *16th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2019, pp. 1–8.
[35] H. Osman, M. Ghafari, and O. Nierstrasz, "Hyperparameter optimization to improve bug prediction accuracy," in *Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*. IEEE, 2017, pp. 33–38.
[36] V.H. Barella, L.P. Garcia, M.P. de Souto, A.C. Lorena, and A. de Carvalho, "Data complexity measures for imbalanced classification tasks," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.
[37] T.K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 3, 2002, pp. 289–300.
[38] J.M. Sotoca, J. Sánchez, and R.A. Mollineda, "A review of data complexity measures and their applicability to pattern classification problems," in *Actas del III Taller Nacional de Mineria de Datos y Aprendizaje*, R. Ruiz, J.C. Riquelme, and J.S. Aguilar-Ruiz, Eds., 2005, pp. 77–83.
[39] A.C. Lorena, L.P. Garcia, J. Lehmann, M.C. Souto, and T.K. Ho, "How complex is your classification problem? A survey on measuring classification complexity," *ACM Computing Surveys (CSUR)*, Vol. 52, No. 5, 2019, pp. 1–34.
[40] Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun et al., "Combining software metrics and text features for vulnerable file prediction," in *20th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2015, pp. 40–49.
[41] I. Abunadi and M. Alenezi, "An empirical investigation of security vulnerabilities within web applications," *Journal of Universal Computer Science*, Vol. 22, 2016, pp. 537–551.
[42] M.N. Khalid, H. Farooq, M. Iqbal, M.T. Alam, and K. Rasheed, "Predicting web vulnerabilities in web applications based on machine learning," in *Intelligent Technologies and Applications*. Singapore: Springer Singapore, 2019, pp. 473–484.
[43] C. Catal, A. Akbulut, E. Ekenoglu, and M. Alemdaroglu, "Development of a software vulnerability prediction web service based on artificial neural networks," in *Trends and Applications in Knowledge Discovery and Data Mining*. Cham: Springer International Publishing, 2017, pp. 59–67.
[44] D. Bassi and H. Singh, "Optimizing hyperparameters for improvement in software vulnerability prediction models," in *Advances in Distributed Computing and Machine Learning*, R.R. Rout,

S.K. Ghosh, P.K. Jana, A.K. Tripathy, J.P. Sahoo et al., Eds., Singapore: Springer Nature, 2022, pp. 533–544.

[45] Z. Jin, J. Shang, Q. Zhu, C. Ling, W. Xie et al., "RFRSF: Employee turnover prediction based on random forests and survival analysis," in *Web Information Systems Engineering – WISE 2020*, Z. Huang, W. Beek, H. Wang, R. Zhou, and Y. Zhang, Eds., Cham: Springer International Publishing, 2020, pp. 503–515.

[46] R.E. Schapire, *The Boosting Approach to Machine Learning: An Overview*. New York, NY: Springer New York, 2003, pp. 149–171.

[47] R. Meir and G. Rätsch, *An Introduction to Boosting and Leveraging*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 118–183.

[48] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, Vol. 37, 1999, pp. 297–336.

[49] H. Zhang, "The optimality of Naive Bayes," in *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*, V. Barr and Z. Markov, Eds., AAAI Press, 2004.

[50] M. Martinez-Arroyo and L.E. Sucar, "Learning an optimal naive bayes classifier," in *18th International Conference on Pattern Recognition (ICPR'06)*, Vol. 3. IEEE, 2006, pp. 1236–1239.

[51] J.D.M. Rennie, L. Shih, J. Teevan, and D.R. Karger, "Tackling the poor assumptions of naive bayes text classifiers," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, 2003, pp. 616–623.

[52] C.C. Chang and C.J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, Vol. 2, No. 3, 2011, pp. 1–27.

[53] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, and V.N. Vapnik, "Support vector regression machines," in *Advances in Neural Information Processing Systems 9 (NIPS)*, M. Mozer, M. Jordan, and T. Petsche, Eds., 1996.

[54] X. Wang, J. Yang, X. Teng, and N. Peng, "Fuzzy-rough set based nearest neighbor clustering classification algorithm," in *Fuzzy Systems and Knowledge Discovery*, L. Wang and Y. Jin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 370–373.

[55] W. Zuo, D. Zhang, and K. Wang, "On kernel difference-weighted k-nearest neighbor classification," *Pattern Analysis and Applications*, Vol. 11, No. 3, 2008, pp. 247–257.

[56] M. Santos, J. Soares, P. Henriques Abreu, H. Araujo, and J. Santos, "Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches," *IEEE Computational Intelligence Magazine*, Vol. 13, 2018, pp. 59–76.

[57] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, Vol. 16, 2002, pp. 321–357.

[58] H. He, Y. Bai, E.A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1322–1328.

[59] H. Han, W.Y. Wang, and B.H. Mao, "Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning," in *Advances in Intelligent Computing*, D.S. Huang, X.P. Zhang, and G.B. Huang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 878–887.

[60] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Vol. abs/1907.10902, 2019. [Online]. http://arxiv.org/abs/1907.10902

[61] V. López, A. Fernández, J.G. Moreno-Torres, and F. Herrera, "Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. Open problems on intrinsic data characteristics," *Expert Systems with Applications*, Vol. 39, No. 7, 2012, pp. 6585–6608.

[62] J. Huang and C.X. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 3, 2005, pp. 299–310.

[63] K. Sultana, B. Williams, and A. Bosu, "A comparison of nano-patterns vs. software metrics in vulnerability prediction," in *Proceedings – 25th Asia-Pacific Software Engineering Conference, APSEC 2018*. IEEE Computer Society, 2018, pp. 355–364.

[64] A.K. Tanwani and M. Farooq, "Classification potential vs. classification accuracy: A comprehensive study of evolutionary algorithms with biomedical datasets," in *Learning Classifier Systems*, J. Bacardit, W. Browne, J. Drugowitsch, E. Bernadó-Mansilla, and M.V. Butz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 127–144.
[65] S.S. Rathore and S. Kumar, "An empirical study of ensemble techniques for software fault prediction," *Applied Intelligence*, Vol. 51, No. 6, 2021, pp. 3615–3644.
[66] D. Tomar and S. Agarwal, "Prediction of defective software modules using class imbalance learning," *Applied Computational Intelligence and Soft Computing*, Vol. 2016, 2016, pp. 1–12.
[67] A. Kaur and K. Kaur, "Statistical comparison of modelling methods for software maintainability prediction," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 23, 2013.
[68] E.W. Weisstein, *Bonferroni Correction*, 2004. [Online]. https://mathworld.wolfram.com/
[69] P. Sedgwick, "Multiple significance tests: The Bonferroni correction," *BMJ (online)*, Vol. 344, 2012, pp. e509–e509.