


Article

Interpretable Software Defect Prediction from Project Effort and Static Code Metrics

Susmita Haldar ^{1,*}, †, ‡  and Luiz Fernando Capretz ^{2,*}, † ¹ School of Information Technology, Fanshawe College, London, ON N5Y 5R6, Canada² Department of Electrical and Computer Engineering, Western University, London, ON N6A 3K7, Canada

* Correspondence: shaldar@fanshawec.ca (S.H.); lcapretz@uwo.ca (L.F.C.)

† Current address: Department of Electrical and Computer Engineering, Western University, London, ON N6A 3K7, Canada.

‡ These authors contributed equally to this work.

Abstract: Software defect prediction models enable test managers to predict defect-prone modules and assist with delivering quality products. A test manager would be willing to identify the attributes that can influence defect prediction and should be able to trust the model outcomes. The objective of this research is to create software defect prediction models with a focus on interpretability. Additionally, it aims to investigate the impact of size, complexity, and other source code metrics on the prediction of software defects. This research also assesses the reliability of cross-project defect prediction. Well-known machine learning techniques, such as support vector machines, k-nearest neighbors, random forest classifiers, and artificial neural networks, were applied to publicly available PROMISE datasets. The interpretability of this approach was demonstrated by SHapley Additive exPlanations (SHAP) and local interpretable model-agnostic explanations (LIME) techniques. The developed interpretable software defect prediction models showed reliability on independent and cross-project data. Finally, the results demonstrate that static code metrics can contribute to the defect prediction models, and the inclusion of explainability assists in establishing trust in the developed models.

Keywords: defect prediction; explainable machine learning; software quality; interpretability; cross-project defect prediction

**Citation:** Haldar, S.; Capretz, L.F.Interpretable Software Defect Prediction from Project Effort and Static Code Metrics. *Computers* **2024**, *13*, 52. <https://doi.org/10.3390/computers13020052>

Academic Editors: Osvaldo Gervasi and Damiano Perri

Received: 13 January 2024

Revised: 10 February 2024

Accepted: 12 February 2024

Published: 16 February 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Software testing demands a significant allocation of time, budget, and resources, which can become expensive when the source code contains multiple faults, necessitating additional retesting efforts. Additionally, test managers face the challenge of determining which modules to test as allocating the same level of effort to all modules may not be practical [1]. Identifying defective modules becomes a significant task during test planning. This has led to the development of automated software defect prediction (SDP) processes that utilize various metrics derived from historical information. Consequently, defect prediction using machine learning techniques has emerged as a popular research area, aiming to automate the manual efforts involved in identifying different types of defects in software applications [2,3].

It is a daunting task to identify which attributes are good predictors for defect prediction, and many different research studies have been conducted in identifying the metrics to be used in the SDP models along with an efficient feature selection process [4,5]. Also, the performance of defect classification models can be hindered by features that are redundant, correlated, or irrelevant. To overcome this problem, researchers have often utilized feature selection techniques to improve the SDP model's performance through either transforming the features or selecting a subset of them, aiming to enhance the classification models' effectiveness [6].

Test managers may face difficulty in placing trust in the results generated by predictive models due to their limited understanding of the internal workings of the systems. Before allocating testing resources to modules identified as error-prone [7], test managers need to comprehend the rationale behind the predictions. The development of an interpretable defect prediction approach, coupled with the ability to understand the static code metrics that are contributing to the identification of defect-prone modules, will allow test managers to provide sufficient data for training the model. By choosing relevant static code metrics obtained from previous projects of a similar nature, test managers can effectively facilitate testing arrangements for new applications.

Considering the aforementioned points, our research aims to contribute to the research community by addressing the following questions: RQ1 Can software defect prediction models generated from different projects with varied sample sizes yield consistent results? RQ2 Does it significantly impact the results if highly correlated independent features are removed from the set of independent variables when developing software defect prediction models? RQ3 Can we rely on prediction models developed using cross-project metrics compared to models built from individual projects? RQ4 Can we consistently interpret software defect prediction models after applying SMOTE techniques to balance unbalanced data?

The main contribution of this paper is the development of software defect prediction models that emphasize interpretability and the impact of various source code metrics, including size and complexity. One of the objectives was to verify how the defect prediction model performs when training with individual projects compared to training with cross-project information. Another major contribution of this work was to analyze the performance and interpretability of the SDP models when highly correlated independent features are removed compared to retaining highly correlated features. This work performed a comparison of the performance of the generation of SDP models with the original imbalanced dataset, and with balanced data after applying the oversampling of minority class technique. Finally, the best-performing model was interpreted using LIME and SHAP techniques.

This paper is organized into several sections. A literature survey on defect prediction using machine learning techniques is presented in Section 2. This is followed by the methodology used in this paper in Section 3. The developed SDP models and results are presented in Section 4. Section 5 summarizes the results analysis, presents a discussion of these, and considers threats to the validity of our work. Finally, the conclusions and future work are described in Section 6.

2. Literature Review

A considerable amount of literature has been published on the software defect prediction problem over the last few decades. When the NASA data defect repository became publicly available, this dataset became a popular reference for many researchers [8–10]. To build these SDP models, researchers have utilized various regression [11] and classification techniques, such as support vector machine [12], the k-nearest neighbor algorithm [13], random forest algorithms [14], deep learning methods incorporating artificial neural networks [15], recurrent neural networks [16] and convolutional neural networks [17], ensemble methods [18], and the transfer learning framework [19], etc. Like these studies, our SDP models will be built using existing machine learning algorithms to support our research questions.

Several studies have devoted time to cleaning the data due to low and inconsistent data quality in SDP research [2], and one of the criteria for cleaning data was handling outliers. One of the common techniques applied was removing outliers based on the interquartile range (IQR) [20,21]. Outliers are considered as data points that are significantly different from the remaining data, and IQR refers to the values that reside in the middle 50% of the scores [22].

The main challenge often lies in identifying the modules that are prone to defects rather than focusing solely on the non-defective modules since most of the modules have a lower defect ratio compared to non-defective modules. Various techniques have been analyzed in research studies to address this issue, including random undersampling (RUS), random oversampling (ROS), and the synthetic minority oversampling technique (SMOTE) [23,24]. These techniques aim to balance the data by giving equal weight to the minority class (defective modules) as well as the majority class (non-defective modules). In this research, the SMOTE technique was applied to balance the dataset as the maximum defect ratio in this dataset was 28%, which can tend to give higher accuracy with predicting non-defective modules, without identifying the defective modules. Since the efficacy of SMOTE lies in its capability to create new instances instead of merely duplicating existing ones, SMOTE has been applied in various studies in defect prediction problems [25–27]. This oversampling approach has demonstrated considerable success in the literature in overcoming the difficulties associated with imbalanced classes. This approach utilizes an oversampling approach in which the minority class is over-sampled by creating “synthetic” examples rather than by oversampling with replacement.

Aleem et al. [8] explored various machine learning techniques for software bug detection and conducted a comparative performance analysis among these algorithms. The study aimed to assess the effectiveness of different machine learning algorithms in detecting software bugs. Once the machine learning techniques are identified, it becomes crucial to determine which features should be selected for developing the SDP models. In this regard, Balogun et al. [5] developed aggregation-based multi-filter feature selection methods specifically for defect prediction.

One challenge when implementing a new software solution is obtaining the necessary historical information from the software repository of similar projects. In some cases, the available historical data may not be sufficient for training purposes, particularly when dealing with similar types of projects. To address this issue, cross-project defect prediction (CPDP) has emerged as a topic of interest in recent SDP research [28]. CPDP involves leveraging data from different projects to predict defects in a new project. Challenges with CPDP include variations in sizes, programming languages, development processes, etc.

It is often observed that multicollinearity can impact the performance of developed machine learning models. To overcome this limitation, various techniques, such as principal component analysis (PCA), ridge regression, etc., have been applied [29,30] in SDP studies. Yang and Wen [30] employed lasso regression and ridge regression in their study on developing SDP models, which improved performance. In this study, we aim to conduct a comparative study by removing the highly correlated features in classification models and evaluating the impact on model performance when retaining the correlated features.

In recent years, the need for explainable machine learning models has increased due to the growing need for explainable artificial intelligence (XAI). Gezici and Tarhan [31] developed an interpretable SDP model using the traditional model-agnostic techniques of SHAP, LIME, and EL5. They applied explainable techniques on the gradient boost classifier. Our research aims to develop interpretable SDP models using four different classifiers instead of a single one for comparability of explainability in different ML algorithms.

Jiarpakdee et al. [32] suggested that research practitioners need to invest more effort in investigating ways to enhance the comprehension of defect prediction models and their predictions.

3. Methodology

The implementation of the SDP model in this study is depicted in Figure 1. This process involves multiple steps, including data collection, feature selection, data preprocessing, model development using selected ML algorithms, and applying model-agnostic techniques. As it is important to interpret the model, the application of model agnostic technique has been highlighted in yellow. The detailed description of the SDP model methodology used in this study can be found below.

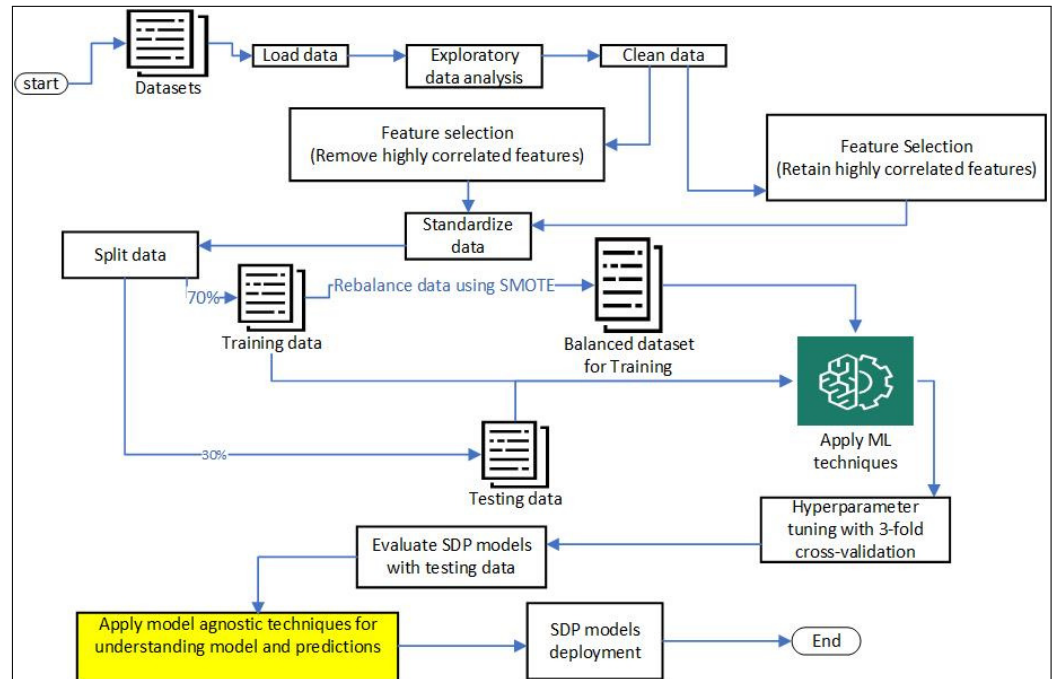


Figure 1. Methodology of developing interpretable SDP models.

3.1. Data Collection

We selected five different files, namely, jm1, pc1, kc1, kc2, and cm1, from the PROMISE repository [33]. These datasets incorporate McCabe and Halstead static code measure metrics. The projects list modules from various programs written in the C or C++ programming languages. Each of these selected files contains 21 independent variables, referred to as features in this study, and one target variable. The target variable indicates whether the selected module is defective or not. Some of the common measures include the total lines of code available in the program, McCabe’s cyclomatic complexity, Halstead’s effort, etc. The statistics of these selected datasets are presented in Table 1, followed by descriptions of each feature available in Table 2.

Table 1. Overview of the selected datasets from the PROMISE repository.

Project Name	Number of Instances	Programming Language	Number of Features	Defect Ratio
cm1	498	C	21	9.8%
kc2	512	C++	21	28.0%
pc1	1109	C	21	7.3%
kc1	2109	C++	21	26.0%
jm1	10,885	C	21	19.3%

Table 2. Individual field or feature description of the selected projects from the PROMISE dataset.

Feature Name	Description
loc	McCabe’s line count of code
l	Halstead’s program length
e	Halstead’s effort
loComment	Halstead’s count of lines of comments
v(g)	McCabe’s cyclomatic complexity
ev(g)	McCabe’s essential complexity
iv(g)	McCabe’s design complexity
n	Halstead’s total operators + operands
v	Halstead’s volume

Table 2. Cont.

Feature Name	Description
d	Halstead's program difficulty
i	Halstead's intelligence
b	Number of delivered bugs
t	Halstead's time estimator
loCode	Halstead's line count
loBlank	Halstead's count of blank lines
locCodeAndComment	Line of code and comment
uniq_Op	Unique operators
uniq_Opnd	Unique operands
total_Op	Total operators
total_Opnd	Total operands
branchCount	Total branches in program
Defects	Represents defective or non-defective module

3.2. Feature Selection and Preprocessing Steps

The process began with loading the stored data from the PROMISE repository. Subsequently, an exploratory data analysis process was conducted, and the features were cleaned and selected using the techniques described in Table 3. The categorical values of the target variable “defects” were mapped to 0 or 1, representing non-defective and defective modules, respectively.

Table 3. Data cleaning criteria applied on PROMISE dataset.

Criterion	Data Cleaning Activity	Explanation
1	Cases with missing values	Instances that contain one or more missing values were dropped from the dataset.
2	Cases with implausible and conflicting feature values	Instances that violated referential integrity constraints were removed. The following conditions were applied: (a) The total line of code is not an integer number. (b) The program's cyclomatic complexity is greater than the total operators plus 1 [34]. (c) Halstead's sum of total operators and operands is 0.
3	Outlier removals	Outliers rely on any value that lies within the range of the 1st and outside the range of the 3rd quartile, respectively. Records within the range of $(Q1 - 1.5 * IQR)$ and outside the range of $(Q3 + 1.5 * IQR)$ were dropped.
4	Removal of duplicates	Duplicated observations were taken out from the dataset.
5	Removal of highly correlated features except for module size, and effort metrics.	Calculated the correlation between independent features, and the attributes with more than 70% of correlation were removed in the first approach demonstrated in this study.

To validate the reliability of the cross-project defect prediction model, a separate dataset was created by merging all the selected files and stored in a Python dataframe named CP. A new field called “project” was introduced to identify whether the project information can influence the prediction. The projects were assigned numbers from 1 to 5 for identification purposes.

The feature selection process involves selecting a subset of the original dataset by removing irrelevant or redundant features from the original feature space. Feature selection algorithms have been used in various fields including in defect prediction [35–38]. The selection of feature subsets significantly affects the complexity and performance of classification algorithms. The challenge with the feature selection process is that too many features may increase the computational cost of the classifier, while too few features may

reduce the model performance [39]. Feature selection methods have two major benefits for classification tasks, which are reducing the data dimensionality and maintaining or improving the performance of the classifier.

Two different approaches were employed during the feature selection process in this study. Both of these approaches cleaned the dataset by removing records with missing variables, duplicated entries, outliers, and implausible and conflicting feature values. The outliers were removed to train the dataset by removing the extreme values and reducing the noise for better performance. The technique for outlier removal was based on keeping the value within the IQR range of Q1 and Q3, as shown in Table 3.

In addition to the above cleaning criteria, the first approach involved removing highly correlated features from the datasets. Researchers attempted to select features from the original feature space that have a high dependency on the class labels and low redundancy with other features [35]. To ensure the features have low redundancy with other independent features, except the field 'effort', the features that had a correlation value higher than 70% with other features, excluding the output variable of defective versus non-defective, were dropped. Figure 2 shows the pairwise correlation matrix of the independent variables or features of project pc1. Only the independent features were considered for comparing this correlation matrix as the objective was not to remove the features that are highly correlated with the output feature but rather to remove the duplicated information that is coming from two different variables. The reason behind considering removing the highly internally correlated features is to ensure that the features do not contain the same information. The impact of each feature in the model prediction should be interpretable and should be prominent for the prediction. From this correlation matrix, only the features that had an absolute value of 70% were retained for further comparison using this feature selection approach. The same technique is applied to other projects.

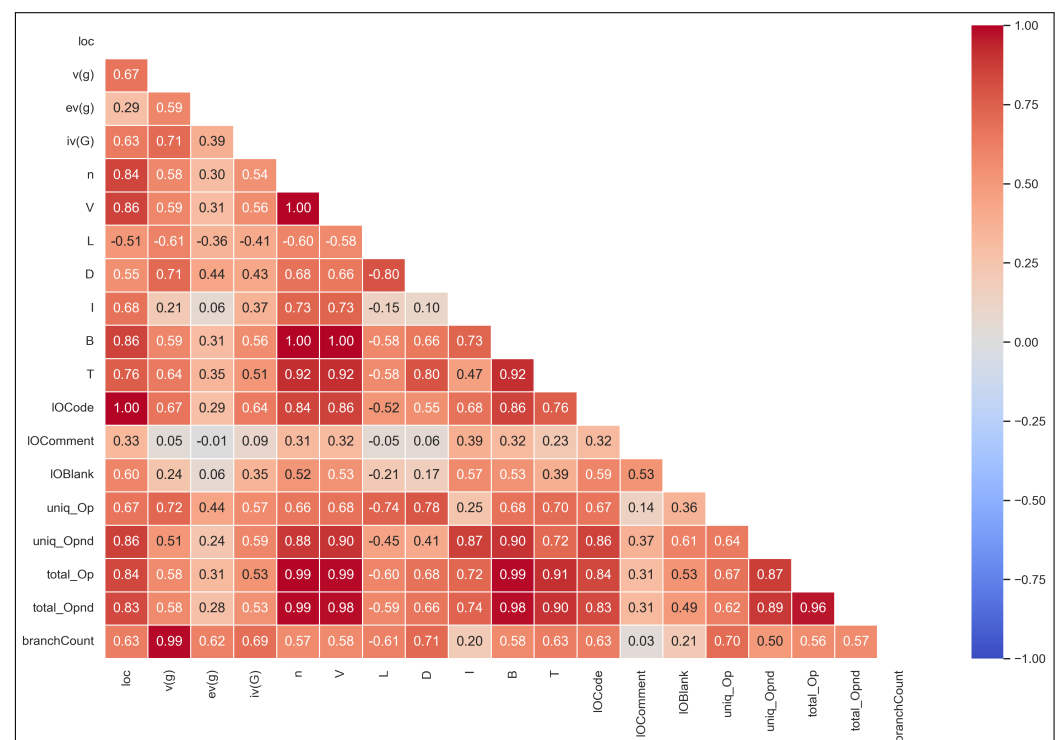


Figure 2. Pairwise correlation matrix from pc1 dataset.

The alternative approach followed the same cleaning steps as the first approach. However, they were retained during the SDP model development instead of removing the highly correlated features to identify if keeping the highly correlated features impacts the model performance.

The common features among all datasets were *e*, *loc*, *l*, *ev(g)*, and *IOComment* when the highly correlated features were removed from each dataset. The selected features, along with the project names, are shown in Table 4. It can be observed that the maximum number of nine features, compared to the original 20 features, was available in the cross-project dataset. At the same time, *kc2* and *jm1* had the minimum number of selected features.

Table 4. Features selection after removing highly correlated attributes.

File Name	Selected Features
<i>kc2</i>	<i>loc</i> , <i>ev(g)</i> , <i>l</i> , <i>e</i> , <i>IOComment</i>
<i>cm1</i>	<i>loc</i> , <i>ev(g)</i> , <i>l</i> , <i>e</i> , <i>IOCode</i> , <i>IOComment</i> , <i>IOBlank</i>
<i>pc1</i>	<i>loc</i> , <i>v(g)</i> , <i>ev(g)</i> , <i>l</i> , <i>e</i> , <i>IOComment</i> , <i>IOBlank</i>
<i>kc1</i>	<i>loc</i> , <i>ev(g)</i> , <i>l</i> , <i>e</i> , <i>IOComment</i> , <i>IOBlank</i>
<i>jm1</i>	<i>loc</i> , <i>l</i> , <i>i</i> , <i>e</i> , <i>IOComment</i>
Cross Project	<i>loc</i> , <i>v(g)</i> , <i>ev(g)</i> , <i>iv(g)</i> , <i>l</i> , <i>i</i> , <i>e</i> , <i>IOComment</i> , <i>project</i>

To develop SDP models, the data need to be divided between training and testing such that enough samples are available for training and validating the model. Some of the common approaches are an 80:20 or 70:30 split, and cross-validation. In several existing defect prediction studies [40–42], 70:30 splits were considered. In this work, each dataset was split into 70% for training and 30% for testing, which would give us a balance between training and testing the model with sufficient data for training and a reasonable number of records for evaluation. Finally, since this approach has been used in earlier research, having a split of 70:30 would make it easier to compare and reproduce results across different studies.

To standardize the features by removing the mean and scaling to unit variance, *StandardScaler* from the Python *scikit-learn* library was used [43]. Furthermore, since the dataset was imbalanced, it was crucial to employ techniques to prevent the SDP model from being biased towards predicting the majority classes (non-defective value with 0) only. Consequently, in the training dataset, the synthetic minority oversampling technique (SMOTE) technique was applied to match the number of instances in the minority class (defective) to the number of instances in the majority class (non-defective), while the testing dataset remained unchanged for validation purposes.

The distribution of samples with defective versus non-defective modules in the finalized training dataset is presented in Table 5. For example, the *kc2* dataset initially had 21 independent features with a total of 522 records or instances. After cleaning the data, the number of available instances in the same dataset was reduced to 198. These records were then split between training and testing with a 70:30 split, allocating 138 instances towards training with the remaining 60 instances kept for testing. In the training dataset, out of these 138 instances, 30 instances were classified as defective and the remaining 108 were marked as non-defective. Subsequently, SMOTE allowed the data to have an equal distribution of 108 records for both defective and non-defective modules, which made the training dataset oversampled with a total number of instances of 276. The SMOTE technique has not been applied in testing datasets as the goal is not to train the dataset with a balanced dataset but to test the overall performance without making any manipulation of the original data.

Table 5. Distribution of defective vs. non-defective samples before and after applying SMOTE in the training dataset.

Project Name	Attributes	Number of Original Instances	Number of Cleaned Instances	Defective vs. Non-Defective Instances in Training Dataset	Defective vs. Non-Defective instances after Applying SMOTE
kc2	5	522	198	30, 108	108, 108
cm1	7	498	289	20, 182	182, 182
pc1	7	1109	601	23, 397	397, 397
kc1	6	2109	718	116, 386	386, 386
jm1	5	10,885	4574	509, 2692	2692, 2692
CP	9	15,123	4608	491, 2734	2734, 2734

3.3. Applied Machine Learning Algorithms

After the data preprocessing steps, we selected widely used machine learning (ML) algorithms that have been applied in previous SDP studies on classification problems [8], namely, support vector machine (SVM), k-nearest-neighbors (KNN), random forest classifiers (RF), and artificial neural networks (ANNs). To improve the performance of each of these models, hyperparameter tuning [44,45] was performed using the grid search method [43] with 3-fold cross-validation on the training dataset. The predictors of the defect prediction models can be optimized by tuning the parameters of the algorithm. Depending on the algorithm applied, the hyperparameter tuning involves tuning the parameters of the algorithms to improve the performance of the developed models. For the SVM model, the parameters that were tuned were ‘C’, with values of 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, and 14, ‘gamma’, with values of 1, 0.1, 0.01, and 0.001, and the kernel parameter was tuned with linear, ‘poly’, ‘rbf’, and ‘sigmoid’. The KNN algorithm depends on the parameter n_neighbors. This algorithm was optimized with the values of 1, 2, 3, 4, 5, 10, 15, and 20 for n_neighbors. The random forest algorithm has several parameters that can affect the performance of the algorithm. We have attempted the use of parameters of ‘n_estimators’, with corresponding values of 3, 10, and 12, ‘max_depth’, with values of 10, 20, 30, 40, and 100, while ‘min_samples_leaf’ was optimized with values of 1, 2, 4, and 8, and the “criterion” parameter was attempted with ‘gini’, ‘entropy’ and ‘log_loss’. Finally, for the ANN algorithm, the epochs, batch size, and hidden layers were tuned to obtain the best performance.

Next, the paper attempts to analyze and compare various methodologies to tune the defect predictors. Once the models were developed, LIME was applied for local prediction on the best model for each of the projects and SHAP was applied for global explanations on the same classifiers. Finally, all the projects were combined into a single dataset to examine the impact of cross-project data and the obtained results were validated. Support vector machine (SVM) is a supervised ML algorithm that defines a hyperplane to separate the data most optimally, ensuring a wide margin between the hyperplane and the observations [8,46]. K-nearest-neighbors (KNN) is a supervised ML algorithm that utilizes proximity to classify or predict the grouping of individual data points [47,48] due to its efficiency. Random forest classifier (RF) is a supervised ensemble ML algorithm that incorporates a combination of tree predictors, where each tree depends on the values of a random vector sampled independently with the same distribution for all trees in the forest [49,50]. Artificial neural networks (ANNs) [15] are a collection of neurons, where each of these neurons or layers is vertically concatenated. An ANN model consists of an input layer and hidden layers and the prediction is performed in the output layer.

3.3.1. Model Interpretability

LIME and SHAP techniques were employed as both of these techniques have been popular in the field of machine learning explainability. Details of these techniques are explained below.

LIME (Local Interpretable Model-Agnostic Explanations)

The LIME [51] technique proposed by Ribeiro et al. constructs a surrogate sparse linear model around each prediction to elucidate the workings of the black box model within that local context [52]. The LIME model can work for tabular data, text, and images. The software defect prediction models utilized in this study included tabular data, and LIME was applied to explain the tabular data on the best-performing model. The LIME equation [51] for interpretability can be expressed as follows:

$$\zeta(x) = \operatorname{argmin}_{g \in G} [L(f, g, \pi_x) + \Omega(g)] \quad (1)$$

In Equation (1), the term $\zeta(x)$ represents the explanation for the prediction made by the SDP model for the instance x . It aims to select an interpretable explanation model for software defect prediction g from the class G , minimizing a combination of fidelity to the original SDP model f and complexity. The fidelity function $L(f, g, \pi_x)$ measures how well g approximates f in the local neighborhood of x , while $\Omega(g)$ quantifies the simplicity of g . By minimizing this combined objective, LIME ensures the explanation $\zeta(x)$ faithfully represents f while being interpretable [53].

As explained in the LIME equation, The SDP model developed based on the random forest model would generate a LIME explanation for a specific record or instance that would start with randomly generated instances by perturbing the data surrounding the instance of interest [54]. Next, LIME uses the black box model, which would be the SDP model generated based on the random forest example that was taken in this scenario to generate predictions of the generated random instances. Afterward, LIME constructs a local regression model using the generated random instances and their generated predictions from the black box model or the given random forest model used in this example. Finally, the coefficients of the regression model indicate the contribution of each metric to the prediction of defective or non-defective modules.

SHAP (SHapley Additive exPlanation)

Developed by Lundberg et al. [55], SHAP corresponds to the idea of Shapley values for model feature influence scoring [55]. The Shapley value corresponds to the average marginal contribution of a feature value over all possible coalitions [51,56]. SHAP can quantitatively explain the prediction of a machine learning model. The Shapley value is a mathematical theory used to determine the contribution of game participants to the game results. It is a method to calculate the contribution of each eigenvalue to the predicted value, which is expressed by the formula:

$$f(z) = f(z') + \sum_{j=1}^M \theta_j (z_j - z'_j) \quad (2)$$

Equation (2) represents the prediction made by the SDP model, where $f(z)$ is the output based on the input features z , and $f(z')$ is the output of a simpler linear model. The term $\sum_{j=1}^M \theta_j (z_j - z'_j)$ captures the difference between the predictions of the SDP model and the simpler linear model, with θ_j representing the Shapley value of each feature and $z_j - z'_j$ representing the deviation of the feature values from a reference point.

$$\phi_i(v) = \sum_{S \subseteq N \setminus i} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [v(S \cup i) - v(S)] \quad (3)$$

Equation (3) calculates the Shapley value $\phi_i(v)$ for a specific feature i , considering all possible subsets S of features. It quantifies the marginal contribution of feature i to the difference in predictions made by the model v for different subsets of features.

Together, these equations illustrate how the Shapley values are used to explain the contribution of individual features to the predictions of the SDP model, shedding light on its working principle and factors influencing the output results.

3.4. Evaluation Strategy

This work verified the performance of the SDP models using effective evaluation strategies for classification models in the literature using precision, recall, the F1-score, accuracy, and the AUC score. For the final measurement before applying model-agnostic explanations, the best performing SDP models evaluated based on accuracy and the AUC score were selected. The selected evaluation metrics are widely used in the literature for the evaluation of classification models' performance [8].

Accuracy is the percentage of correctly classified results [8]. Accuracy can be calculated using Equation (4).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

Precision measures the accuracy of positive predictions made by the model. It is the ratio of true positives (correctly predicted positive instances) to the sum of true positives and false positives (incorrectly predicted positive instances), whereas recall measures the ability of the model to correctly identify all positive instances. It is the ratio of true positives to the sum of true positives and false negatives (positive instances incorrectly classified as negative). The F1-score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall.

The area under the receiving operating characteristics (ROC) curve (AUC score) [46] is a measure of how well a parameter can distinguish between two classes (defective/non-defective). The "true positive rate" (TPR) is the proportion of instances labeled as defective that were correctly predicted, and the "false positive rate" (FPR) is the proportion of instances labeled as non-defective that are incorrectly predicted as defective. The higher the AUC score, the better the model's performance at distinguishing between the positive and negative classes; the objective is to obtain an AUC score of greater than 0.50.

Finally, to compare the performance of defect prediction using cross-project metrics with individual project metrics, the average accuracy and AUC scores were calculated. The differences between the average scores and the cross-project defect prediction models were examined for each classifier.

4. Results

This section presents the results in the order of the software defect prediction model based on an original dataset with a comparison of retaining highly correlated features and removing highly correlated features without attempting to adjust the imbalanced dataset.

4.1. Performance of Machine Learning Algorithms on Original Dataset

Table 6 presents the results obtained from the original dataset while keeping the majority of the features, and Table 7 shows the performance of the SDP models on the original dataset on the reduced features using a removal of highly correlated features selection strategy.

Both of the tables show the project name, the machine learning techniques applied, and the evaluation metrics of accuracy, precision, recall, F1-score, and AUC, respectively. The numbers in bold format show the highest value of each of the evaluation metrics for each of these selected projects.

When all the features were retained in the datasets, the accuracy of the models was in a range between 67% and 96%. The maximum precision was 67% when the KNN algorithm was applied in the jm1 project. The jm1 project had 4574 instances, which made this dataset larger compared to the other datasets except the cross-project (cp) dataset used in this study. The CP project had a precision value ranging from 21% to 34% depending on the algorithm applied. The precision score among all the projects had a wide range starting

from a zero value to a maximum score of 67%. The highest recall value found when applied on these selected projects was 40% and this score was achieved when the SVM algorithm was applied in the cp project. The maximum F1-score was 28% with zero or very low scores found from the majority of the algorithms. The highest AUC score was 60%. Relatively, the ANN algorithm and RF algorithms performed better than the other algorithms for the selected datasets.

Table 6. Performance of the evaluation metrics on the original dataset with retention of highly correlated features.

Project Name	ML Technique	Accuracy	Precision	Recall	F1-Score	AUC
kc1	SVM	0.75	0.00	0.00	0.00	0.50
	KNN	0.75	0.00	0.00	0.00	0.50
	RF	0.74	0.43	0.19	0.26	0.55
	ANN	0.79	0.43	0.21	0.29	0.57
cm1	SVM	0.91	0.00	0.00	0.00	0.48
	KNN	0.67	0.04	0.25	0.06	0.47
	RF	0.83	0.08	0.25	0.12	0.55
	ANN	0.90	0.14	0.25	0.18	0.59
kc2	SVM	0.73	0.40	0.13	0.20	0.53
	KNN	0.75	0.00	0.00	0.00	0.50
	RF	0.73	0.45	0.33	0.38	0.60
	ANN	0.75	0.00	0.00	0.00	0.50
pc1	SVM	0.96	0.00	0.00	0.00	0.50
	KNN	0.96	0.00	0.00	0.00	0.50
	RF	0.96	0.00	0.00	0.00	0.50
	ANN	0.95	0.25	0.14	0.18	0.56
jm1	SVM	0.82	0.22	0.06	0.09	0.51
	KNN	0.84	0.67	0.01	0.02	0.50
	RF	0.84	0.50	0.01	0.02	0.50
	ANN	0.85	0.62	0.04	0.07	0.52
CP	SVM	0.69	0.21	0.40	0.28	0.57
	KNN	0.75	0.24	0.29	0.26	0.56
	RF	0.83	0.34	0.17	0.23	0.56
	ANN	0.74	0.23	0.32	0.27	0.57

The numbers in bold represent the highest score in each of the evaluation metrics by projects.

The AUC score from most of the algorithms was around 50, apart from a few exceptions with random forest or the ANN algorithm. This does not give us confidence that the models are fully reliable as the majority of the predictions are probably biased toward predicting the non-defective module. Applying feature selection algorithms in the datasets by reducing the highly correlated features, as shown in Table 7, did not demonstrate a significant difference from the results obtained in Table 6, where all the features were retained. The accuracy score ranged from 72% to 95%. However, when the majority of the features were kept, the accuracy ranged from 67% to 96%. The lower bound for accuracy is higher compared to the dataset with reduced features.

The AUC score was slightly better for the pc1 project with reduced features when the random forest algorithm was applied, which showed an AUC score of 63% and an accuracy score of 95%. In terms of precision, except for the random forest model, which was able to detect all the positive instances on the cm1 project, the score varied for other projects, ranging from zero to 50%. The maximum recall value of the reduced features datasets was 29% compared to 40% when all the features were retained.

The maximum F1-score was 31% when the random forest algorithm was applied in the pc1 project with reduced features. On the other hand, the highest F1-score was 38% when the random forest algorithm was applied in the kc2 project with all features, but had a score of zero on the pc1 project when the random forest algorithm was applied. On the

reduced features datasets, the random forest algorithm worked better compared to the other algorithms with relatively higher accuracy and AUC score in all projects. To see the interpretation of this random forest model on the original dataset with reduced features, the LIME technique was applied on a single instance of the pc1 project to illustrate the local explanation, whereas SHAP was applied on the same project when the random forest algorithm was applied.

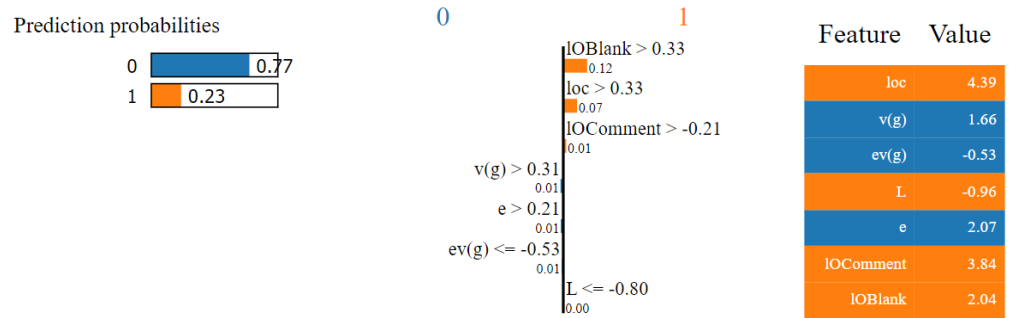
Table 7. Performance of the evaluation metrics on the original dataset with reduced features through the feature selection process of removing highly correlated features.

Project Name	ML Technique	Accuracy	Precision	Recall	FI-Score	AUC
kc1	SVM	0.81	0.00	0.00	0.00	0.50
	KNN	0.72	0.24	0.21	0.23	0.53
	RF	0.81	0.50	0.17	0.25	0.56
	ANN	0.80	0.43	0.07	0.12	0.52
cm1	SVM	0.85	0.00	0.00	0.00	0.46
	KNN	0.92	0.00	0.00	0.00	0.50
	RF	0.93	1.00	0.14	0.25	0.57
	ANN	0.92	0.50	0.14	0.22	0.56
kc2	SVM	0.77	0.00	0.00	0.00	0.50
	KNN	0.77	0.00	0.00	0.00	0.50
	RF	0.73	0.25	0.07	0.11	0.50
	ANN	0.75	0.00	0.00	0.00	0.49
pc1	SVM	0.93	0.20	0.29	0.24	0.62
	KNN	0.95	0.00	0.00	0.00	0.49
	RF	0.95	0.33	0.28	0.31	0.63
	ANN	0.93	0.22	0.29	0.25	0.62
jm1	SVM	0.83	0.16	0.02	0.03	0.50
	KNN	0.84	0.00	0.00	0.00	0.50
	RF	0.84	0.00	0.00	0.00	0.50
	ANN	0.84	0.00	0.00	0.00	0.50
CP	SVM	0.83	0.22	0.05	0.08	0.51
	KNN	0.85	0.50	0.00	0.01	0.50
	RF	0.85	0.33	0.01	0.02	0.50
	ANN	0.85	0.40	0.02	0.04	0.51

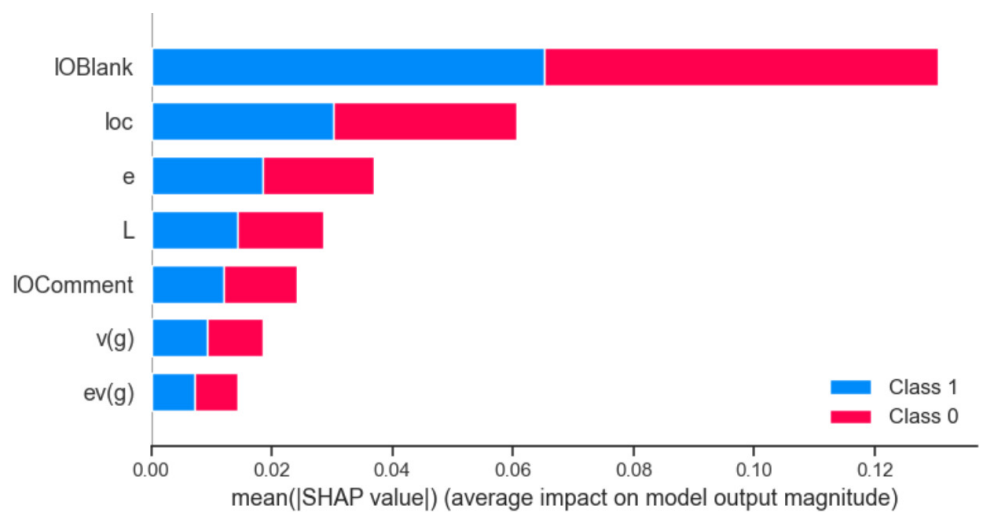
The numbers in bold represent the highest score in each of the evaluation metrics by projects.

Figure 3 demonstrates the local and global interpretation of the pc1 project when the random forest algorithm was applied to the dataset with reduced features. Figure 3a demonstrates the local interpretation by the LIME technique for the selected instance. The LIME model predicted the outcome of this instance as 0, with a confidence of 77% of this instance not being defective and 23% probability of this instance being defective. The orange color shows the probability of being defective, and the blue represents the probability of being non-defective. The right-hand side of the picture shows the values of loc (line of code), l (length), LOComment, and IOBlank are contributing towards the module being defective, whereas the cyclomatic complexity, essential complexity, and effort values are contributing to the model for predicting this instance as defective. Since the total feature ranking of the model as non-defective is higher, this instance is considered as 0 or non-defective.

Figure 3b shows the global explanation of the same project when SHAP was applied. This graph shows IOBlank makes the most contribution towards the model followed by loc, effort, length, and the other features. The blue and red color distribution shows that there was no bias towards predicting defective versus non-defective modules.



(a) LIME model—dataset contains reduced features.



(b) SHAP model—dataset contains reduced features.

Figure 3. SDP model interpretation on the original dataset of project pc1 with reduced features.

4.2. Performance of Machine Learning Algorithms on Balanced Dataset

Table 8 presents the results obtained from the dataset after the oversampling technique SMOTE was applied while keeping the majority of the features, and Table 9 shows the performance of the SDP models on the balanced dataset where highly correlated features were removed.

Both of the tables show the header table information of the project name, the machine learning techniques applied, and the evaluation metrics of accuracy, precision, recall, F1-score, and AUC, respectively, as demonstrated in the model performance on the original datasets. When all the features were retained in the datasets, the accuracy of the models was in a range between 61% and 96%. The maximum precision was 47% when the KNN algorithm was applied in the kc2 project. The maximum recall value was 71%, which is much higher than the recall value that was found in the original dataset. The F1-score ranged from 16% to 53%, whereas in the original dataset, the maximum F1-score was 28%, with zero or very low scores found from the majority of the algorithms. The highest AUC score in the balanced dataset was 77% for the pc1 project when the ANN algorithm was applied, whereas the maximum AUC score on the all features' original dataset was 60%. The performance of the algorithms varied. For different projects with varying numbers of instances, the applied algorithms performed differently.

Table 8. Performance of the models after applying SMOTE on datasets that retained highly correlated features.

Project Name	ML Technique	Accuracy	Precision	Recall	F1-Score	AUC
kc1	SVM	0.65	0.28	0.50	0.36	0.60
	KNN	0.66	0.18	0.21	0.20	0.50
	RF	0.67	0.16	0.17	0.16	0.48
	ANN	0.61	0.25	0.53	0.34	0.58
cm1	SVM	0.84	0.11	0.14	0.35	0.52
	KNN	0.64	0.14	0.71	0.24	0.68
	RF	0.82	0.09	0.14	0.11	0.51
	ANN	0.89	0.29	0.29	0.29	0.61
kc2	SVM	0.63	0.32	0.40	0.35	0.56
	KNN	0.73	0.47	0.60	0.53	0.69
	RF	0.67	0.37	0.47	0.41	0.60
	ANN	0.65	0.36	0.53	0.43	0.61
pc1	SVM	0.96	0.43	0.43	0.43	0.70
	KNN	0.88	0.18	0.57	0.28	0.73
	RF	0.93	0.25	0.43	0.32	0.69
	ANN	0.95	0.40	0.57	0.47	0.77
jm1	SVM	0.71	0.23	0.38	0.29	0.57
	KNN	0.77	0.29	0.35	0.32	0.60
	RF	0.77	0.30	0.34	0.31	0.59
	ANN	0.69	0.24	0.47	0.32	0.60
CP	SVM	0.69	0.21	0.40	0.28	0.57
	KNN	0.75	0.24	0.29	0.26	0.56
	RF	0.83	0.34	0.17	0.23	0.56
	ANN	0.73	0.24	0.53	0.29	0.58

The numbers in bold represent the highest score in each of the evaluation metrics by projects.

Applying feature selection algorithms in the datasets by reducing the highly correlated features, as shown in Table 9, did not demonstrate a significant difference from the results obtained in Table 8, where all the features were retained. The accuracy score ranged from 56% to 92%. However, when the majority of the features were kept, the accuracy ranged from 61% to 96%.

The minimum AUC score was 53% and the highest score for this metric was 71%. The minimum precision score was 18% and the maximum precision score was 38%. For the original dataset, several algorithms showed zero values for precision, which means the accuracy of predicting defective instances of the model was not satisfactory in the original dataset.

The recall has all positive values ranging from 15% to 71%. The F1-score also shows all positive values, unlike the original dataset, where a couple of the algorithms returned zero scores on these projects.

Table 10 demonstrates the final evaluation by summarizing the content obtained from Tables 8 and 9. We considered accuracy and the AUC score before selecting a model for applying the model-agnostic technique. Although having a high AUC score does not guarantee high value for precision, recall, and the F1-score, we note from this table that when the AUC was high, the accuracy, precision, recall, and F1-score were in an acceptable range in the balanced dataset with non-zero values.

Table 9. Evaluation metrics after SMOTE was applied in the training dataset after removing the highly correlated features.

Project Name	ML Technique	Accuracy	Precision	Recall	FI-Score	AUC
kc1	SVM	0.68	0.29	0.45	0.35	0.59
	KNN	0.69	0.27	0.33	0.30	0.56
	RF	0.73	0.21	0.29	0.29	0.56
	ANN	0.70	0.29	0.38	0.33	0.58
cm1	SVM	0.56	0.08	0.43	0.14	0.50
	KNN	0.70	0.17	0.71	0.28	0.71
	RF	0.90	0.38	0.43	0.40	0.68
	ANN	0.83	0.17	0.29	0.21	0.58
kc2	SVM	0.75	0.50	0.53	0.52	0.68
	KNN	0.73	0.48	0.67	0.56	0.71
	RF	0.70	0.42	0.53	0.47	0.64
	ANN	0.73	0.47	0.53	0.25	0.67
pc1	SVM	0.90	0.18	0.43	0.25	0.67
	KNN	0.85	0.14	0.57	0.22	0.71
	RF	0.90	0.17	0.43	0.24	0.67
	ANN	0.92	0.21	0.43	0.29	0.68
jm1	SVM	0.64	0.19	0.40	0.26	0.54
	KNN	0.58	0.19	0.53	0.28	0.56
	RF	0.66	0.21	0.43	0.28	0.57
	ANN	0.63	0.20	0.44	0.27	0.55
CP	SVM	0.66	0.21	0.46	0.29	0.58
	KNN	0.71	0.18	0.27	0.22	0.53
	RF	0.80	0.25	0.15	0.19	0.54
	ANN	0.70	0.24	0.48	0.32	0.61

The numbers in bold represent the highest score in each of the evaluation metrics by projects.

In Table 10, the projects are listed in order of the dataset size where the number of instances available in the cleaned dataset are as shown in Table 5. The numbers in bold format show the highest accuracy score and the AUC score of each of the projects, whereas the highlighted yellow fields represent the model where the AUC score was highest for the selected project. Since this was an imbalanced dataset, rather than prioritizing the accuracy score, we considered the classifiers with the highest AUC score as the best-performing model for the referred project. For instance, when the cm1 project with all features was considered, a higher accuracy score was observed in the ANN classifier with a value of 89% compared to an accuracy value of 64% in the KNN classifier. However, we considered the KNN classifier as the best performing model, as the AUC score was highest among all the classifiers, with a value of 68%, as our primary goal was to identify the defective modules rather than detecting non-defective modules only.

It appears in both cases of selecting all features versus selecting reduced features that the KNN algorithm performed well for relatively smaller projects, namely kc2 and cm1. Also, it is worth noting that the KNN model achieved a higher accuracy score of 75% in the cross-project model, surpassing the average accuracy of the independent project scores of 74%. On the other hand, the obtained AUC score of 56% applied in the same classifier can be treated as relatively poor. Taking into consideration the overall AUC and accuracy, the ANN model outperformed the other classifiers in the cross-project models.

The top half of this Table 10 illustrates the accuracy and AUC score of each of the projects on the selected machine learning algorithms. These algorithms have been applied in the full-feature datasets except for the LocCodeAndComment feature. The bottom half of the figure followed the same strategy except that the SDP models were created on the reduced-feature datasets, which were obtained after removing the highly correlated independent features. The row denoted "Average" calculates the average of the accuracy and

AUC score for each of the projects on the SVM, KNN, RF, and ANN models. The average score has been compared with the cross-project(CP) SDP model scores.

Table 10. Comparison of the performance of each project along with an average of individual project scores with cross-project scores for the approach selected with the most features and reduced features.

Projects with 20 Common Features								
Project Name	SVM		KNN		RF		ANN	
	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
kc2	0.63	0.56	0.73	0.69	0.67	0.60	0.65	0.61
cm1	0.84	0.52	0.64	0.68	0.82	0.51	0.89	0.61
pc1	0.96	0.70	0.88	0.73	0.93	0.69	0.95	0.77
kc1	0.65	0.59	0.66	0.49	0.67	0.48	0.62	0.58
jm1	0.71	0.57	0.77	0.60	0.77	0.59	0.69	0.60
Average	0.76	0.59	0.74	0.64	0.77	0.57	0.76	0.63
CP	0.69	0.57	0/75	0.56	0.82	0.56	0.73	0.58
Projects with Reduced Features								
Project Name	SVM		KNN		RF		ANN	
	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
kc2	0.75	0.68	0.73	0.71	0.70	0.64	0.73	0.67
cm1	0.56	0.50	0.70	0.71	0.90	0.68	0.83	0.58
pc1	0.90	0.67	0.85	0.71	0.90	0.67	0.92	0.68
kc1	0.68	0.59	0.69	0.56	0.73	0.56	0.69	0.58
jm1	0.64	0.54	0.58	0.56	0.66	0.57	0.63	0.55
Average	0.71	0.60	0.71	0.65	0.78	0.62	0.76	0.61
CP	0.66	0.58	0.71	0.53	0.80	0.54	0.70	0.61

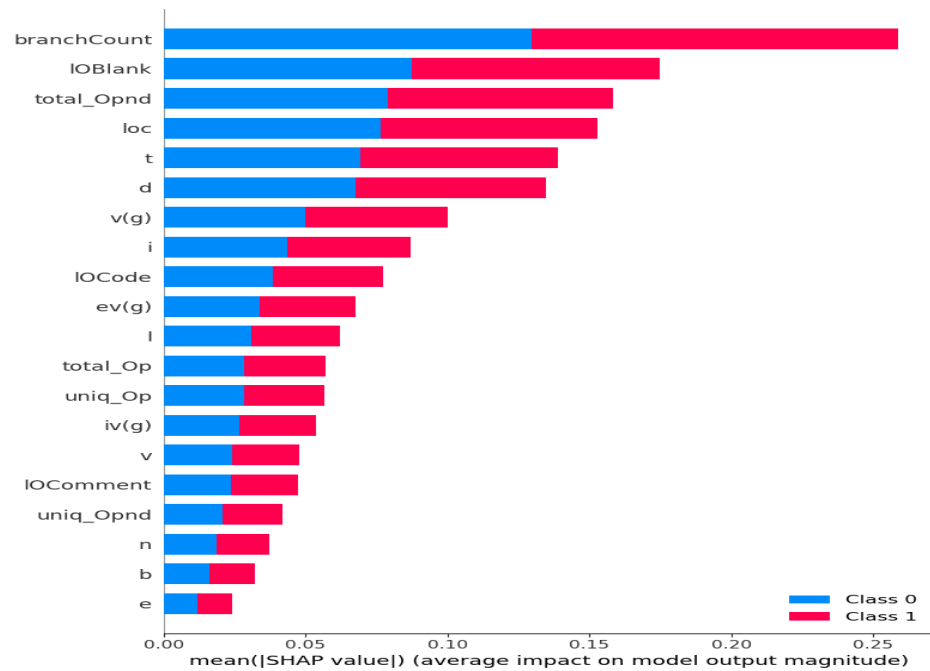
The numbers in bold represent either the highest accuracy or AUC for the referred project among the applied algorithms. The numbers highlighted in yellow represent the models with the highest AUC score.

The SVM model performed better on the kc1 project, which consists of a dataset with 718 instances available after filters were applied. Compared to the other datasets, this one is considered mid-size (Table 5).

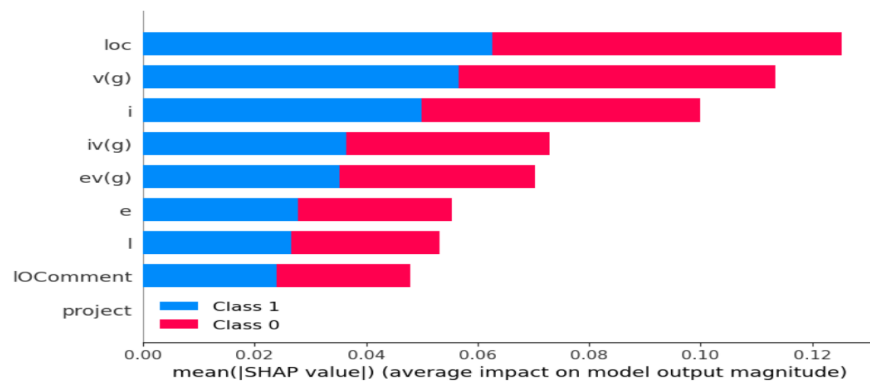
In terms of the cross-project datasets, the ANN model developed on the reduced features achieved a slightly higher AUC score of 61% compared to 58%. For the pc1 project, which had mid-sized samples, the ANN classifier showed higher accuracy, with values of 95% and 92% for the all features and reduced features datasets, respectively. Although for the pc1 dataset, the same ANN model with all features showed the highest AUC of 77%, there was a reduction in the AUC score, with a value of 68% on the reduced feature dataset.

Figure 4a demonstrates the cross-project defect prediction model developed using SHAP on the ANN model incorporating all features on the balanced dataset. The highest ranking of the mean SHAP value for this project is for branchcount followed by IOBlank. This model has all the features, but the impact of the attribute effort “e” is in the lowest position compared to the other source code and size metrics. In the same ANN classifier with reduced features, SHAP was applied to provide the global explanation, as shown in Figure 4b. This figure illustrates that in the SDP model constructed with the ANN classifier, the “loc” attribute makes the highest contribution towards the prediction outcome followed by cyclomatic complexity and intelligence. “Effort” is considered a predictor that has more importance than program length and lines of code, but less than program complexity-

related features. Additionally, the “project” field is shown to have the lowest importance. This indicates that the project source, which includes cross-project information, does not significantly impact the development of this model.



(a) SHAP model—dataset contains all features.



(b) SHAP model—dataset contains reduced features.

Figure 4. SHAP applied on cross-project dataset developed using ANN model.

Figure 5 demonstrates the interpretation of a local observation of the same ANN-based model using LIME. The LIME model predicted this record as non-defective with a confidence level of 71%. In this figure, the blue color represents the feature contribution towards being predictive as a non-defective instance compared to the orange color moving toward a defective module. The effort attribute was one of the important contributors in this prediction model; a value of less than -0.61 pushes the prediction towards a value of 0 (non-defective). Similarly, the cyclomatic complexity of the code was less than -64 , and intelligence also contributes to the prediction of non-defectiveness. It seems that the attribute “intelligence”, which determines the amount of intelligence presented in the program, was lower than the given threshold of -75 to be considered defective. This local prediction explains that the selected module had a lower value than the set threshold for intelligence, cyclomatic complexity, and intelligence to be considered defective. At the same time, the same prediction shows the probability of 29% of this module being defective as the loc, l, and LOComments are higher and ev(g) is less than the given threshold. The test

lead can better reflect the outcome by observing the impact of each of the attributes on this prediction. This aligns with the logical concept that software that requires less effort and is not very complex may have relatively fewer bugs compared to a complex system.

The interpretability of the kc1 project with mid-sample size (based on the number of instances available) is examined in Figures 6 and 7. Both models interpreted with LIME and SHAP show the importance of the “effort” feature for this algorithm. The local prediction using the LIME method was able to explain the observation with 60% confidence that this module is not likely to be defective. In both models, the “loc” feature is less important compared to the effort metrics. We observe that there can be slight differences between these predictions—the global agnostic model provides generic information, while the LIME method offers insight into individual predictions.

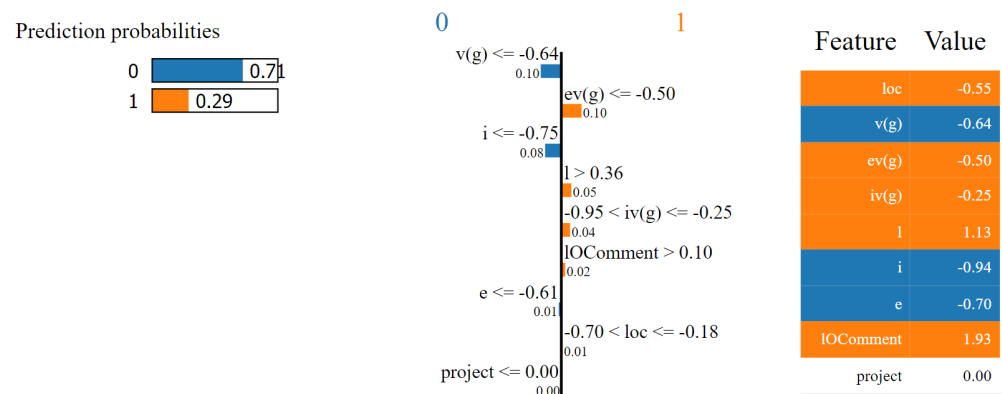


Figure 5. Demonstration of local interpretation of selected records from cross-project data when ANN was applied on reduced-feature-based model.

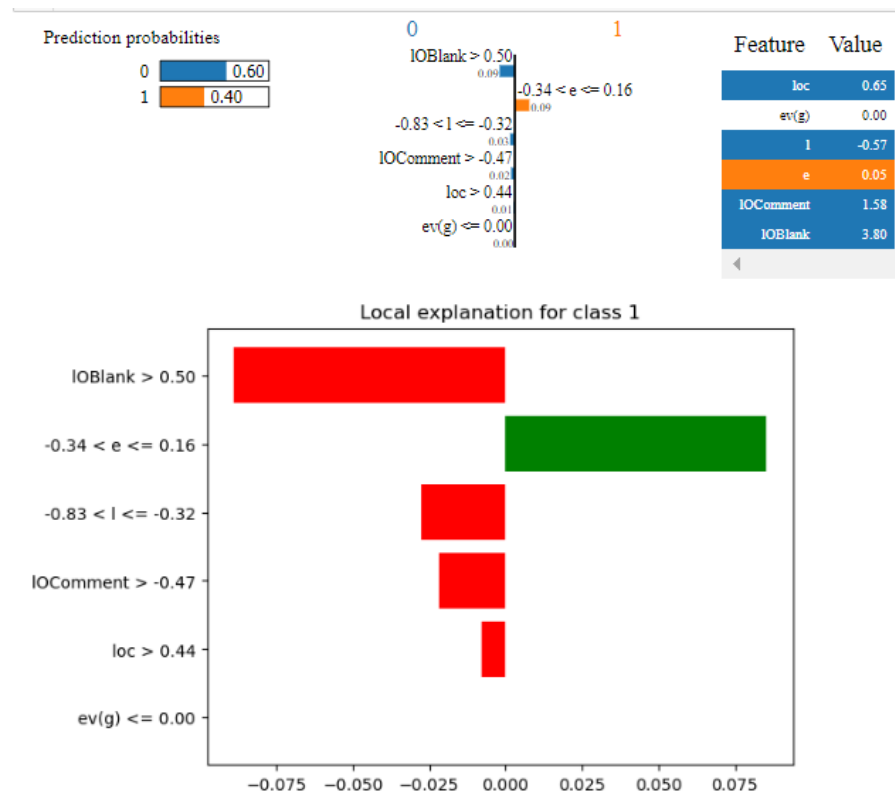


Figure 6. Model interpretation of SDP model generated from kc1 file with LIME on SVM classifier. The green color bar shows the features that are contributing to increasing the probability of this instance being defective, and red represents the feature’s probability of not being defective.

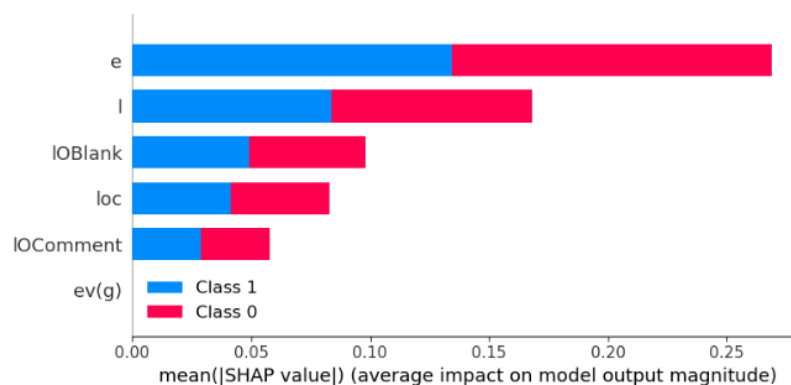


Figure 7. Model interpretation of SDP model generated from kc1 file with SHAP on SVM classifier.

5. Analysis and Discussion

In this study, we aimed to address multiple research questions and the findings are as follows:

Regarding RQ1, the predicted models provided consistent results, with slight variations depending on the sample size and selected features. For example, the smallest sample size after cleaning the data was project kc2, and the KNN model performed the best in both feature selection approaches. Additionally, the cross-project data yielded good results for the ANN classifier, as deep learning models tend to perform better with larger datasets. It was observed that there were trade-offs between accuracy and AUC; project pc1 demonstrated the highest performance with an accuracy above 90% and an AUC score close to 70%. It was observed that the “effort” attribute was not necessarily the most influential of all the outcomes. The SHAP and LIME models applied to the developed classifiers showed that for cross-project defect prediction, “effort” ranked lower in terms of feature importance. However, for a few individual projects, “effort” ranked second in terms of importance. This finding aligns with the understanding that project size and complexity are crucial factors in defect prediction.

RQ2 investigated the impact of removing highly correlated independent features. It was found that this removal improved the average AUC score for the SVM, KNN, and RF models with a slight decrease in the ANN model. This indicates that removing correlated features had varying effects on model performance but did not significantly alter the overall outcomes. The SDP models became easier to interpret with the SHAP and LIME techniques, as having more features would show blank values for features that did not contribute to the predicted outcomes. Although the SVM and KNN models’ accuracy decreased slightly with reduced features, this was compensated by a relatively higher AUC score.

To address RQ3, we compared the cross-project (CP) model score with an average score from the models built from individual projects. It appears that the CP models performed slightly lower than the individual projects’ average performance. However, the performance did not go down drastically, and the results were very close to the average score. For instance, the ANN model with reduced features shows the average accuracy and AUC score as 76% and 61%, whereas the cross-project model had accuracy and AUC values of 70% and 61%, respectively. CP can be useful when historical information is not available for similar projects.

Finally, RQ4 focused on interpreting the predictions even after applying oversampling using SMOTE. It was found that both SHAP and LIME successfully explained the predictions, providing insights into the contributing features.

For the feature selection algorithm, we used a threshold of 70%. Further experiments can be conducted to find the best threshold for selecting features when multicollinearity exists.

We experimented with both imbalanced data and after applying the SMOTE technique for creating synthetic data for the minority class. Training the model with more balanced data would enhance confidence in generalizing the model.

Also, for comparison, five projects were merged from the same source to observe the impact of cross-project defect prediction. This can be extended by using data from various sources to bring more variation to the dataset.

6. Conclusions and Future Work

This study aimed to develop software defect prediction models with a focus on interpretability, using individual projects and by combining the individual projects with a cross-project dataset. Feature selection was applied by reducing and retaining the highly correlated data for developing comparative studies. Test managers may need to work with a subset of features rather than having all the features due to not having historical information on various metrics. Our research indicates that removing the multicollinear features yielded consistent results.

The findings indicate that the cross-project defect prediction model does not significantly compromise performance. While the average of the individual projects generally achieved better AUC scores, the results were comparable to the scores of the cross-projects. This implies that when historical information for an exact similar project is unavailable, users can still utilize the cross-project dataset for predicting defective outcomes in new software applications.

Model-agnostic techniques, such as SHAP and LIME, were employed to explain the models. Despite the use of SMOTE for data oversampling, the SHAP model demonstrated unbiased predictions and the data re-balancing approach did not affect the interpretability. Regarding feature importance, both the feature selection processes provided interpretations that aligned with expectations. Occasionally, there were slight differences between local predictions and global predictions. This can be attributed to the fact that local predictions consider individual records and certain features may contribute differently to specific outcomes, resulting in slight variations from the global prediction. Notably, it was possible to predict defects using only five attributes in one of the datasets.

In the future, there is room for research to be conducted on applying PCA and other methods to tackle multicollinearity issues and to consider their impact on interpretability.

In conclusion, it was possible to develop a defect prediction model without bias. It appears that both source code and effort metrics can be important for defect prediction.

Author Contributions: Conceptualization, S.H. and L.F.C.; methodology, S.H. and L.F.C.; software, S.H. and L.F.C.; writing, reviewing and editing, S.H. and L.F.C.; supervision, L.F.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Data Availability Statement: The original data used in this study can be accessed from the PROMISE Data Repository at the following URL: <http://promise.site.uottawa.ca/SERepository/datasets-page.html> (accessed on 11 February 2024).

Acknowledgments: The authors would like to thank Mary Pierce, Dean of Faculty of Business, Information Technology and Part-time Studies, Fanshawe College, and Dev Sainani, Associate Dean of School of Information Technology of Fanshawe College, London, Ontario for supporting this research work.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LIME	Local Interpretable Model-Agnostic Explanations
SHAP	SHapley Additive Explanations
SDP	Software Defect Prediction

References

1. Punitha, K.; Chitra, S. Software defect prediction using software metrics—A survey. In Proceedings of the 2013 International Conference on Information Communication and Embedded Systems (ICICES), Chennai, India, 21–22 February 2013; pp. 555–558. [\[CrossRef\]](#)
2. Shepperd, M.; Song, Q.; Sun, Z.; Mair, C. Data quality: Some comments on the nasa software defect datasets. *IEEE Trans. Softw. Eng.* **2013**, *39*, 1208–1215. [\[CrossRef\]](#)
3. Li, Z.; Jing, X.Y.; Zhu, X. Progress on approaches to software defect prediction. *Int. J. Softw. Eng.* **2018**, *12*, 161–175. [\[CrossRef\]](#)
4. He, P.; Li, B.; Liu, X.; Chen, J.; Ma, Y. An empirical study on software defect prediction with a simplified metric set. *Inf. Softw. Technol.* **2015**, *59*, 170–190. [\[CrossRef\]](#)
5. Balogun, A.O.; Basri, S.; Mahamad, S.; Abdulkadir, S.J.; Capretz, L.F.; Imam, A.A.; Almomani, M.A.; Adeyemo, V.E.; Kumar, G. Empirical analysis of rank aggregation-based multi-filter feature selection methods in software defect prediction. *Electronics* **2021**, *10*, 179. [\[CrossRef\]](#)
6. Ghotra, B.; McIntosh, S.; Hassan, A.E. A large-scale study of the impact of feature selection techniques on defect classification models. In Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, Argentina, 20–21 May 2017; pp. 146–157.
7. Haldar, S.; Capretz, L.F. Explainable Software Defect Prediction from Cross Company Project Metrics using Machine Learning. In Proceedings of the 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 17–19 May 2023; pp. 150–157.
8. Aleem, S.; Capretz, L.; Ahmed, F. Benchmarking Machine Learning Techniques for Software Defect Detection. *Int. J. Softw. Eng. Appl.* **2015**, *6*, 11–23. [\[CrossRef\]](#)
9. Aydin, Z.B.G.; Samli, R. Performance Evaluation of Some Machine Learning Algorithms in NASA Defect Prediction Data Sets. In Proceedings of the 2020 5th International Conference on Computer Science and Engineering (UBMK), Diyarbakir, Turkey, 9–11 September 2020; pp. 1–3.
10. Menzies, T.; Greenwald, J.; Frank, A. Data mining static code attributes to learn defect predictors. *IEEE Trans. Softw. Eng.* **2007**, *33*, 2–13. [\[CrossRef\]](#)
11. Nassif, A.B.; Ho, D.; Capretz, L.F. Regression model for software effort estimation based on the use case point method. In Proceedings of the 2011 International Conference on Computer and Software Modeling, Singapore, 16–18 September 2011; Volume 14, pp. 106–110.
12. Goyal, S. Effective software defect prediction using support vector machines (SVMs). *Int. J. Syst. Assur. Eng. Manag.* **2022**, *13*, 681–696. [\[CrossRef\]](#)
13. Ryu, D.; Jang, J.I.; Baik, J. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. *J. Comput. Sci. Technol.* **2015**, *30*, 969–980. [\[CrossRef\]](#)
14. Thapa, S.; Alsadoon, A.; Prasad, P.; Al-Dala'in, T.; Rashid, T.A. Software Defect Prediction Using Atomic Rule Mining and Random Forest. In Proceedings of the 2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA), Sydney, Australia, 25–27 November 2020; pp. 1–8. [\[CrossRef\]](#)
15. Jayanthi, R.; Florence, L. Software defect prediction techniques using metrics based on neural network classifier. *Clust. Comput.* **2019**, *22*, 77–88. [\[CrossRef\]](#)
16. Fan, G.; Diao, X.; Yu, H.; Yang, K.; Chen, L. Software defect prediction via attention-based recurrent neural network. *Sci. Program.* **2019**, *2019*, 6230953. [\[CrossRef\]](#)
17. Tang, Y.; Dai, Q.; Yang, M.; Du, T.; Chen, L. Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm. *Int. J. Mach. Learn. Cybern.* **2023**, *14*, 1967–1987. [\[CrossRef\]](#)
18. Balasubramaniam, S.; Gollagi, S.G. Software defect prediction via optimal trained convolutional neural network. *Adv. Eng. Softw.* **2022**, *169*, 103138. [\[CrossRef\]](#)
19. Bai, J.; Jia, J.; Capretz, L.F. A three-stage transfer learning framework for multi-source cross-project software defect prediction. *Inf. Softw. Technol.* **2022**, *150*, 106985. [\[CrossRef\]](#)
20. Cao, Q.; Sun, Q.; Cao, Q.; Tan, H. Software defect prediction via transfer learning based neural network. In Proceedings of the 2015 First International Conference on Reliability Systems Engineering (ICRSE), Beijing, China, 21–23 October 2015; pp. 1–10.
21. Joon, A.; Kumar Tyagi, R.; Kumar, K. Noise Filtering and Imbalance Class Distribution Removal for Optimizing Software Fault Prediction using Best Software Metrics Suite. In Proceedings of the 2020 5th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 10–12 June 2020; pp. 1381–1389. [\[CrossRef\]](#)
22. Aggarwal, C.C.; Aggarwal, C.C. *An Introduction to Outlier Analysis*; Springer: Cham, Switzerland, 2017.
23. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [\[CrossRef\]](#)
24. Balogun, A.; Basri, S.; Jadid Abdulkadir, S.; Adeyemo, V.; Abubakar Imam, A.; Bajeh, A. Software Defect Prediction: Analysis Of Class Imbalance and Performance Stability. *J. Eng. Sci. Technol.* **2019**, *14*, 3294–3308.
25. Pelayo, L.; Dick, S. Applying novel resampling strategies to software defect prediction. In Proceedings of the NAFIPS 2007—2007 Annual Meeting of the North American Fuzzy Information Processing Society, San Diego, CA, USA, 24–27 June 2007; pp. 69–72. [\[CrossRef\]](#)

26. Dipa, W.A.; Sunindyo, W.D. Software Defect Prediction Using SMOTE and Artificial Neural Network. In Proceedings of the 2021 International Conference on Data and Software Engineering (ICoDSE), Bandung, Indonesia, 3–4 November 2021; pp. 1–4. [\[CrossRef\]](#)
27. Yedida, R.; Menzies, T. On the value of oversampling for deep learning in software defect prediction. *IEEE Trans. Softw. Eng.* **2021**, *48*, 3103–3116. [\[CrossRef\]](#)
28. Chen, D.; Chen, X.; Li, H.; Xie, J.; Mu, Y. Deepcpdp: Deep learning based cross-project defect prediction. *IEEE Access* **2019**, *7*, 184832–184848. [\[CrossRef\]](#)
29. Altland, H.W. Regression analysis: Statistical modeling of a response variable. *Technometrics* **1999**, *41*, 367–368. [\[CrossRef\]](#)
30. Yang, X.; Wen, W. Ridge and Lasso Regression Models for Cross-Version Defect Prediction. *IEEE Trans. Reliab.* **2018**, *67*, 885–896. [\[CrossRef\]](#)
31. Gezici, B.; Tarhan, A.K. Explainable AI for Software Defect Prediction with Gradient Boosting Classifier. In Proceedings of the 2022 7th International Conference on Computer Science and Engineering (UBMK), Diyarbakir, Turkey, 14–16 September 2022; pp. 1–6.
32. Jiarpakdee, J.; Tantithamthavorn, C.K.; Grundy, J. Practitioners’ Perceptions of the Goals and Visual Explanations of Defect Prediction Models. In Proceedings of the 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), Madrid, Spain, 17–19 May 2021; pp. 432–443. [\[CrossRef\]](#)
33. Sayyad Shirabad, J.; Menzies, T.J. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. Available online: <http://promise.site.uottawa.ca/SERepository> (accessed on 11 February 2024).
34. Gray, D.; Bowes, D.; Davey, N.; Sun, Y.; Christianson, B. The misuse of the NASA metrics data program data sets for automated software defect prediction. In Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), Durham, UK, 11–12 April 2011; pp. 96–103. [\[CrossRef\]](#)
35. Li, J.; Cheng, K.; Wang, S.; Morstatter, F.; Trevino, R.P.; Tang, J.; Liu, H. Feature Selection: A Data Perspective. *ACM Comput. Surv.* **2017**, *50*, 94. [\[CrossRef\]](#)
36. Rahman Khan Mamun, M.M.; Alouani, A. Arrhythmia Classification Using Hybrid Feature Selection Approach and Ensemble Learning Technique. In Proceedings of the 2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), Virtual Event, ON, Canada, 12–17 September 2021; pp. 1–6. [\[CrossRef\]](#)
37. Rosati, S.; Gianfreda, C.M.; Balestra, G.; Martincich, L.; Giannini, V.; Regge, D. Correlation based Feature Selection impact on the classification of breast cancer patients response to neoadjuvant chemotherapy. In Proceedings of the 2018 IEEE International Symposium on Medical Measurements and Applications (MeMeA), Rome, Italy, 11–13 June 2018; pp. 1–5. [\[CrossRef\]](#)
38. Abualigah, L.; Dulaimi, A.J. A novel feature selection method for data mining tasks using hybrid sine cosine algorithm and genetic algorithm. *Clust. Comput.* **2021**, *24*, 2161–2176. [\[CrossRef\]](#)
39. Chandrashekar, G.; Sahin, F. A survey on feature selection methods. *Comput. Electr. Eng.* **2014**, *40*, 16–28. [\[CrossRef\]](#)
40. Thant, M.W.; Aung, N.T.T. Software defect prediction using hybrid approach. In Proceedings of the 2019 International Conference on Advanced Information Technologies (ICAIT), Yangon, Myanmar, 6–7 November 2019; pp. 262–267. [\[CrossRef\]](#)
41. Rajnish, K.; Bhattacharjee, V.; Chandrabanshi, V. Applying Cognitive and Neural Network Approach over Control Flow Graph for Software Defect Prediction. In Proceedings of the 2021 Thirteenth International Conference on Contemporary Computing (IC3-2021), Noida, India, 5–7 August 2021; pp. 13–17.
42. Jindal, R.; Malhotra, R.; Jain, A. Software defect prediction using neural networks. In Proceedings of the 3rd International Conference on Reliability, Infocom Technologies and Optimization, Noida, India, 8–10 October 2014; pp. 1–6.
43. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
44. Rana, G.; Haq, E.u.; Bhatia, E.; Katarya, R. A Study of Hyper-Parameter Tuning in The Field of Software Analytics. In Proceedings of the 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 5–7 November 2020; pp. 455–459. [\[CrossRef\]](#)
45. Osman, H.; Ghafari, M.; Nierstrasz, O. Hyperparameter optimization to improve bug prediction accuracy. In Proceedings of the 2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTesQuE), Klagenfurt, Austria, 21–21 February 2017; pp. 33–38.
46. Shan, C.; Chen, B.; Hu, C.; Xue, J.; Li, N. Software defect prediction model based on LLE and SVM. *IET Conf. Publ.* **2014**, *2014*, CP 653. [\[CrossRef\]](#)
47. Cover, T.; Hart, P. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* **1967**, *13*, 21–27. [\[CrossRef\]](#)
48. Nasser, A.B.; Ghanem, W.; Abdul-Qawy, A.S.H.; Ali, M.A.H.; Saad, A.M.; Ghaleb, S.A.A.; Alduais, N. A Robust Tuned K-Nearest Neighbours Classifier for Software Defect Prediction. In Proceedings of the 2nd International Conference on Emerging Technologies and Intelligent Systems, Sanya, China, 20–22 January 2022; Al-Sharafi, M.A., Al-Emran, M., Al-Kabi, M.N., Shaalan, K., Eds.; Springer: Cham, Switzerland, 2023; pp. 181–193.
49. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [\[CrossRef\]](#)
50. Soe, Y.N.; Santosa, P.I.; Hartanto, R. Software defect prediction using random forest algorithm. In Proceedings of the 2018 12th South East Asian Technical University Consortium (SEATUC), Yogyakarta, Indonesia, 12–13 March 2018; Volume 1, pp. 1–5. [\[CrossRef\]](#)

51. Ribeiro, M.T.; Singh, S.; Guestrin, C. “Why should i trust you?” Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 1135–1144. [[CrossRef](#)]
52. Adadi, A.; Berrada, M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access* **2018**, *6*, 52138–52160. [[CrossRef](#)]
53. Biecek, P.; Burzykowski, T. Local interpretable model-agnostic explanations (LIME). In *Explanatory Model Analysis*; Chapman and Hall/CRC: New York, NY, USA, 2021; pp. 107–123. [[CrossRef](#)]
54. Jiarpakdee, J.; Tantithamthavorn, C.K.; Dam, H.K.; Grundy, J. An empirical study of model-agnostic techniques for defect prediction models. *IEEE Trans. Softw. Eng.* **2020**, *48*, 166–185. [[CrossRef](#)]
55. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*; Neural Information Processing Systems Foundation, Inc.: La Jolla, CA, USA, 2017.
56. Esteves, G.; Figueiredo, E.; Veloso, A.; Viggiato, M.; Ziviani, N. Understanding machine learning software defect predictions. *Autom. Softw. Eng.* **2020**, *27*, 369–392. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.