Interpretable Software Defect Prediction from Project Effort and Static Code Metrics

Susmita Haldar * and Luiz Fernando Capretz

Posted Date: 15 January 2024

doi: 10.20944/preprints202401.1133.v1

Keywords: Defect Prediction, Explainable Machine Learning, Software Quality, Interpretability, Cross-Project Defect Prediction

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, Canada
[2] Department of Electrical and Computer Engineering, Western University, London, Canada; lcapretz@uwo.ca
* Correspondence: shaldar@fanshawec.ca

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 will also assess 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 has been 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

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 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 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. This will help test leads with choosing relevant static code metrics obtained from previous projects of a similar nature for facilitating testing arrangements for new applications.

 doi:10.20944/preprints202401.1133.v1

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?

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 result analysis and discussion. Threats to the validity of our work are presented in Section 6. Finally, the conclusion, and future work are described in Section 7.

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], k-nearest neighbor algorithm [13], random forest algorithms [14], deep learning methods incorporating artificial neural networks [15], recurrent neural network [16] and convolutional neural networks [17], ensemble methods [18], and 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 Inter-Quartile Range (IQR) [20,21]. Outlier is 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 Under Sampling (RUS), Random Over Sampling (ROS), and 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 has been 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, and 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 overcoming the difficulties associated with imbalanced classes.

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.

 doi:10.20944/preprints202401.1133.v1

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 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] revealed 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 for interpretability.

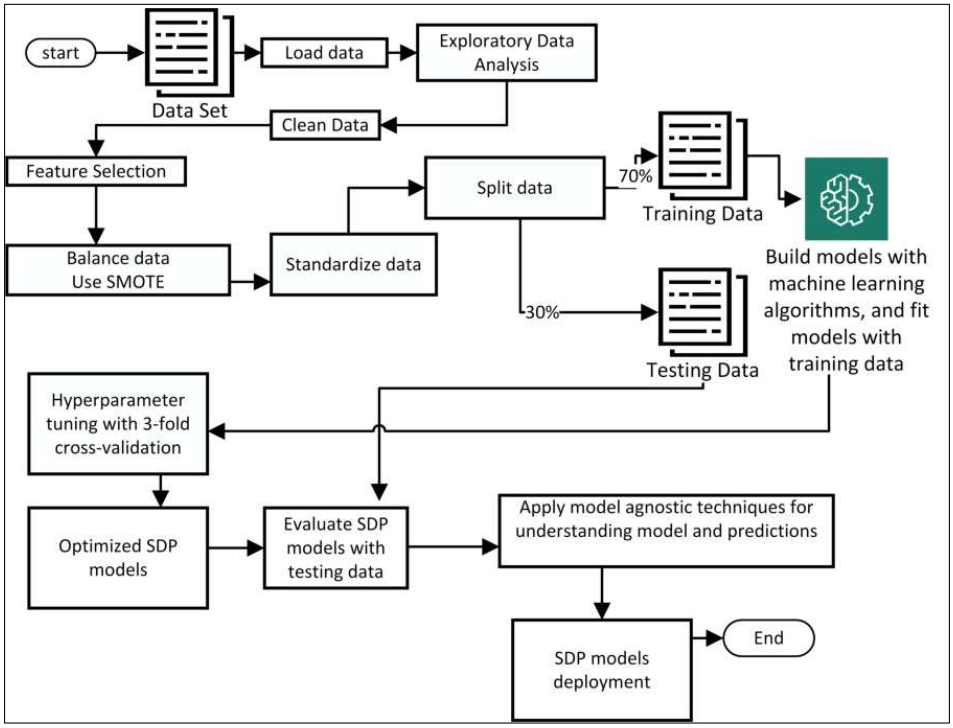


Figure 1. Methodology of developing interpretable SDP models.

3.1. Data Collection

We have 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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 January 2024 doi:10.20944/preprints202401.1133.v1

projects list modules from various programs written in C or C++ programming languages. Each of these selected files contains 21 independent variables and 1 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 attribute available in Table 2.

Table 1. PROMISE Dataset.

File name	No of observation	Programming language	Predictors	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	10885	C	21	19.3%

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

Feature name	Description	Type
loc	McCabe's line count of code	float64
l	Halstead program length	float64
e	Halstead effort	float64
loComment	Halstead's count of lines of comments	float64
v(g)	McCabe's cyclomatic complexity	float64
ev(g)	McCabe's essential complexity	float64
iv(g)	McCabe's design complexity	float64
n	Halstead total operators + operands	float64
v	Halstead volume	float64
d	Halstead program difficulty	float64
i	Halstead intelligence	float64
b	Number of delivered bugs	float64
t	Halstead's time estimator	float64
loCode	Halstead's line count	float64
loBlank	Halstead's count of blank lines	float64
locCodeAndComment	Line of code and comment	float64
uniq_Op	Unique operators	float64
uniq_Opnd	Unique operands	float64
total_Op	Total operators	float64
total_Opnd	Total operands	float64
branchCount	Total branches in program	float64
Defects	{False, true} defective/non-defective module	Category

3.2. Feature Selection and Preprocessing Steps

The process began with loading the stored data from the PROMISE repository. Subsequently, the exploratory data analysis process was conducted, and 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.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 January 2024 doi:10.20944/preprints202401.1133.v1

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.

Two different approaches were employed during the feature selection process in this study. The first approach cleaned the dataset by removing records with missing variables, duplicated entries, outliers, and implausible and conflicting feature values. Additionally, features with a correlation value higher than 70% were dropped. 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. 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 develop SDP models, data needs 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 80:20, 70:30 split, and cross-validation. In several existing defect prediction studies [34–36], 70:30 splits were considered. In this work, each dataset has been 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 [37]. 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 Over-Sampling 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.

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 [38]. 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 1st and outside the range of 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.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 January 2024 doi:10.20944/preprints202401.1133.v1

The distribution of samples with defective vs. non-defective modules in the finalized training dataset is presented in Table 4. For example, the kc2 dataset initially had 21 independent features and 522 records. After cleaning the data, the number of records was reduced to 198. These records were then split between the testing and training datasets. In the training dataset, there were 30 observations classified as defective and 108 as non-defective. Subsequently, SMOTE allowed the data to have an equal distribution of 108 records for both defective and non-defective modules.

Table 4. Distribution of defective vs. non-defective samples before and after applying SMOTE in the training dataset.

Project name	Features	Total samples	Cleaned samples	Regular	SMOTE
kc2	**5**	**522**	**198**	**30, 108**	**108,108**
cm1	**7**	**498**	**289**	**20 ,182**	**182, 182**
pc1	**7**	**1109**	**601**	**23, 397**	**23, 397**
kc1	**6**	**2109**	**718**	**116, 386**	**386, 386**
jm1	**5**	**10885**	**4574**	**509, 2692**	**2692, 2692**
CP	**9**	**15123**	**4608**	**491, 2734**	**2734, 2734**

3.3. Applied Machine Learning Algorithms

After 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 network (ANN). To improve the performance of each of these models, hyperparameter tuning was performed using the grid search method [37] with 3-fold cross-validation on the training dataset. 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)

SVM is a supervised ML algorithm that defines a hyperplane to separate data most optimally, ensuring a wide margin between the hyperplane and the observations. SVM models are known to perform well with imbalanced datasets exhibit efficiency with small datasets and have been widely utilized in various SDP studies [8,39].

K-Nearest-Neighbors (KNN)

KNN is a supervised ML algorithm that utilizes proximity to classify or predict the grouping of individual data points. KNN has been extensively used in SDP problems [40,41] due to its efficiency.

Random Forest Classifier (RF)

Random Forest Classifier 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 [42,43]

Artificial Neural Network (ANN)

The artificial neural network [15] is 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 done in the output layer.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 January 2024 doi:10.20944/preprints202401.1133.v1

SHAP and LIME, which are available in Python libraries, were used to provide explanations for the models.

LIME (Local Interpretable Model-agnostic Explanations)

LIME [44] builds sparse linear models around each prediction to explain how the black box model works in that local vicinity.

SHAP (SHapley Additive exPlanation) SHAP corresponds to the idea of Shapley values for model feature influence scoring [45]. Shapley value corresponds to the average marginal contribution of a feature value over all possible coalitions [46,47]

.

3.4. Evaluation Strategy

The efficiency of the SDP models was evaluated by accuracy and roc auc score. The selected criteria are widely used in the literature for the evaluation of predictor performance [8].

Accuracy is the percentage of correctly classified results [8]. Accuracy can be calculated using eq. 1.

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

The area under the receiving operating characteristics curve (**ROC AUC score**) [39] is a measure of how well a parameter can distinguish between two classes (defective/non-defective).

"True Positive Rate" (TPR) is the proportion of instances labeled as defective that were correctly predicted, and "False Positive Rate" (FPR) is the proportion of instances labeled as non-defective that are incorrectly predicted as defective. The higher the roc auc score, the better the model's performance at distinguishing between the positive and negative classes.

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

4. Results

The common features among all datasets were e, loc, l, ev(g), and lOComment when the highly correlated features were removed from each dataset. The selected features, along with the project names, are shown in Table 5. It can be observed that the maximum number of 9 features, compared to the original 20 features, was available in the cross-project dataset, while kc2 and jm1 had the minimum number of selected features.

Table 5. List of available features after highly correlated features were removed.

File Name	Selected Features
kc2	loc, ev(g), l, e, lOComment
cm1	loc, ev(g), l, e, lOCode, lOComment, lOBlank
pc1	loc,v(g), ev(g), l , e, lOComment, lOBlank
kc1	loc, ev(g), l,e, lOComment, lOBlank
jm1	loc, l, i, e, lOComment
Cross Project	loc, v(g), ev(g), iv(g), l, i, e, lOComment, project

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 January 2024 doi:10.20944/preprints202401.1133.v1

Figure 2 presents the results obtained from the developed SDP models. The top half of this figure illustrates the accuracy and roc auc score of each of the projects on the selected machine learning algorithms and has been applied in the full-feature datasets except the LocCodeAndComment feature. The bottom half of the figure followed the same strategy except the SDP models were created on the reduced-feature datasets which were obtained after removing the multicollinear features. The row called Avg. calculates the average of the accuracy and roc auc score for each of the projects on SVM, KNN, RF, and ANN models. The Avg. score has been compared with the cross-project(CP) SDP model scores.

The projects are listed in order of sample size. The numbers in bold format show the highest accuracy score, and the roc auc score of each of the projects whereas the highlighted yellow fields represent the model where the roc 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 roc 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 roc auc score is 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, vs. selecting reduced features, 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 independent project scores of 74%. On the other hand, the obtained roc auc score of 56% applied in the same classifier can be treated as relatively poor. Taking into consideration the overall roc auc and accuracy, the ANN model outperformed the other classifiers in the cross-project models.

Projects with 20 common features								
Proj. Name	**SVM**		**KNN**		**RF**		**ANN**	
	Acc.	**Roc auc.**	**Acc.**	**Roc auc.**	**Acc.**	**Roc auc.**	**Acc.**	**Roc auc.**
kc2	0.63	0.56	**0.73**	**0.69**	0.67	0.6	0.65	0.61
cm1	0.84	0.52	0.64	**0.68**	0.82	0.51	**0.89**	0.61
pc1	**0.96**	0.7	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**	**0.6**	**0.77**	0.59	0.69	**0.6**
Avg.	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 selected features (reduced features)								
Proj. Name	**SVM**		**KNN**		**RF**		**ANN**	
	Acc.	**Roc auc.**	**Acc.**	**Roc auc.**	**Acc.**	**Roc auc.**	**Acc.**	**Roc auc.**
kc2	**0.75**	0.68	0.73	**0.71**	0.7	0.64	0.73	0.67
cm1	0.56	0.5	0.7	**0.71**	0.9	0.68	**0.83**	0.58
pc1	0.9	0.67	0.85	**0.71**	0.9	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
Avg.	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	**0.8**	0.54	0.7	**0.61**

Figure 2. 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.

 doi:10.20944/preprints202401.1133.v1

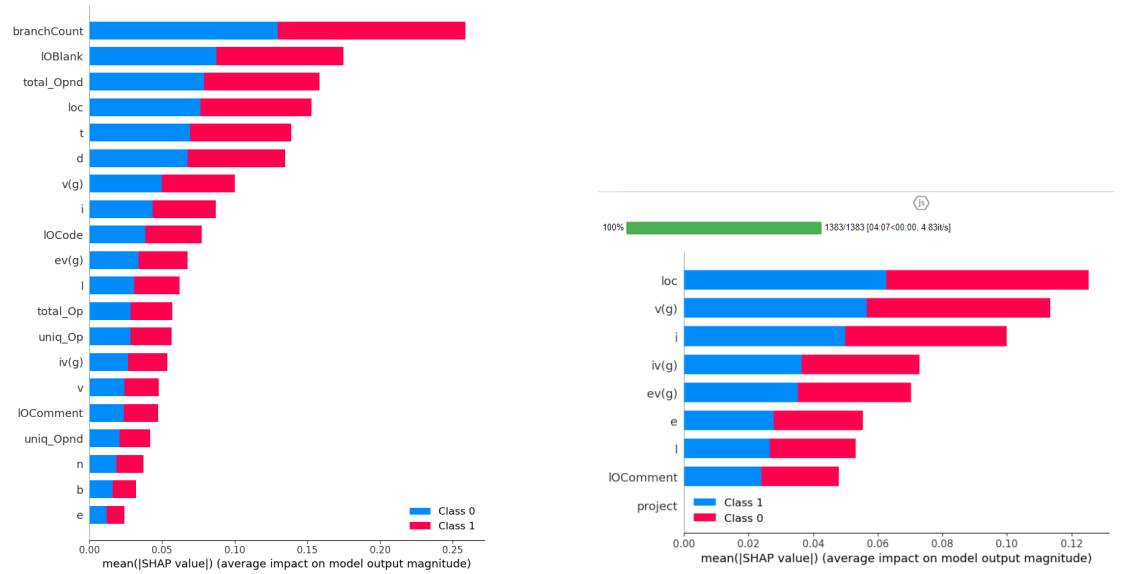
The SVM model performed better on medium-sized dataset namely kc1 project. In terms of cross-project datasets, the ANN model developed on the reduced features achieved a slightly higher roc auc score of 61% compared to 58%. For the pc1 project, which had mid-sized samples, the ANN classifier showed higher accuracy with a value of 95% and 92% for all features, and reduced features dataset respectively. Although, for the pc1 dataset, the same ANN model with all features showed the highest roc auc of 77%, there was a reduction in the roc auc with a value of 68% on the reduced feature dataset.

Figure 3a demonstrates the cross-project defect prediction model developed using SHAP on ANN model incorporating all features. This model has all the features, but the impact of attribute effort "e" is in the lowest position compared to other source code and size metrics. In the same ANN classifier with reduced features, SHAP was applied to have the global explanation on the first 20 observations, as shown in Figure 3b. It reveals that in the SDP model constructed with the ANN classifier, the "loc" attribute has 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 line of code, but less than the 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.

Figure 4 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%. The effort attribute was one of the important contributors in this prediction model, and 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 understand 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 project with mid-sample size was examined in Figure 5 and Figure 6. Both models interpreted with LIME and SHAP respectively, 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 effort metrics. We can 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.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 January 2024 doi:10.20944/preprints202401.1133.v1



(a) SHAP model - dataset contains all features. **(b)** SHAP model - dataset contains reduced features.

Figure 3. SHAP applied on cross-project dataset developed using ANN model.

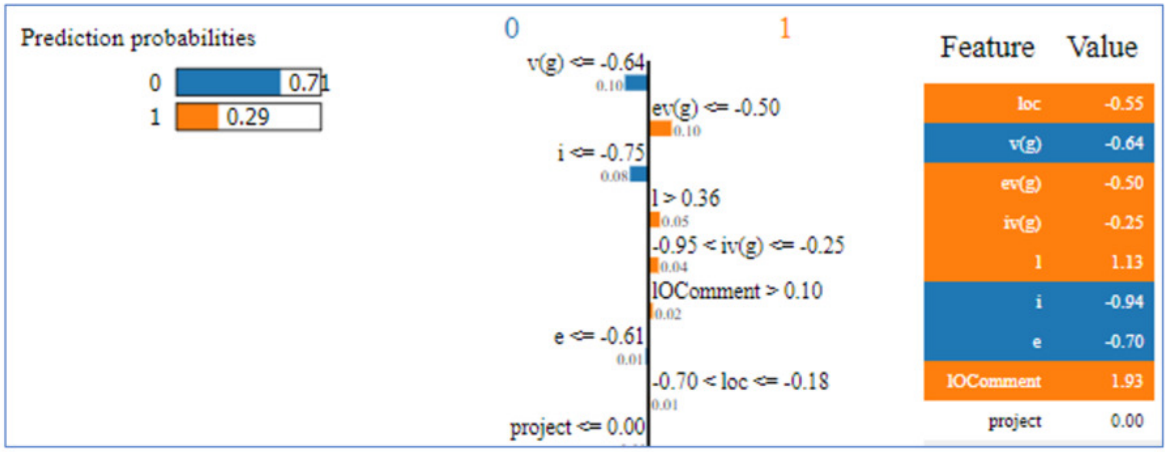


Figure 4. Demonstration of local interpretation of selected record from cross-project data when ANN was applied on reduced-feature based model.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 January 2024 doi:10.20944/preprints202401.1133.v1

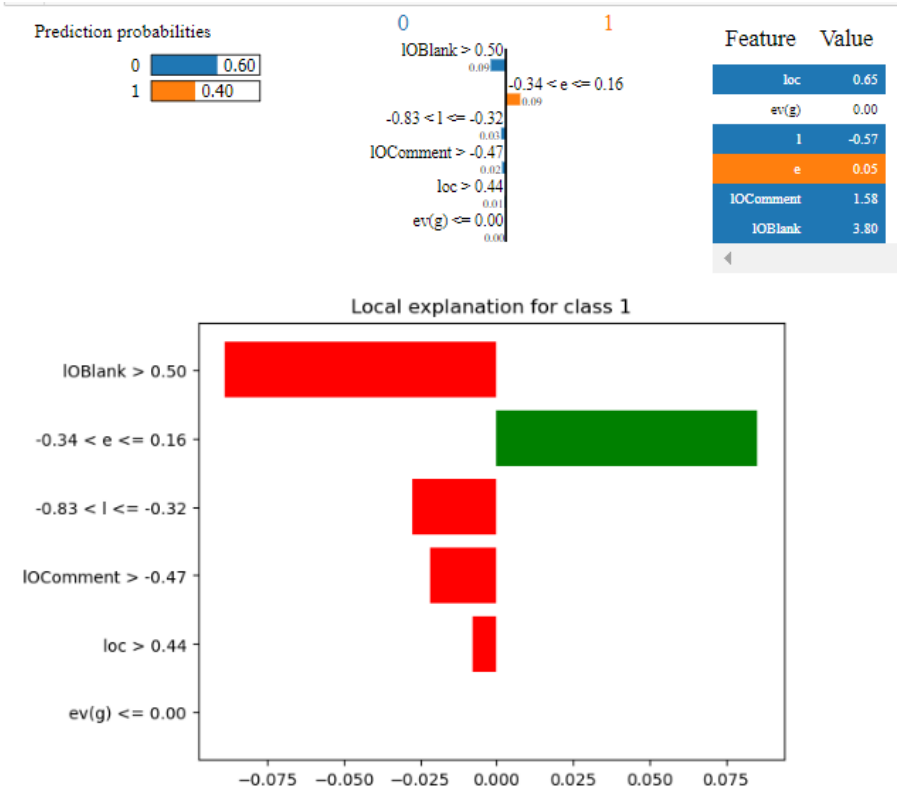


Figure 5. Model interpretation of SDP model generated from kc1 file with LIME on SVM classifier.

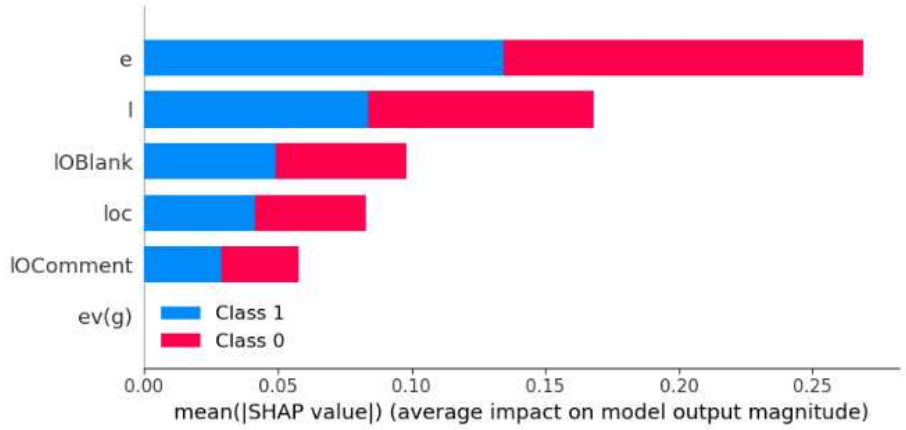


Figure 6. Model interpretation of SDP model generated from kc1 file with SHAP on SVM classifier.

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 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 roc auc, and project pc1 demonstrated the highest performance with an accuracy above 90% and a roc auc score close to 70%. It was observed that the "effort" attribute was not necessarily the most influential of all outcomes. The SHAP and LIME models applied to the developed classifiers showed

 doi:10.20944/preprints202401.1133.v1

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, and it was found that this removal improved the average roc auc score for SVM, KNN, and RF models with a slight decrease in the ANN model. This justifies 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 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 got slightly decreased with reduced features, this was compensated by a relatively higher roc auc score.

To address RQ3, we compared the cross-project (CP) model score with an average score from models built from individual projects. It appears CP models performed slightly lower than the individual projects' average performance. However, the performance did not go down drastically, and they are very close to the average score. For instance, the ANN model with reduced features shows the average accuracy and roc auc score as 76% and 61% whereas the cross-project model had accuracy and roc auc as 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 over-sampling using SMOTE. It was found that both SHAP and LIME successfully explained the predictions, providing insights into the contributing features.

6. Threats to validity

We have utilized the SMOTE technique for creating synthetic data for the minority class. Although this has been proven already, training the model with more balanced data would provide us with reliable models. Also, for comparison, five projects were merged from the same source to see the impact of cross-project defect prediction. This can be extended by bringing data from various sources to bring more variation to the dataset.

7. Conclusion 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. The 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 is still yielding 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 roc 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 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.

 doi:10.20944/preprints202401.1133.v1

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

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

Author Contributions: Conceptualization, S.H.and L.F.; methodology, S.H., and L.F.; software, S.H., and L.F.; writing, reviewing and editing, S.H., L.F.; supervision, L.F.. 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.

Acknowledgments: The authors would like to thank Mrs. Mary Pierce, Dean of Faculty of Business, Information Technology and Part-time Studies, Fanshawe College, and Dr. 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:

MDPI	Multidisciplinary Digital Publishing Institute
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. 2013 International Conference on Information Communication and Embedded Systems (ICICES). IEEE, 2013, pp. 555–558. doi:10.1109/ICICES.2013.6508369.
2. Shepperd, M.; Song, Q.; Sun, Z.; Mair, C. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering* **2013**, *39*, 1208–1215.
3. Li, Z.; Jing, X.Y.; Zhu, X. Progress on approaches to software defect prediction. *Iet Software* **2018**, *12*, 161–175.
4. He, P.; Li, B.; Liu, X.; Chen, J.; Ma, Y. An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology* **2015**, *59*, 170–190.
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.
6. Ghotra, B.; McIntosh, S.; Hassan, A.E. A large-scale study of the impact of feature selection techniques on defect classification models. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017, pp. 146–157.
7. Haldar, S.; Capretz, L.F. Explainable Software Defect Prediction from Cross Company Project Metrics using Machine Learning. 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS). IEEE, 2023, pp. 150–157.
8. Aleem, S.; Capretz, L.; Ahmed, F. Benchmarking Machine Learning Techniques for Software Defect Detection. *International Journal of Software Engineering & Applications* **2015**, *6*, 11–23. doi:10.5121/ijsea.2015.6302.
9. Aydin, Z.B.G.; Samli, R. Performance Evaluation of Some Machine Learning Algorithms in NASA Defect Prediction Data Sets. 2020 5th International Conference on Computer Science and Engineering (UBMK). IEEE, 2020, pp. 1–3.
10. Menzies, T.; Greenwald, J.; Frank, A. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering* **2007**, *33*, 2–13. doi:10.1109/TSE.2007.256941.
11. Nassif, A.B.; Ho, D.; Capretz, L.F. Regression model for software effort estimation based on the use case point method. 2011 International Conference on Computer and Software Modeling, 2011, Vol. 14, pp. 106–110.

12. Goyal, S. Effective software defect prediction using support vector machines (SVMs). *International Journal of System Assurance Engineering and Management* **2022**, *13*, 681–696. doi:10.1007/s13198-021-01326-1.
13. Ryu, D.; Jang, J.I.; Baik, J. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. *Journal of Computer Science and Technology* **2015**, *30*, 969–980. doi:10.1007/s11390-015-1575-5.
14. Thapa, S.; Alsadoon, A.; Prasad, P.; Al-Dala'in, T.; Rashid, T.A. Software Defect Prediction Using Atomic Rule Mining and Random Forest. 2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA), 2020, pp. 1–8. doi:10.1109/CITISIA50690.2020.9371797.
15. Jayanthi, R.; Florence, L. Software defect prediction techniques using metrics based on neural network classifier. *Cluster Computing* **2019**, *22*, 77–88. doi:10.1007/s10586-018-1730-1.
16. Fan, G.; Diao, X.; Yu, H.; Yang, K.; Chen, L.; others. Software defect prediction via attention-based recurrent neural network. *Scientific Programming* **2019**, *2019*, 1—-14.
17. Tang, Y.; Dai, Q.; Yang, M.; Du, T.; Chen, L. Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm. *International Journal of Machine Learning and Cybernetics* **2023**, pp. 1–21.
18. Balasubramaniam, S.; Gollagi, S.G. Software defect prediction via optimal trained convolutional neural network. *Advances in Engineering Software* **2022**, *169*, 103138.
19. Bai, J.; Jia, J.; Capretz, L.F. A three-stage transfer learning framework for multi-source cross-project software defect prediction. *Information and Software Technology* **2022**, *150*, 106985.
20. Cao, Q.; Sun, Q.; Cao, Q.; Tan, H. Software defect prediction via transfer learning based neural network. 2015 First international conference on reliability systems engineering (ICRSE). IEEE, 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. 2020 5th International Conference on Communication and Electronics Systems (ICCES), 2020, pp. 1381–1389. doi:10.1109/ICCES48766.2020.9137899.
22. Aggarwal, C.C.; Aggarwal, C.C. *An introduction to outlier analysis*; Springer, 2017.
23. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* **2002**, *16*, 321–357.
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. *Journal of Engineering Science and Technology* **2019**, *14*, 3294–3308.
25. Pelayo, L.; Dick, S. Applying novel resampling strategies to software defect prediction. NAFIPS 2007-2007 Annual meeting of the North American fuzzy information processing society. IEEE, 2007, pp. 69–72. doi:10.1109/NAFIPS.2007.383813.
26. Dipa, W.A.; Sunindyo, W.D. Software Defect Prediction Using SMOTE and Artificial Neural Network. 2021 International Conference on Data and Software Engineering (ICoDSE), 2021, pp. 1–4. doi:10.1109/ICoDSE53690.2021.9648476.
27. Yedida, R.; Menzies, T. On the value of oversampling for deep learning in software defect prediction. *IEEE Transactions on Software Engineering* **2021**, *48*, 3103–3116. doi:10.1109/TSE.2021.3079841.
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.
29. Altland, H.W. Regression analysis: statistical modeling of a response variable, 1999.
30. Yang, X.; Wen, W. Ridge and Lasso Regression Models for Cross-Version Defect Prediction. *IEEE Transactions on Reliability* **2018**, *67*, 885–896. doi:10.1109/TR.2018.2847353.
31. Gezici, B.; Tarhan, A.K. Explainable AI for Software Defect Prediction with Gradient Boosting Classifier. 2022 7th International Conference on Computer Science and Engineering (UBMK). IEEE, 2022, pp. 1–6.
32. Jiarpakdee, J.; Tantithamthavorn, C.K.; Grundy, J. Practitioners' Perceptions of the Goals and Visual Explanations of Defect Prediction Models. 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), 2021, pp. 432–443. doi:10.1109/MSR52588.2021.00055.
33. Sayyad Shirabad, J.; Menzies, T.J. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering **2005**.

34. Thant, M.W.; Aung, N.T.T. Software defect prediction using hybrid approach. 2019 International Conference on Advanced Information Technologies (ICAIT). IEEE, 2019, pp. 262–267. doi:10.1109/AITC.2019.8921374.
35. Rajnish, K.; Bhattacharjee, V.; Chandrabanshi, V. Applying Cognitive and Neural Network Approach over Control Flow Graph for Software Defect Prediction. 2021 Thirteenth International Conference on Contemporary Computing (IC3-2021), 2021, pp. 13–17.
36. Jindal, R.; Malhotra, R.; Jain, A. Software defect prediction using neural networks. Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization. IEEE, 2014, pp. 1–6.
37. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; others. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* **2011**, *12*, 2825–2830.
38. 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. 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 2011, pp. 96–103. doi:10.1049/ic.2011.0012.
39. Shan, C.; Chen, B.; Hu, C.; Xue, J.; Li, N. Software defect prediction model based on LLE and SVM. *IET Conference Publications* **2014**, *2014*. doi:10.1049/cp.2014.0749.
40. Cover, T.; Hart, P. Nearest neighbor pattern classification. *IEEE transactions on information theory* **1967**, *13*, 21–27. doi:10.1109/TIT.1967.1053964.
41. 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. Proceedings of the 2nd International Conference on Emerging Technologies and Intelligent Systems; Al-Sharafi, M.A.; Al-Emran, M.; Al-Kabi, M.N.; Shaalan, K., Eds.; Springer International Publishing: Cham, 2023; pp. 181–193.
42. Breiman, L. Random forests. *Machine learning* **2001**, *45*, 5–32.
43. Soe, Y.N.; Santosa, P.I.; Hartanto, R. Software defect prediction using random forest algorithm. 2018 12th South East Asian Technical University Consortium (SEATUC). IEEE, 2018, Vol. 1, pp. 1–5. doi:10.1109/SEATUC.2018.8788881.
44. Biecek, P.; Burzykowski, T. Local interpretable model-agnostic explanations (LIME). *Explanatory Model Analysis; Chapman and Hall/CRC: New York, NY, USA* **2021**, pp. 107–123. doi:10.1201/9780429027192-11.
45. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. *Advances in neural information processing systems* **2017**, *30*.
46. Ribeiro, M.T.; Singh, S.; Guestrin, C. " Why should i trust you?" Explaining the predictions of any classifier. Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1135–1144. doi:10.1145/2939672.2939778.
47. Esteves, G.; Figueiredo, E.; Veloso, A.; Viggiato, M.; Ziviani, N. Understanding machine learning software defect predictions. *Automated Software Engineering* **2020**, *27*, 369–392.