The impact of feature importance methods on the interpretation of defect classifiers

Gopi Krishnan Rajbahadur, Shaowei Wang, Gustavo A. Oliva, Yasutaka Kamei, and Ahmed E. Hassan

Abstract—Classifier specific (CS) and classifier agnostic (CA) feature importance methods are widely used (often interchangeably) by prior studies to derive feature importance ranks from a defect classifier. However, different feature importance methods are likely to compute different feature importance ranks even for the same dataset and classifier. Hence such interchangeable use of feature importance methods can lead to conclusion instabilities unless there is a strong agreement among different methods. Therefore, in this paper, we evaluate the agreement between the feature importance ranks associated with the studied classifiers through a case study of 18 software projects and six commonly used classifiers. We find that: 1) The computed feature importance ranks by CA and CS methods do not always strongly agree with each other. 2) The computed feature importance ranks by the studied CA methods exhibit a strong agreement including the features reported at top-1 and top-3 ranks for a given dataset and classifier, while even the commonly used CS methods yield vastly different feature importance ranks. Such findings raise concerns about the stability of conclusions across replicated studies. We further observe that the commonly used defect datasets are rife with feature interactions and these feature interactions impact the computed feature importance ranks of the CS methods (not the CA methods). We demonstrate that removing these feature interactions, even with simple methods like CFS improves agreement between the computed feature importance ranks of CA and CS methods. In light of our findings, we provide guidelines for stakeholders and practitioners when performing model interpretation and directions for future research, e.g., future research is needed to investigate the impact of advanced feature interaction removal methods on computed feature importance ranks of different CS methods.

Index Terms—Model interpretation, Model Agnostic interpretation, Built-in interpretation, Feature Importance Analysis, Variable Importance

1 Introduction

Defect classifiers are widely used by many large software corporations [1–4] and researchers [5–7]. Defect classifiers are commonly interpreted to uncover insights to improve software quality. Such insights help practitioners formulate strategies for effective testing, defect avoidance, and quality assurance [8, 9]. Therefore it is pivotal that these generated insights are reliable.

When interpreting classifiers, prior studies typically employ a feature importance method to compute a ranking of feature importances (a.k.a., feature importance ranks) [8, 10–13]. These feature importance ranks reflect the order in which the studied features contribute to the predictive capability of the studied classifier [14]. These feature importance methods can be divided in two categories: classifier-specific (CS) and classifier-agnostic (CA) methods. A classifier-specific (CS) method makes use of a given classifier's internals to measure the degree to which each feature contributes to a classifier's predictions [15]. We note, however, that a CS method is not always readily available for a given classifier. For example, complex classifiers like SVMs and deep neural networks do not have a widely accepted CS method(s) [16].

For cases such as those, or when a universal way of comparing the feature importance ranks of different classifiers is required [17, 18], classifier-agnostic (CA) methods are typically used. Such CA methods measure the contribution of each feature towards a classifier's predictions. For instance, some CA methods measure the contribution of each feature by effecting changes to that particular feature in the dataset and observing its impact on the outcome. The primary advantage of CA methods is that they can be used for any classifier (i.e., from interpretable to black box classifiers).

Despite computing feature importance ranks using different ways, CS and CA methods are indiscriminately and interchangeably used in software engineering studies (Table 1). For instance, to compute feature importance ranks for a random forest classifier, Treude and Wagner [19] and Yu et al. [20] use CS methods: the Gini importance and the Breiman's importance methods respectively. On the other hand, Mori and Uchihira [13] and Herzig [21] use CA methods: Partial Dependence Plot (PDP) and filterVarImp respectively. Since these methods compute feature importances differently (Section 3.3), different CS or CA methods are likely to compute different feature importance ranks for the same classifier. Yet, we observe that the rationale for choosing a given feature importance method is rarely motivated by prior studies (Section 2.1).

The interchangeable use of feature importance methods is

- *Gopi Krishnan Rajbahadur is with the Centre for Software Excellence, Huawei, Canada.*
  Email:gopi.krishnan.rajbahadur1@huawei.com
- *Gustavo A. Oliva, and Ahmed E. Hassan are with Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Canada.*
  E-mail: {gustavo, ahmed}@cs.queensu.ca
- *Shaowei Wang is with the department of computer science, University of Manitoba, Canada.*
  Email: shaowei@cs.umanitoba.ca
- *Yasutaka Kamei is with Principles of Software Languages (POSL) Lab, Graduate School and Faulty of Information Science and Electrical Engineering, Kyushu University, Japan.*
  Email: kamei@ait.kyushu-u.ac.jp
- *Shaowei Wang is the corresponding author*

acceptable only if the feature importance ranks computed by these methods do not differ from each other. Therefore, in order to determine the extent to which the importance ranks computed by different importance methods agree with each other, we conduct a case study on 18 popularly used software defect datasets using classifiers from six different families. We compute the feature importance ranks using six CS and two CA methods on these datasets and classifiers. The list of CS methods is summarized in Table 6. The two CA methods are: permutation importance (Permutation) and SHapley Additive ExPlanations (SHAP). Finally, we compute Kendall's Tau, Kendall's W, and Top-k ($k \in \{1, 3\}$) overlap to quantify the agreement between the computed feature importance ranks by the different studied feature importance methods for a given classifier and dataset. While Kendall's measures compute differences across the different feature importance ranks, the Top-K overlap measure focuses on the top-k items of these rankings (more details in Section 3.5). We highlight our findings below:

- The computed feature importance ranks by CA and CS methods do not always strongly agree with each other. For two of the five studied classifiers, even the most important feature varies across CA and CS methods.
- The computed feature importance ranks by the studied CA methods exhibit a strong agreement including the features reported at top-1 and top-3 ranks for a given dataset and classifier.
- On a given dataset, even the commonly used CS methods yield vastly different feature importance ranks, including the top-3 and the top-1 most important feature(s).

We then investigate why the agreement between the different CS methods and CA and CS methods remains weak, while the agreement between the computed feature importance ranks by CA methods is strong. Through a simulation study we find that, as hinted by prior studies [22–26], feature interactions present in the studied datasets impact the computed feature importance ranks of CS methods. We then investigate if removal of feature interaction in a given dataset (through a simple method like Correlation-based Feature Selection (CFS) [27, 28]) improves the agreement between the computed feature importance ranks of studied feature importance methods. We find that removal of feature interaction, significantly improves the agreement between the computed feature importance ranks of CA and CS methods. However, the improvement in agreement between the computed feature importance ranks of the studied CS methods remains marginal. In light of these findings, we suggest that future research on defect classification should:

1) Identify (e.g., using methods like Friedman's H-statistic (more details in Section 5.1)) and remove feature interactions present in the dataset (e.g., using simple methods like CFS) before interpreting the classifier, as they hinder the classifier interpretation.

2) One should always specify the used feature importance method to increase the reproducibility of their study and the generalizability of its insights.

Paper Organization. Section 2 presents the motivation of our study and the related work. Section 3 explains how we conducted our case study. In Section 4, we present the results of our case study which examines the extent to which the feature importance ranks computed by different feature importance methods vary. In Section 5, we investigate the impact of feature interactions on the computed feature importance ranks by studied interpretation methods. Section 6 presents the implications of our results and avenues for future research. Section 7 lists the threats to the validity of our study. Finally, Section 8 concludes our study.

2 Motivation and Related Work

In this section, we motivate our study based on how prior studies employed feature importance methods (Section 2.1). Next, we situate our study relative to prior related work (Section 2.2).

2.1 Motivation

We conduct a literature survey of the used feature importance methods in prior studies. To survey the literature, we searched Google Scholar with the terms "software engineering", "variable importance", "feature importance" and the name of each classifier that is studied in our paper (Section 3.4). We searched the Google Scholar multiple times, once for each studied classifier. We eliminated all the papers that were from before the year 2000 to restrict the scope of our survey to recent studies. We read each paper from the search results in order to check if they employed any feature importance method(s) to generate insights. We consider all the studies presented in the google scholar and do not filter based on venues. However, we do not include the papers in which one of the authors of this current study was involved to avoid potential confirmation bias. A summary of our literature survey is shown in Table 1.

We observe that studies rarely specify the reason for choosing their feature importance method – only four out of the 29 surveyed studies provide a rationale for choosing their used feature importance method.

We note that both CA and CS methods are widely used. For instance, from Table 1, we see that both Gini importance and filterVarImp have been used to interpret a random forest classifier. However, given that (i) feature importance methods are typically used to generate insights and (ii) different methods compute the feature importance using different approaches, such interchangeable usage of methods on a given classifier in prior studies is troublesome [56].

For instance, Zimmermann and Nagappan [31] used an F-Test on the coefficients of a logistic regression classifier (a CS method) to show that there exists a strong empirical relationship between Social Network Analysis (SNA) metrics and the defect proneness of a file. Later, Premraj and Herzig [38] used filterVarImp (a CA method) and logistic regression classifier to show that empirical relationship between SNA metrics and the defective files are negligible. Given that CS and CA methods can produce different feature importance ranks, it is unclear whether the aforementioned conflicting result is due to absence of an empirical relationship in the data or simply due to the differing feature importance methods and as such leads to conclusion instability.

More generally, the interchangeable use of feature importance methods (i.e., CS and CA methods), when replicating a study, is acceptable only if the computed ranks by different methods are not vastly different on a given dataset. Otherwise, it raises concerns about the stability of conclusions across the replicated studies. Hence, we investigate the following research question:

(RQ1) For a given classifier, how much do the computed feature importance ranks by CA and CS methods differ across datasets?

Similar concerns exist regarding the interchangeable use of different CA methods, even for the same classifier. From Table 1,

TABLE 1: Different feature importance methods used for interpreting various classifiers in the software engineering literature

Classifier Family	Papers using CS	Used CS methods	Papers using CA	Used CA methods
Statistical Techniques	[29][30]✓[13, 31–36]	Regression coefficients, ANOVA	[10, 21, 37, 38]	Boruta✳, filterVarImp†
Rule-Based Techniques	[39]✓,[40]	Interpreting rules, varImp✇	[37]	Boruta✳
Neural Networks	[41]✖ [42]✓	MODE[42]	[37]	Boruta✳
Decision Trees	[43][44]✓[45]	Decision branches, Gini importance	[10, 21, 37, 38]	Boruta✳, filterVarImp†
Ensemble methods-Bagging	[19, 20, 46] [11]✖ [47, 48] [49]✖ [12, 50, 51] [52]✖	Permutation importance, Gini importance	[10, 13, 21, 37, 38, 53, 54]	Boruta✳, filterVarImp†, PDP, Marks method[55], BestFirst★
Ensemble methods-Boosting	-	-	[10, 21, 37]	Boruta✳, filterVarImp†

✖ - The used method for computing the feature importance ranks is not mentioned
✓ - Papers in which the rationale for choosing a given feature importance method is specified
✳ - https://cran.r-project.org/web/packages/Boruta/index.html
† - https://www.rdocumentation.org/packages/caret/versions/6.0-84/topics/filterVarImp
★ - https://www.rdocumentation.org/packages/FSelector/versions/0.31/topics/best.first.search
✇ - https://www.rdocumentation.org/packages/caret/versions/6.0-84/topics/varImp

we observe that, within each classifier family, different studies use different CA methods. The rationale for choosing a given CA method (for instance, filterVarImp) over another (for instance, PDP) is rarely provided. For instance, none of the studies using a CA method in Table 1 provide reasons for choosing one CA method over another. Yet, the extent to which these CA agree with each other is unclear. Such a concern becomes particularly relevant with the recent rise of complex classifiers for defect prediction [7, 57, 58], as these classifiers do not have a universally agreed-upon or popular CS method. Hence, we investigate the following research question:

(RQ2) For a given dataset and classifier, how much do the computed feature importance ranks by the different CA methods differ?

A number of general prior studies already note that feature importance ranks differ vastly between CS methods [59]. However, such a comparison among CS methods pertaining to different classifier (i.e., CS method associated with a decision tree classifier and a random forest classifier) has not been studied, in the context of defect prediction and software engineering. Such a study is extremely important to understand the limits of reproducibility and generalizability of prior studies. For instance, Jahanshahi et al. [12] replicate the study of McIntosh and Kamei [60] using random forest (and the CS methods of random forest classifier) as opposed to the non-linear logistic regression classifier and its associated CS method. Jahanshahi et al. [12] observe that their feature importance ranks differ from those of the original study. In particular, different CS methods are likely to compute feature importances differently and the difference in insight could be attributed to the used CS method rather than the underlying phenomena (e.g., just-in-time defect prediction) that is being studied. Therefore, we study the following research question along with the previous ones:

(RQ3) On a given dataset, how much do the computed feature importance ranks by different CS methods differ?

2.2 Related Work

As summarized in Section 2.1, both CA and CS methods have been widely used by the software engineering researchers to compute feature importance ranks. In the following, we describe related work regarding (i) usage of feature importance methods in software engineering (ii) the problems associated with widely used feature importance methods and (iii) sensitivity of feature importance methods:

Usage of Feature Importance Methods in Software Engineering. Both CA and CS methods have been widely used by software engineering researchers to compute feature importance ranks. For instance, McIntosh et al. [61] and Morales et al. [34] construct regression models and use ANOVA (a CS method) to understand which aspects of code review impact software quality. In turn, Fan et al. [50] use the CS methods that are associated with the random forest classifier to identify the features that distinguish between merged and abandoned code changes. Similarly, various CS methods that are associated with the random forest classifier have been used to identify features that are important for determining who will leave a company [62], who will become a long time contributor to an open source project [51], code metrics that signal defective code [11], popularity of a mobile app [63], likelihood of an issue being listed in software release notes [64]. Furthermore, CS methods that are associated with logistic regression and decision trees have also been used to generate insights on similar themes [37, 39, 65–68]. Correspondingly, previous studies also use CA methods to interpret classifiers. For example, Tantithamthavorn et al. [18] and Rajbahadur et al. [17] use the permutation CA method to study the impact of data pre-processing on a classifier's feature importance ranks. Furthermore, Dey and Mockus [52] use partial dependence plots (PDP) to identify why certain metrics are not important for predicting the change popularity of an npm package, whereas Mori and Uchihira [13] use PDP to compute the feature importance of random forest classifiers. More recently, Jiarpakdee et al. [69] demonstrated how instance-level CA methods like LIME-HPO (Locally Interpretable Model-Agnostic Explanations with Hyperparameter Optimization) and Breakdown [70] can be used to identify features that are important in determining whether a given module will become defective. They further show that these instance-level feature importances mostly agree with the traditional feature importance ranks.

Problems associated with widely used feature importance methods. Prior studies investigated the potential problems and concerns regarding the widely used feature importance methods. Strobl et al. [59] find that CS methods associated with the widely used random forest classifier are biased and that different CS methods might yield different feature importance ranks when features are correlated. To deal with such correlations, Strobl et al. [71] propose a conditional permutation importance method. Similarly, Candes et al. [72] propose ways to mitigate to potential false discoveries caused by the feature importance method of generalized linear models. Lundberg et al. [26] and Gosiewska and Biecek [70] identify that several popular CA methods produce imprecise feature importance ranks.

Sensitivity of feature importance methods. While the aforementioned studies focus on finding potential problems with existing methods, very few studies compare the existing and widely used feature importance methods. For instance, Grömping [73] compares the feature importance methods of linear regression and logistic regression and identifies both similarities and differences, whereas, Auret and Aldrich [74] compare several tree-based feature importance methods and find that CS method associated with conditional bagged inference trees to be robust.

Jiarpakdee et al. [8] analyze the impact of correlated features on the feature importance ranks of a defect classifier. They find that including correlated features when building a defect classifier results in generation of inconsistent feature importance ranks. In order to avoid that, Jiarpakdee et al. [8] recommend practitioners to remove all the correlated features before building a defect classifier. Through a different study, Jiarpakdee et al. [75] propose an automated method called AutoSpearman that helps practitioners to automatically remove correlated and redundant features from a dataset. They demonstrate that the AutoSpearman method helps one to avoid the harmful impact of these correlated metrics on the computed feature importance ranks of a defect classifier. Similarly, Tantithamthavorn et al. [76] find that noise introduced in the defect datasets due to the mislabelling of defective modules influences the computed feature importance ranks. They show that, among the features reported in the top-3 ranks, the noise introduced by mislabelling does not impact the feature reported at rank 1. However, it influences the features reported at rank 2 and rank 3 across several defect classifiers that they study.

Similarly, several prior studies from Tantithamthavorn et al. [18, 77, 78] investigate how various experimental design choices impact the computed feature importance ranks of a classifier. For instance, Tantithamthavorn et al. [18] investigated whether class rebalancing methods impact the computed feature importance ranks of a classifier. They observe that using class rebalancing methods can introduce *concept drift* in a defect dataset. This concept drift, in turn, impacts the computed feature importance ranks of a defect classifier constructed on the rebalanced defect dataset. Tantithamthavorn et al. [78]also investigate the impact of hyperparameter optimization on the computed feature importance ranks of a classifier. They find that the features reported at the top-3 features importance ranks differ significantly between the hyperparameter tuned classifiers and untuned classifiers. Rajbahadur et al. [17] investigate if the feature importance ranks computed for the regression-based random forest classifiers vary when computed by a CS method and Permutation CA method.

However, to the best of our knowledge, our study is the first work to empirically measure the agreement between the feature importance ranks computed by CA and CS methods across 18 datasets, six classifiers, and 8 feature importance methods, especially in the context of defect prediction. Our study enables the software engineering community to assess the impact of using various feature importance methods interchangeably. As it raises concerns about the stability of conclusions across replicated studies – as such conclusion instabilities might be due primarily to changes in the used feature importance methods instead of characteristics that are inherent in the domain.

3 Case study setup

In this section, we describe the studied datasets (Section 3.1), classifiers (Section 3.2), and feature importance methods (Section 3.3). We then describe our case study approach (Section 3.4), as well as the evaluation metrics that we employ (Section 3.5).

3.1 Studied Datasets

We use the software project datasets from the PROMISE repository [79]. The data set contains the defect data of 101 software projects that are diverse in nature. Use of such varied software projects in our study helps us successfully mitigate the researcher bias identified by Shepperd et al. [80, 81]. In addition, similar to the prior studies by Rajbahadur et al. [17] and Tantithamthavorn et al. [18], we further filter the datasets to study based on two criteria. We remove the datasets with EPV less than 10 and the datasets with defective ratio less than 50. After filtering the 101 datasets from PROMISE with the aforementioned criteria, we end up with 18 datasets for our study: Poi-3.0, Camel-1.2, Xalan-2.5, Xalan-2.6, Eclipse34_debug, Eclipse34_swt, Pde, PC5, Mylyn, Eclipse-2.0, JM1, Eclipse-2.1, Prop-5, Prop-4, Prop-3, Eclipse-3.0, Prop-1, Prop-2. Table 5 in Appendix A.1 shows various basic characteristics about each of the studied datasets.

3.2 Studied Classifiers

We construct classifiers to evaluate our outlined research questions from Section 2.1. We choose the classifiers based on two criteria. First, the classifiers should be representative of the eight commonly used machine learning families in Software Engineering literature as given by Ghotra et al. [82]. The goal behind this criterion is to foster the generalizability and applicability of our results. Second, the chosen classifiers should have a CS method. We only choose classifiers with a CS method, so that we can compare and evaluate the computed feature importance ranks by the CA methods against the CS feature importance ranks for a given classifier (RQ1). In addition, such a choice enables us to compare the agreement between the computed feature importance ranks between different classifiers (RQ3). After the application of these criteria, we eliminate three machine learning families (clustering based classifiers, support vector machines, and nearest neighbour), as the classifiers from those families do not have a CS method. Furthermore, we split the ensemble methods family given by Ghotra et al. [82] into two categories to include classifiers belonging to both bagging and boosting families. The classifiers we finally choose are: Regularized Logistic Regression (glmnet), C5.0 Rule-Based Tree (C5.0Rules), Neural Networks (with model averaging) (avNNet), Recursive partitioning and Regression Trees (rpart), Random Forest (rf), Extreme Gradient Boosting Trees (xgbTree). Table 6 in Appendix A.2 shows the studied classifiers, their hyperparameters and machine learning families to which they belong.

We choose one representative classifier from each of the machine learning family from the **caret**[1] package in R. Table 6

1. https://cran.r-project.org/web/packages/caret/index.html

also shows the caret function that was used to build the classifiers. The selected classifiers have a CS method, that is given by the **varImp()** function in the caret package.

Inherently interpretable classifiers (e.g., fast-and-frugal trees, naive-bayes classifiers and simple decision trees) do not benefit as much from feature importance methods. Hence, such classifiers are out of the scope of this study. Nevertheless, we strongly suggest that the future studies should also explore the reliability of the insights that is derived from simple interpretable classifiers.

3.3 Studied Feature Importance Methods

3.3.1 Classifier Specific Feature Importance (CS) methods

The CS methods typically make use of a given classifier's internals to compute the feature importance scores. These methods are widely used in software engineering to compute feature importance ranks as evidenced from Table 1.

We use six CS methods that are associated with the six classifiers that we study, namely: Logistic Regression FI (LRFI), C5.0 Rule-Based Tree FI (CRFI), Neural Networks (with model averaging) FI (NNFI), Recursive Partitioning and Regression Trees FI (RFI), Random Forest FI (RFFI), and Extreme Gradient Boosting Trees FI (XGFI). Table 7 in Appendix A.3 provides a brief explanation about the inner working of these CS methods on a given classifier. For a more detailed explanation we refer the readers to Kuhn [83].

3.3.2 Classifier Agnostic Feature Importance (CA) methods

CA methods compute the importance of a feature by treating the classifier as a "black-box", i.e., without using any classifier-specific details. In this study, we use the permutation feature importance (Permutation) and SHapley Additive exPlanation (SHAP) CA methods. We use these two CA methods instead of others for the following reasons. First, Permutation is one of the oldest and most popularly used CA methods in both machine learning and software engineering communities [17, 18, 84–88]. It was first introduced by Breiman [89] as way of measuring the feature importance ranks of a random forest classifier and later was adopted as a CA method. Second, we consider SHAP, as it one of the more recent global feature importance method that is theoretically guaranteed to produce optimal feature importance ranks (more details are given below) [90]. Though SHAP was proposed by Lundberg and Lee [91] only in 2017, it has already garnered over 2,000 citations. Furthermore, SHAP is being increasingly adopted in the software engineering community to compute feature importance ranks, as evidenced by its usage in recent studies [92, 93]. Finally, both of these feature importance methods do not require any hyperparameter optimization unlike the CA techniques used by Jiarpakdee et al. [69] and Peng and Menzies [94]. Appendix A.4 describes Permutation and SHAP in more detail.

3.4 Approach

Figure 1 provides an overview of our case study approach. We use this approach to answer all of our aforementioned research questions in Section 2.2.

3.4.1 Data Pre-processing

Correlation and redundancy analysis. We perform correlation and redundancy analysis on the independent features of the studied defect datasets, since the presence of correlated or redundant features impacts the interpretation of a classifier and yields unstable feature importance ranks [8, 95, 96]. Similar to Jiarpakdee et al. [69] we employ the AutoSpearman technique [75] using `AutoSpearman` function from the `Rnalytica` R package to remove the correlated and redundant features from our studied datasets.

3.4.2 Classifier Construction

Out-of-sample bootstrap. To ensure the statistical validity and robustness of our findings, we use an out-of-sample bootstrap method with 100 repetitions to construct the classifiers [77, 97]. More specifically, for each studied dataset, every classifier is trained 100 times on the 100 resampled *train* sets, then these classifiers are used for computing the 100 feature importance scores. The performance of these trained classifiers are also evaluated on the 100 out-of-sample *test* sets. Appendix A.5 describes the out-of-sample bootstrap method in more detail.

Classifier construction with hyperparameter tuning. Several prior studies [18, 98] show that hyperparameter tuning is pivotal to ensure that the trained classifiers fit the data well. Furthermore, Tantithamthavorn et al. [18] show that feature importance ranks shift between hyperparameter tuned and untuned classifiers. Therefore, we tune the hyperparameters for each of the studied classifiers using random search [99] in every bootstrap iteration using `caret` R package [100]. In every iteration, we pass the *train* set, classifier and the associated hyperparameters outlined in Table 6 to `train` function of the `caret` package. similar to Jiarpakdee et al. [69], we then use the automated hyperparameter optimization option provided by train function to conduct a random search (with 10-fold cross-validation) for the optimal hyperparameters that maximizes AUC measure. Once the optimal hyperparameters that maximizes the AUC measure for the given classifier are found, we use these hyperparameters to construct the classifier on the train set. Finally, we use these hyperparameter tuned classifiers to conduct the rest of our study.

3.4.3 Performance Computation

Similar to a recent study by Jiarpakdee et al. [69], we compute the AUC (Area Under the Receiver Operator Characteristic Curve) and the IFA (Initial False Alarm) to measure the performance of our classifiers. We choose these two performance measures in particular over other performance measures for the following reasons. First, several prior studies recommend the use of the AUC measure over other performance measures to quantify the discriminative capability of a classifier [82, 95, 101, 102]. Second, as Parnin and Orso [103] and Huang et al. [104] argue, the IFA of a classifier being low is extremely important for a classifier to be adopted in practice. For these reasons, we choose the AUC and IFA measures to evaluate the performance of our classifiers. In Appendix A.6, we describe provide more details about AUC and IFA, including how they are calculated.

3.4.4 Computation of Feature Importance Scores

We use both the CS and CA methods to computer feature importance scores, as detailed in Section 3.3, for all the studied classifiers in each bootstrap iteration. For CA methods: we use

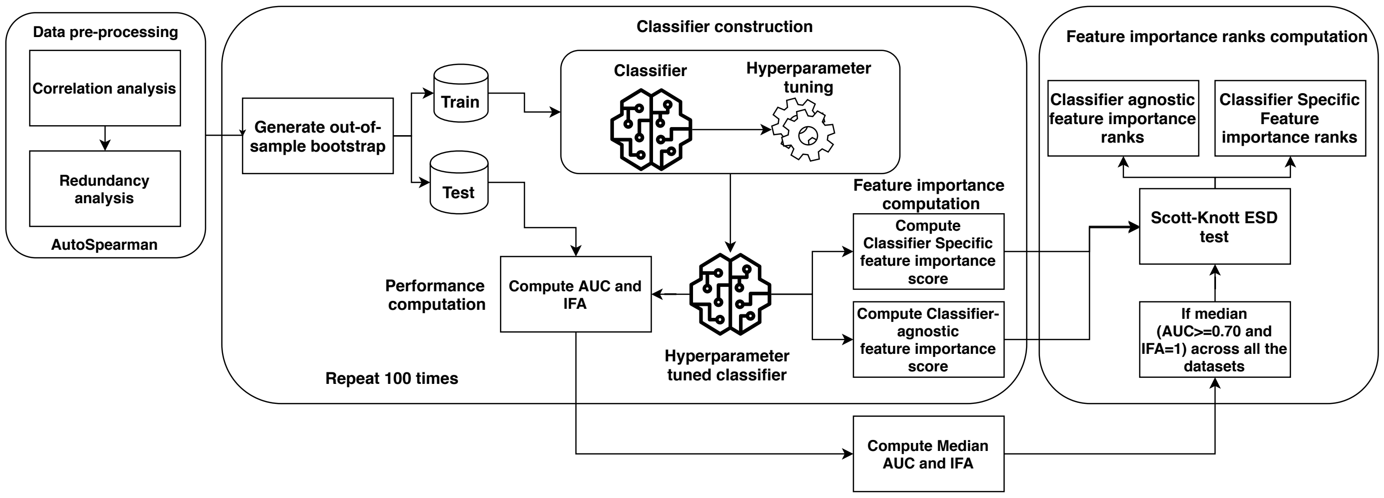
Data pre-processing
Correlation analysis
Redundancy analysis
AutoSpearman
Classifier construction
Generate out-of-sample bootstrap
Train
Test
Classifier
Hyperparameter tuning
Feature importance computation
Compute Classifier Specific feature importance score
Compute Classifier-agnostic feature importance score
Performance computation
Compute AUC and IFA
Hyperparameter tuned classifier
Repeat 100 times
Feature importance ranks computation
Classifier agnostic feature importance ranks
Classifier Specific Feature importance ranks
Scott-Knott ESD test
If median (AUC>=0.70 and IFA=1) across all the datasets
Compute Median AUC and IFA

Fig. 1: Overview of our case study approach.

the vip package and the method outlined by Rajbahadur et al. [17] to compute the PDP and Permutation CA methods feature importance scores respectively. For the CS computation, we use the `VarImp()` function of the caret R package [83].

3.4.5 Computation of Feature Importance Ranks

We use the Scott-Knott Effect Size Difference (SK-ESD) test (v2.0) [105] to compute the feature importance ranks from the feature importance scores computed in the previous step, as done by prior studies [8, 102]. For each dataset and studied classifier, three feature importance scores are computed (one CS score and two CA scores) for each bootstrap iteration. The SK-ESD test is applied on these scores to compute three feature importance rank lists (one CS rank list and two CA rank lists) for all the 6 studied classifiers on each dataset. The process of feature importance rank computation from the feature importance scores is depicted in the right-hand side of Figure 1. Also, we note that we only compute the feature importance ranks for classifiers that simultaneously have a median AUC greater than 0.7 and a median IFA = 1. We do so, as Chen et al. [7] and Lipton [106] argue, a classifier should have a good operational performance for the computed feature importance ranks to be trusted. Due to this constraint, we discarded the classifier C5.0Rules from our studied classifiers.

3.5 Evaluation Metrics

We measure the difference between the different feature importance rank lists by measuring how much they agree with each other.

Kendall's Tau coefficient (τ) [107] is a widely used non-parametric rank correlation statistic that is used to compute the similarity between *two* rank lists [108, 109]. Kendall's τ ranges between -1 to 1, where -1 indicates a perfect disagreement and 1 indicates a perfect agreement. We use the interpretation scheme suggested by Akoglu [110]:

$$\text{Kendall's } \tau \text{ Agreement} = \begin{cases} \text{weak}, & \text{if } |\tau| \leq 0.3 \\ \text{moderate}, & \text{if } 0.3 < |\tau| \leq 0.6 \\ \text{strong} & \text{if } |\tau| > 0.6 \end{cases}$$

Kendall's W coefficient [107] is typically used to measure the extent of agreement among *multiple* rank lists given by different raters (CS methods in our case and raters ≥ 2). The Kendall's W ranges between 0 to 1, where 1 indicates that all classifiers agree perfectly with each other and 0 indicates perfect disagreement. We use the Kendall's W in RQ3 to estimate extent to which the different feature importance ranks that are computed by CS methods agree across all the studied classifiers for a given dataset. We use the same interpretation scheme for Kendall's W as we use for Kendall's Tau.

Top-3 overlap is a simple metric that computes the amount of overlap that exists between features at the top-3 ranks in relation to the total number of features at the top-3 ranks across n feature importance rank lists. This metric does not consider the ordinality of the features in the top-3 ranks, i.e., the order in which a given feature appears in the top-3 ranks. Rather, it only checks if a given feature appeared in all of the top-3 rank lists. Top-3 overlap is adapted from the popular Jaccard Index [111] for measuring similarity. We compute the top-3 overlap among n feature importance lists with the equation 1 (k = 3), where $list_i$ is the ith feature list and n is the total number of lists of features to compare (for in RQ1 and RQ2, n = 2, whereas in RQ3, n = 5).

$$Top - k\ overlap = \frac{\bigcap_{i \geq 2}^{n} Features\ at\ top\ k\ ranks\ of\ list_i}{\bigcup_{i \geq 2}^{n} Features\ at\ top\ k\ ranks\ of\ list_i} \tag{1}$$

We define the interpretation scheme for Top 3 overlap as follows, which aims to enable easier interpretation of the results:

$$\text{Top-3 Agreement} = \begin{cases} \text{negligible}, & \text{if } 0.00 \leq \text{top-3 overlap} \leq 0.25 \\ \text{small}, & \text{if } 0.25 < \text{top-3 overlap} \leq 0.50 \\ \text{medium}, & \text{if } 0.50 < \text{top-3 overlap} \leq 0.75 \\ \text{large} & \text{if } 0.75 < \text{top-3 overlap} \leq 1.00 \end{cases}$$

For example, assume that the top-3 features for CS and CA on a given dataset and classifier are $Imp_{CS}(Top\ 3) = \{cbo, loc, pre\}$ and $Imp_{CA}(Top\ 3) = \{loc, lcom3, dit\}$ respectively. Then the top-3 overlap corresponds to $1/5 = 0.2$ (as $n = 2, k = 3$).

Top-1 overlap is analogous to the Top-3 overlap metric (Equation 1, with k = 1). We define the interpretation scheme for Top-1 overlap as follows: if top-1 overlap is ≤ 0.5 then agreement is low, otherwise agreement is deemed high.

4 Case Study Results

In this section, we detail the results of our case study with regards to our research questions from Section 2.

4.1 (RQ1) For a given classifier, how much do the computed feature importance ranks by CA and CS methods differ across datasets?

Approach: For each of the five constructed classifiers with (median AUC > 0.7 and median IFA ≤ 1 across the studied datasets), we compare the feature importance ranks that are computed by the CA and CS methods across all the studied datasets. For each classifier, on a given dataset, we compare the feature importance ranks computed by SHAP and Permutation CA methods with the feature importance ranks that are computed by the studied CS method of a classifier. We quantify the agreement between the two rank lists in terms of Top-1 overlap, Top-3 overlap and Kendall's Tau. We compute the Top-1 and Top-3 overlap in addition to the Kendall's Tau because some of the prior work primarily examines the top x important features [1, 7, 112]. Finally, we aggregate the comparisons with respect to each classifier across the studied datasets.

For instance, for the avNNet classifier, we first compare the feature importance ranks that are computed by SHAP (CA method) with those that are computed by the CS method of avNNet (i.e. NNFI, see Table 7) on the `eclipse-2.0` dataset. Next, we determine the agreement between the two lists according to Top-1, Top-3 overlap, and Kendall's Tau. We then repeat this step for every dataset and plot the distribution for each agreement metric. An analogous process is followed in order to compare the Permutation method with the NNFI method.

The goal of this RQ is to determine the extent to which the feature importance ranks that are computed by CA methods differ from the more widely used and accepted CS methods for each classifier. If the studied CA methods consistently have a high agreement with the CS methods for each classifier and across all the studied datasets, then one can use both CS methods and CA methods interchangeably.

Results: Result 1) ***The SHAP and Permutation CA methods have a low median top-1 overlap with the CS methods on two of the five studied classifiers.*** The leftmost lane in Figure 2 shows the top-1 overlap between the feature importance rank lists that are computed by the CS and CA methods for each classifier and across all the studied datasets. We observe that the median top-1 overlap between the studied CA methods and the CS method of a classifier is low for two classifiers, namely rpart and avNNet. In other words, even the most important feature varies between the rankings that are computed by the CA and CS methods for two of the studied classifiers.

Result 2) ***Both CA methods have a small median top-3 overlap with CS methods on two of the five studied classifiers.*** We see from the middle lane of Figure 2 that the features that are reported at the top-3 ranks by both the SHAP and Permutation method do not exhibit a large overlap with the feature importance ranks that are computed by the CS method for *any* of the studied classifiers. Furthermore, from Figure 2, we observe that even on cases where the median overlap is medium, the spread of the density plot is also large (i.e., several datasets exhibit small and even negligible top-3 overlap).

Result 3) ***For three out of the five studied classifiers, the Kendall's Tau agreement between CA and CS methods is only moderate at best.*** Kendall's Tau values between the feature importance ranks that are computed by CA and CS methods for each classifier and across all the studied datasets are depicted as density distributions in Figure 2. For both rpart and glmnet, the median Kendall's Tau agreement is weak. Even for avNNet, where the median agreement between the CA and CS methods is moderate, the spread of the density lot is large with several datasets exhibiting negligible agreement. We observe that the median Kendall's Tau indicates a strong agreement for only for two of the six studied classifiers, namely xgbTree and rf (note the vertical bars inside the density plots in the right-most lane).

In summary, the CA and CS methods do not always exhibit strong agreement for the computed feature importance ranks across the studied classifiers. Therefore, we discourage the interchangeable use of CA and CS methods in general. In particular, we suggest that unless agreement between the CA and CS methods can be improved (we present a potential solution in Section 5.2), whenever possible, future defect prediction studies should preferably choose the same feature importance method when replicating or seeking to validate a prior study.

The computed feature importance ranks by CA and CS methods do not always strongly agree with each other. For two of the five studied classifiers, even the most important feature varies across CA and CS methods.

4.2 (RQ2) For a given dataset and classifier, how much do the computed feature importance ranks by the different CA methods differ?

Approach: For each of the studied datasets, we check the extent to which the feature importance ranks computed by SHAP and Permutation CA methods agree with each other for all the five studied classifiers. Similarly to the previous RQ, in order to quantify agreement, we compute the Top-1 Overlap, Top-3 Overlap, and Kendall's Tau measures.

Result 4) ***SHAP and Permutation methods have a high median Top-1 overlap and a strong agreement (in terms of Kendall's Tau) across all the studied datasets.*** Furthermore, SHAP and Permutation CA methods do not have a small or negligible overlap across any of the studied datasets. Except on Prop-4 and PC5 datasets, the feature importance ranks computed by SHAP and Permutation CA methods have a large median Top-3 overlap. Furthermore, from the rightmost lane in Figure 3 we observe that except for the rf classifier on prop-4 dataset, the Kendall's Tau agreement values are consistently strong for all the studied classifiers.

The computed feature importance ranks by the studied CA methods exhibit a strong agreement including the features reported at top-1 and top-3 ranks for a given dataset and classifier.

4.3 (RQ3) On a given dataset, how much do the computed feature importance ranks by different CS methods differ?

Approach: For each of the datasets, we obtain the computed feature importance ranks by the studied CS methods of each of the five studied classifiers. We then calculate the Kendall's W between the six feature importance rank lists that are computed by the studied CS method of each classifier. Unlike the previous

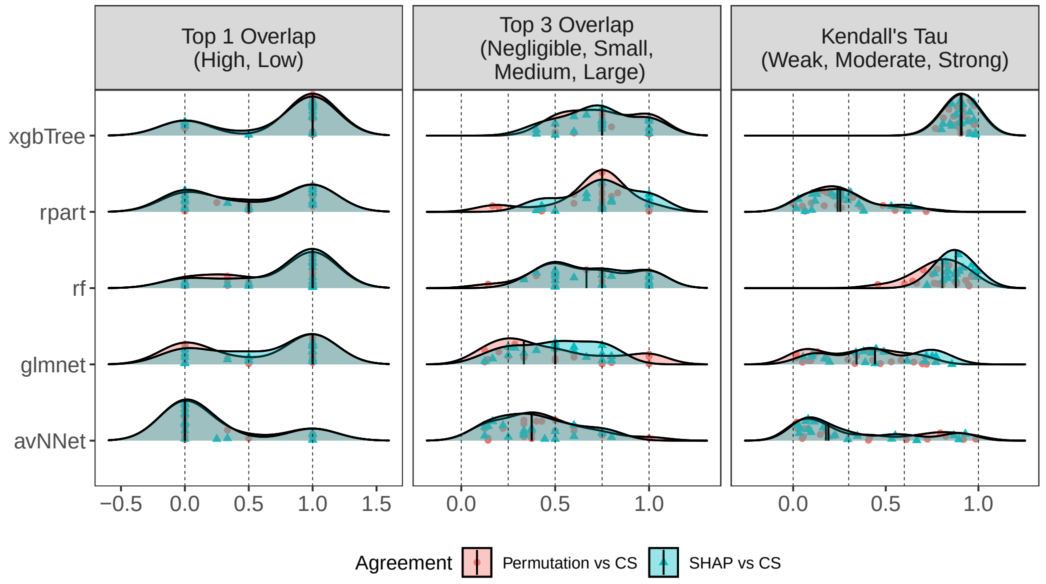

Fig. 2: A density plot of Top-1 Overlap, Top-3 Overlap, and Kendall's Tau between CA and CS methods for each classifier across the studied datasets. The circles and triangles correspond to individual observations. The dotted lines correspond to the metric-specific interpretation scheme outlined in Section 3.5. The vertical lines inside the density plots correspond to the median of the distributions. The vertical lines inside the density plots correspond to the median of the distributions.

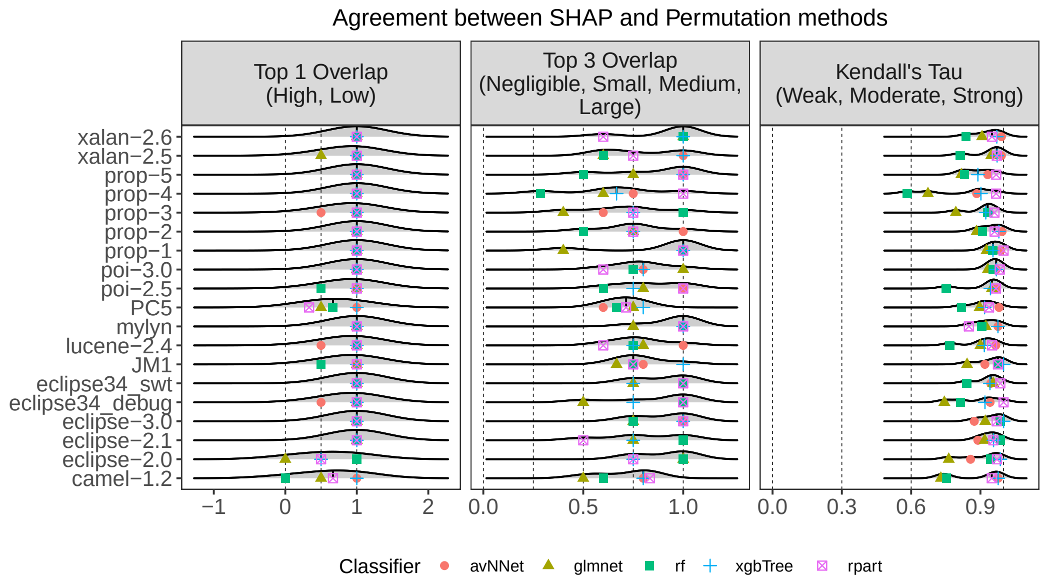

Fig. 3: A density plot of Top-1 Overlap, Top-3 Overlap, and Kendall's Tau values between the SHAP and Permutation CA methods for each of the studied classifiers across all the studied datasets. The dotted lines correspond to the metric-specific interpretation scheme outlined in Section 3.5. The vertical lines inside the density plots correspond to the median of the distributions. Please note the empty space in the rightmost figure is due to the observed range of Kendall's Tau values in this case varying between 0.5 and 1. In other words, across all the studied datasets the computed feature importance ranks of SHAP and Permutation at least had a moderate agreement.

RQs, we compute the Kendall's W instead of Kendall's Tau, as Kendall's W is able to measure agreement among multiple feature importance rank lists (Section 3.5). Furthermore, we also calculate the Top-3 overlap among all the five feature importance rank lists. We do so for all the studied datasets. A high Kendall's W and a high Top-3 overlap across all the studied datasets among the constructed classifiers would indicate high agreement between the computed feature importance ranks by different classifiers and the studied CS method of each classifier.

Results: Result 5) ***The computed feature importance ranks by different CS methods vary extensively.*** None of CS methods agree on the most important feature (as evidenced by the results presented in Table 2). Furthermore, the maximum top-3 overlap is only small and it happens for only three out of 18 datasets. Finally, we also observe that, on a given dataset, only on two occasions the feature importance rank lists computed by the different CS methods strongly agree with each other. We summarize the Top 1 overlap, the Top-3 overlap and Kendall's W among the computed feature importance ranks for all the six studied classifiers across the studied datasets in Table 2.

From Table 2, we observe that both the Kendall's W and top-3 overlap among the feature importance ranks that are computed

TABLE 2: Top-1 overlap, Top-3 overlap, and Kendall's W among the computed feature importance ranks by the CS method of each classifier. Best results for each metric are shown in bold.

Dataset	Top-1 overlap	Top-3 overlap	Kendall's W
poi-3.0	Low (0)	Negligible (0)	Weak (0.13)
camel-1.2	Low (0)	Negligible (0)	Weak (0.22)
xalan-2.5	Low (0)	Negligible (0.10)	Weak (0.18)
xalan-2.6	Low (0)	Negligible (0.16)	Weak (0.25)
eclipse34_debug	Low (0)	Negligible (0.14)	Weak (0.26)
eclipse34_swt	Low (0)	small (0.33)	**Strong (0.62)**
pde	Low (0)	**Small (0.4)**	**Strong (0.70)**
PC5	Low (0)	Negligible (0)	Weak (0.18)
mylyn	Low (0)	**Small (0.29)**	Weak (0.16)
eclipse-2.0	Low (0)	Negligible (0.17)	Weak (0.26)
JM1	Low (0)	Negligible (0.20)	**Moderate (0.31)**
eclipse-2.1	Low (0)	Negligible (0.17)	**Moderate (0.31)**
prop-5	Low (0)	Negligible (0)	Weak (0.17)
prop-4	Low (0)	Negligible (0.14)	**Moderate (0.37)**
prop-3	Low (0)	Negligible (0.09)	**Moderate (0.32)**
eclipse-3.0	Low (0)	**Small (0.28)**	**Moderate (0.32)**
prop-1	Low (0)	Negligible (0.16)	**Moderate (0.36)**
prop-2	Low (0)	Negligible (0)	Weak (0.28)

by studied CS methods associated with each of the classifier – which are widely used in the software engineering community– is very low. Such a small Top-3 overlap and Top-1 overlap for all the datasets indicates that computed feature importance ranks by CS methods differ substantially among themselves. Hence, different classifiers and their associated CS methods cannot be used interchangeably.

On a given dataset, even the commonly used CS methods yield vastly different feature importance ranks, including the top-3 and the top-1 most important feature(s).

5 Discussion

5.1 Why do different feature importance methods produce different top-3 features on a given dataset?

Motivation: From the results presented for RQ1 (Section 4.1) we observe that, on a given dataset and classifier the CA methods and CS methods produce different feature importance ranks (including the top-3 ones). Similarly, from the results presented in RQ3 (Section 4.3), on a given dataset, even the widely used CS methods produce vastly different feature importance ranks. Such a result is in spite of us having removed the correlated and redundant features from the datasets in a pre-processing step using a state-of-the-art technique like AutoSpearman [75]. However, in contrast to the results presented in RQ1 and RQ3, in RQ2 (Section 4.2), we observe that the studied CA methods (i.e., Permutation and SHAP methods) produce similar feature importance ranks on a given dataset and classifier.

We hypothesize that different CS methods produce different top-3 feature importance ranks even on the same dataset and classifier (and when compared to the feature importance ranks computed by the CA methods) because of feature interactions that are present in the studied datasets. Feature interactions can be defined as a phenomenon where the effect of the independent features on a dependent feature is not purely additive [15, 23]). We arrive at such a hypothesis as many of the prior studies show that the presence of feature interactions in a given dataset can affect the different feature importance methods differently and make them assign different feature importance ranks to features [22–26]. Therefore, in this section, we seek to find out if different feature importance methods (in addition to being inherently different) yield different feature importance ranks due to the presence of feature interactions in the dataset.

We further note that computed feature importance ranks by the CA methods exhibit a high agreement and overlap as both Permutation and SHAP are typically not impacted by the feature interactions present in a dataset [15].

Approach: To test our hypothesis and detect if any of the features presented in a given dataset interact with other features in that dataset, we compute the *Friedman H-Statistic* [113] for each feature against all other features. The Friedman H-statistic works as follows. First, a classifier (any classifier - we use random forest as it captures interactions well [114]) is constructed using the given dataset. For instance, consider the Eclipse-2.0 dataset and that we wish to compute the Friedman H-Statistic between the feature `pre` and all the other features. We first compute the partial dependence between `pre` and the dependent variable with respect to the random forest classifier (`PD_pre`) for all the data points. Following which, partial dependence between all the features (as a single block) in Eclipse-2.0 (except `pre`) is computed (`PD_rest`) for all the data points. If there is no feature interaction between `pre` and the other features in Eclipse-2.0, the outcome probability of the constructed classifier can be expressed as a sum of `PD_pre` and `PD_rest`. Therefore, the difference between the outcome probability of the classifier and the sum of `PD_pre` and `PD_rest` is computed and the variance of this difference is reported as the Friedman H-Statistic. We compute the Friedman H-Statistic for all the studied datasets 10 times, as Friedman H-Statistic is known to exhibit fluctuations because of the internal data sampling [15]. We then consider the median score of the Friedman H-Statistic for each dataset. We use the R package `iml`[2] to compute the Friedman H-Statistic.

The Friedman H-Statistic is a numeric score that ranges between 0 and 1. However, it can sometimes exceed 1 when the higher order interactions are stronger than the individual features [15]). A Friedman H-Statistic of 0 or closer to 0 indicates that no interaction exists between the given feature and the rest of the features and a Friedman H-Statistic of 1 indicates extremely high levels of interaction. For a more theoretical and detailed explanation we refer the readers to [15, 113]. In this study, we consider a feature to exhibit interactions with other features if the Friedman H-Statistic is ≥ 0.3. We choose 0.3 as a cut-off, to only indicate the existence of a feature interaction, but not to qualify the strength of the interaction, because the presence of feature interactions irrespective of the strength could potentially impact feature importance ranks [22–26]. In addition we also report the results for the number of features that exhibit a Friedman H-statistic ≥ 0.5. We choose to report the results on multiple thresholds to present a comprehensive depiction of the feature interactions. Furthermore, there is no established guideline in the literature regarding how one should interpret the Friedman H-statistic or how thresholds should be selected.

Construction of a synthetic dataset without any feature interactions. Next, to determine if the absence of feature interactions enables the different CS methods to compute the same top-3 features, we simulate a dataset with no feature interactions. We do so instead of using a real dataset as it is difficult to find a real-world defect dataset without any feature

2. https://cran.r-project.org/web/packages/iml/index.html

interactions. We generate a dataset with 1,500 data points and 11 independent features of which five features carry the signal: `signal = {x1,x2,x3,x4,x5}` and six features are just noise i.e., does not exhibit any relationship to the dependent feature `noise = {n1, n2, n3, n4, n5, n6}`. We add noise features to make our simulated dataset similar to that of a real-world defect dataset. All the signal features and `n1, n5, n6` are generated by randomly sampling the normal distribution with mean = 0 and standard deviation = 1. Similarly, we sample the uniform distribution between 0 and 1 to generate the values for `n2, n3, n4`. We use both the normal and the uniform distributions for generating the noise features to ensure the presence of different types of noise in our simulated dataset. Next, to construct our dependent feature for the dataset, we construct y_{signal} with the signal variables as given in Equation 2. We assign different weights to the different signal features when constructing the y_{signal} to ensure that we know the true importance of each of the signal features. We then convert the y_{signal} in to a probability vector y_{prob} with a sigmoid function as given in Equation 3. Finally, we generate the dependant feature by sampling the binomial distribution to generate the dependent feature $y_{dependant}$ with y_{prob} as given in Equation 4.

$$y_{signal} = 20x1 + 10x2 + 5x3 + 2.5x4 + 0.5x5 \tag{2}$$

$$y_{prob} = \frac{1}{1 + e^{-y_{signal}}} \tag{3}$$

$$y_{dependant} = Binomial(1500, y_{prob}) \tag{4}$$

Construction of classifiers and calculation of top-1 and top-3 overlaps. We then construct all of the studied classifiers on the simulated dataset with $y_{dependent}$ as the dependent feature. We construct all the classifiers with 100-Out-of-sample bootstrap on the simulated dataset and compute the feature importance ranks computed by the CA and CS methods as outlined in Section 3.4. For each of the studied classifiers, we calculate the top-1 and top-3 overlap between the feature importance ranks computed by the CA and CS methods (similarly to RQ1). Furthermore, we also calculate the top-1 and top-3 overlaps between feature importance ranks computed by the studied CS methods (similarly to RQ3). We then check if they exhibit a top-1 and top-3 overlap close to 1 for all the classifiers between the computed feature importance ranks of the CS and the CA methods. In addition we also check the top-1 and top-3 overlap among the computed feature importance ranks of the various CS methods. If in both the cases they exhibit a top-1 and top-3 overlap close to 1, we can then assert that the feature interactions in the dataset affects the top-3 features computed by the different CS methods and vice versa.

Determining whether feature interactions impact interpretation. To verify that the feature interactions present in the dataset impact only the studied CS methods and not the studied CA methods, we compute the feature importance ranks for all the studied classifiers with the studied CS and CA methods on a simulated dataset with feature interactions. To simulate a dataset with interactions, we take the simulated dataset from earlier (the one without any interactions) and introduce interactions by modifying the Equation 2. We modify the $y_{signal\_with\_interactions}$ to depend on hidden interactions as given by Equation 5. The rest of the data generation process remains the same as earlier i.e., the independent features are still given by `independent features = {signal,noise}` where `signal = {x1,x2,x3,x4,x5}` and `noise = {n1, n2, n3, n4, n5, n6}`. Finally, $y_{dependant\_with\_interactions}$ is generated using Equations 3 and 4.

$$y_{signal\_with\_interactions} = y\_signal + x1 * x3 + x2 * x3 + x2 * x1 \tag{5}$$

We then construct all of the studied classifiers with $y_{dependent\_with\_interactions}$ on the simulated dataset with interactions using 100-Out-of-sample bootstrap. Next, we compute the feature importance ranks using the studied CA and CS methods. Finally, we calculate the top-1 and top-3 overlap between the computed feature importance ranks of CA and CS methods, respectively (similarly to RQ2 and RQ3). If feature interactions do not impact the CA methods, we should observe high top-1 and top-3 overlaps (close to 1) between the computed feature importance ranks of the CA methods and vice versa. Similarly, if the feature interactions impact the computed feature importance ranks of CS methods, we should observe low top-1 and top-3 overlaps between the computed feature importance ranks of the different CS methods and vice versa.

Results: Result 6) ***At least two features and as many as eight features interact with the rest of the features in all of the 18 studied datasets (i.e., Friedman H-Statistic*** ≥ 0.3***). Furthermore, we find that 14 of the 18 datasets have at least two features with a Friedman H-Statistic*** $\geq$ ***0.5.*** We present the number of features in each dataset with a Friedman H-statistic $\geq$ 0.3 and $\geq$ 0.5 in Table 8 (Appendix B). From Table 8, we observe that all datasets contain more than two features that interact with the other features. Though Friedman H-Statistic only computes if a given feature interacts with the rest of the features and excludes other feature interactions like second-order interactions, pairwise interactions, and higher-order interactions, it gives us a hint as to the presence or absence of feature interactions in a dataset.

Result 7) ***The top-3 and the top-1 overlap between the feature importance ranks computed by the CS and CA methods on each of the classifiers is 1 on our simulated dataset devoid of feature interactions.*** In addition, the top-3 and top-1 overlap between the computed feature importance ranks of the CS methods on the simulated dataset without interaction is 1. Such a result indicates that in the dataset without feature interactions, all the studied feature importance methods identify the same top-3 features. Furthermore, we also observe that all the studied important features are identified x1, x2, x3 as the top-3 features in the same order of importance. Thus, we assert that the different feature importance methods are able to assign the feature importance ranks correctly when independent features' contribution to the dependent feature is additive without any interactions.

Result 8) ***The top-3 and the top-1 overlap between the feature importance ranks computed by the studied CA methods is 1 on the simulated dataset with interactions.*** In turn, we find that the top-1 and top-3 overlap between the feature importance ranks computed by the different CS methods is 0 on the simulated dataset with interactions. Such a result indicates that studied CA methods (i.e., SHAP and Permutation) are not impacted by the feature interactions in the simulated dataset. However, the computed feature importance ranks of the studied CS methods are heavily influenced by the presence of feature interactions in the dataset.

Hence, we conclude that the presence of feature interactions in the studied defect datasets could be the reason why different CS methods produce a different top-3 set of features. In addition, we conclude that alongside the fact that different feature importance methods compute feature importances differently, feature interactions in the datasets can also be a key confounder that affects the computed feature importance ranks by the studied feature importance methods.

5.2 Can we mitigate the impact of feature interactions?

Motivation: From the results presented in Section 5.1 we observe that feature interactions impact feature importance ranks computed by the CS methods. Such a result indicates that CS methods cannot be interchangeably used. Though comprehensively identifying and removing all the feature interactions from a dataset is still an open area of research, there exist several simple methods like Correlation-based Feature Selection (CFS) [27], wrapper methods [27] that allows us to remove lower order feature interactions [115–117].

Therefore, in this discussion, we investigate if removing the feature interactions present in a dataset through a simple method like CFS would increase the agreement between the computed feature importance ranks of the studied CS methods on a given dataset. We also investigate if removing feature interactions yields an improved agreement between the computed feature importance ranks of CA and CS methods. Finally we also study if the removal of feature interactions has any impact on the agreement between the feature importance ranks computed by the studied CA methods. If it does, then we can recommend researchers and practitioners to remove the feature interactions in their datasets using CFS prior to building the machine learning classifiers.

Approach: We use the CFS method to remove the feature interactions across the studied datasets. We use the CFS method in lieu of other methods like Wrapper in the dataset for the following reasons. First, several prior studies show that, in addition to eliminating correlation between the features, CFS method is also useful for mitigating feature interactions [27, 28]. Second, CFS method has been widely used in the software engineering community (though for removing correlated features) [69]. Third, it is simple to implement and is extremely fast. Finally, since its a filter type method, it does not make any assumptions about the dataset or the subsequent classifier that is to be built.

Therefore, for each of the studied datasets, following the removal of correlated and redundant features using AutoSpearman, we apply the CFS [27] method to remove the feature interactions. The CFS method chooses a subset of features that exhibits the strongest relationship with the dependent feature while exhibiting a minimal correlation among themselves. It is important to note that we apply the CFS method to the features that were not flagged as correlated/redundant by AutoSpearman technique (as opposed to applying CFS method by itself to eliminate both the observed inter-feature correlation and feature interaction), as Jiarpakdee et al. [69] argue CFS method might not remove all the correlated features from the dataset effectively.

After removing the feature interactions using CFS methods, we re-run the analysis that we conducted in RQ1 (Section 4.1) and RQ3 (Section 4.3) using the same approach. We then evaluate whether the agreement in terms of top-1 and top-3 overlap increases between the CA and CS methods (compared to the results presented in Section 4.1). Similarly, we also evaluate whether the agreement between the computed feature importance ranks of different CS methods on a given dataset increases in terms of top-1 and top-3 overlaps (compared to the results presented in Section 4.3).

TABLE 3: The median top-1 and top-3 overlap improvements upon removal of feature interactions (FI) between the computed feature importance ranks of the studied CA and CS methods.

Classifier	Top-1 Overlap		Top-3 Overlap	
	RQ1 results	**After removal of FI**	**RQ1 results**	**After removal of FI**
xgbTree	High	High	Medium	**Strong**
rpart	Low	**High**	Medium	**Strong**
rf	High	High	Medium	**Strong**
glmnet	High	High	Small	**Moderate**
avNNet	Low	Low	Small	**Moderate**

TABLE 4: The median top-1 and top-3 overlap improvements upon removal of feature interactions (FI) between the computed feature importance ranks of the studied CS methods.

Dataset	Top-1 Overlap		Top-3 Overlap	
	RQ3 results	**After removal of FI**	**RQ3 results**	**After removal of FI**
poi-3.0	Low	Low	Negligible	Negligible
camel-1.2	Low	Low	Negligible	**Small**
xalan-2.5	Low	Low	Negligible	**Medium**
xalan-2.6	Low	**High**	Small	Small
eclipse34_debug	Low	**High**	Small	Small
eclipse3_swt	Low	Low	Negligible	**Small**
pde	Low	Low	Small	Small
PC5	Low	Low	Negligible	Negligible
mylyn	Low	Low	Small	Small
eclipse-2.0	Low	Low	Negligible	Negligible
JM1	Low	Low	Negligible	**Small**
eclipse-2.1	Low	Low	Negligible	**Small**
prop-5	Low	**High**	Negligible	Negligible
prop-4	Low	**High**	Negligible	**Small**
prop-3	Low	Low	Negligible	Negligible
eclipse-3.0	Low	Low	Small	Small
prop-1	Low	Low	Negligible	**Small**
prop-2	Low	Low	Negligible	**Small**

Results: Result 9) ***Removing the feature interactions increases the median top-1 and top-3 overlap between the studied CA and CS methods across all the five studied classifiers.*** From Table 3 we observe that across all the classifiers, the top-1 and top-3 overlaps improve by at least one level. Such a result indicates that, removing feature interactions with CFS generally yields a higher agreement between the CA and CS methods with regards to feature ranking (i.e., promotes stability of the results).

Result 10) ***Removing the feature interactions increases the median top-1 and top-3 overlap between the studied CS methods on four and five of the 18 studied datasets respectively.*** Table 4 depicts the improvements in top-1 and top-3 overlaps between the studied CS methods across all the studied datasets. From Table 4 we observe that the removal of feature interactions with CFS yields only small improvements. Therefore, we suggest that researchers and practitioners should be cautious when using different CS methods interchangeably even after removing feature interactions. However, our inference is exploratory in nature and thus further research should be conducted to understand advanced feature interaction removal methods may help improve the agreement across different CS methods.

Result 11) ***After removing the feature interactions, SHAP and Permutation yield a strong agreement on all datasets.*** From the results presented in Section 4.2 we observe that the Permutation and SHAP had less than large overlap only on the Prop-4 and PC5 datasets. However, upon removal of feature interactions from these two datasets, the feature importance ranks computed by the studied CA methods have a large top-1 and top-3 overlap across all the studied datasets and classifiers.

6 Implications

In this section, we outline the implications that one can derive from our results, including potential pitfalls to avoid and future research opportunities.

Implication 1) ***Researchers and practitioners should be aware that feature interactions can hinder classifier interpretation. We recommend these stakeholders to detect feature interactions in their datasets (e.g., by means of the Friedman's H-statistic) and, in the positive scenario, remove these interactions if possible (e.g., by preprocessing the dataset with the CFS method).*** From Section 5.1 and Section 5.2, we find that removing the feature interactions, even with a simple method like CFS, increases the agreement between the feature importance rankings produced by the studied feature importance methods. In other words, once feature interactions are removed, the final feature ranking tends to change. Hence, we consider that feature interactions hinder the interpretability of machine learning models in defect prediction. We thus encourage researchers and practitioners to detect feature interactions in their dataset (e.g., by means of the Friedman's H-statistic). In case interactions are discovered, we encourage these stakeholders to remove them if possible (e.g., by preprocessing the dataset with the CFS method).

Implication 2) ***The lack of clear specification of the feature importance method employed in software engineering studies seriously threatens the reproducibility of these studies and the generalizability of their insights.*** 14% of the studies listed in Table 1 do not specify their employed feature importance method to arrive at their insights. The absence of the specification of the feature importance method employed is more prevalent for random forest classifiers (3/11 studies) – the classifier that is widely used in software engineering. This poses a serious threat, as many random forest implementation across the popular data mining toolboxes come with many different ways of computing the feature importance. For instance, random forest implementation in the R package *randomForest*[3] has 3 feature importance methods available and the R package *partykit*[4] has 2 implementations of feature importance methods for random forest.

Implication 3) ***Future research should evaluate the extent to which SHAP and Permutation can be used interchangeably.*** We conjecture that using either SHAP or Permutation would lead to similar rankings for defect datasets that have similar characteristics to those studied by us. We also conjecture that the feature importance rankings of prior studies in defect classification would not change much if Permutation were to be replaced with SHAP or vice-versa. Nonetheless, we do emphasize that even small changes in rankings could be meaningful in practice and lead to completely different action plans (e.g., in terms of prioritizing pieces of code to be tested or reviewed). Hence, despite the similar rankings produced by SHAP and Permutation, concluding that they can be used interchangeably would be an overstatement. We thus invite future work to further evaluate the differences in the rankings produced by SHAP and Permutation (e.g., by evaluating defect datasets that have different characteristics compared to the ones that we studied).

Implication 4) ***Future research should consider investigating whether more advanced feature interaction removal methods can increase the agreement between the feature interaction rankings produced by different feature importance methods.*** By means of a simple feature interaction removal method like CFS, we are able to improve the agreement between the feature importance rankings produced by CS and CA methods (Section 5.2). Even between the computed feature importance ranks of CS methods, removing feature interactions helps to improve agreement in some cases. Such a result suggests that more sophisticated feature interaction removal methods have the potential to further improve the agreement between the computed feature importance ranks of different feature importance methods.

7 Threats to Validity

In the following, we discuss the threats to the validity of our study.

Internal validity. We choose classifier families who has a CS method. As previous studies show that different classifiers may have different performance on a given dataset [17, 82], this could be a potential threat. However, we choose representative classifiers from 6 of the 8 commonly used classifier families as outlined by Ghotra et al. [82].

We use the AUC and IFA performance measures to shortlist the classifiers used in this study. However, using different performance measures like P_{opt20}, MCC, and F-Measure to shortlist the classifiers to include in our study might potentially bias the conclusion of our study and we declare it as a threat to internal validity. We invite the future studies to revisit our study by using different performance measures to shortlist the classifiers to be used to compare feature importance ranks computed by different feature importance methods.

The classifiers that we choose to study are either probabilistic or stochastic in nature. We do not include any deterministic classifiers like Naive Bayes in our study and it could be a potential threat to the internal validity of our study. Particularly since, Wu et al. [118] point out that computed feature importance ranks of classifiers like decision trees can be particularly sensitive to data characteristics. We invite the future research to revisit our findings on deterministic and stable learners.

Construct validity. In our study, we choose classifiers where all the studied datasets achieved an AUC above 0.70. According to Muller et al. [119], an AUC score above 0.70 indicates the fair discriminative capability of a classifier. Furthermore, these datasets have been used in many of the studies as outlined in Table 1.

We use a simple random search method to hyperparameter tune our studied classifiers. Our decision stems from the work of Tantithamthavorn et al. [78] who show that, irrespective of the performance measure considered, different hyperparameter tuning methods (including grid search, random search, differential evolution based methods and genetic algorithm based methods) yield similar performance improvements. The authors suggest that, as far as performance improvements are concerned, researchers

3. https://cran.r-project.org/web/packages/randomForest/index.html
4. https://cran.r-project.org/web/packages/partykit/index.html

and practitioners can safely use any of the aforementioned automated hyperparameter tuning methods to tune defect classifiers. Therefore, we consider that employing different or more advanced hyperparameter optimization methods would not necessarily result in better hyperparameters for our studied classifiers. Nonetheless, we invite future work to revisit our findings by using more advanced hyperparameter tuning methods to tune the classifiers.

The hyperparameters that we tune for our studied classifiers are limited. Similar to Jiarpakdee et al. [69], we use the automated hyperparameter optimization option provided by caret package to tune the hyperparameters of the studied classifiers. caret package only supports tuning a limited number of hyperparameters (please see Table 6). Tuning a wider range of hyperparameters for the studied classifiers might potentially change our findings. We consider it a threat and invite the future studies to revisit our findings by tuning a wider range of hyperparameters for the studied classifiers.

External validity. In this study, we choose 18 datasets that represent software projects across several corpora (e.g., NASA and PROMISE) and domains (both proprietary and open-source). However, our results likely do not generalize to all software defect datasets. Nevertheless, the datasets that we use in our study are extensively used in the field of software defect prediction [2, 8, 17, 31, 38, 77] and is representative of several corpora and domains. Therefore we argue that our results will still hold. However, future replication across different datasets using our developed methodology might be fruitful.

Secondly, we only consider one defect prediction context in our study (i.e., within-project defect prediction). Yet, there are multiple defect prediction contexts such as Just-In-Time defect prediction [120, 121] and cross-project defect prediction [2]. Hence future studies are needed to explore these richer contexts.

Finally, we study a limited number of CS and CA methods and therefore, our results might not readily generalize to other feature importance methods. For instance, there are recent developments like LIME [122] that have been proposed in the machine learning community for generating feature importance ranks. Nevertheless, the approach and the metrics that we use in our study are applicable to any feature importance method. Therefore, we invite future studies to use our approach to re-examine our findings on other (current and future) feature importance methods.

8 Conclusion

Classifiers are increasingly used to derive insights from data. Typically, insights are generated from the feature importance ranks that are computed by either CS or CA methods. However, the choice between the CS and CA methods to derive those insights remains arbitrary, even for the same classifier. In addition, the choice of the exact feature important method is seldom justified. In other words, several prior studies use feature importance methods interchangeably without any specific rationale, even though different methods compute the feature importance ranks differently. Therefore, in this study, we set out to estimate the extent to which feature importance ranks that are computed by CS and CA methods differ.

By means of a case study on 18 defect datasets and 6 defect classifiers, we observe that while the computed feature importance ranks by different CA methods strongly agree with each other, the computed feature importance ranks of CS methods do not. Furthermore, the computed feature importance ranks of studied CA and CS methods do not strongly agree with each other either. Except when using different studied CA methods, even the feature reported as the most important feature differs for many of the studied classifiers – raising concerns about the stability of conclusions across replicated studies.

We further find that the commonly used defect datasets are rife with feature interactions and these feature interactions impact the computed feature importance ranks of the CS methods (not the CA methods). We also demonstrate that removing these feature interactions, even with simple methods like CFS improves agreement between the computed feature importance ranks of CA and CS methods. We end our study by providing several guidance for future studies, e.g., future research is needed to investigate the impact of advanced feature interaction removal methods on computed feature importance ranks of different CS methods.

ACKNOWLEDGEMENT: This research was partially supported by JSPS KAKENHI Japan (Grant Numbers: JP18H03222) and JSPS International Joint Research Program with SNSF (Project "SENSOR").

References

[1] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead Jr, "Does bug prediction support human developers? findings from a google case study," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 372–381.

[2] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009, pp. 91–100.

[3] B. Caglayan, B. Turhan, A. Bener, M. Habayeb, A. Miransky, and E. Cialini, "Merits of organizational metrics in defect prediction: an industrial replication," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, 2015, pp. 89–98.

[4] E. Shihab, A. E. Hassan, B. Adams, and Z. M. Jiang, "An industrial study on the risk of software changes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 62.

[5] F. Zhang, A. E. Hassan, S. McIntosh, and Y. Zou, "The use of summation to aggregate software metrics hinders the performance of defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 5, pp. 476–491, 2016.

[6] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: a study of breakage and surprise defects," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 300–310.

[7] D. Chen, W. Fu, R. Krishna, and T. Menzies, "Applications of psychological science for actionable analytics," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 456–467.

[8] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on the interpretation of defect models," *IEEE Transactions on Software Engineering*, 2019.

[9] C. Theisen, K. Herzig, P. Morrison, B. Murphy, and L. Williams, "Approximating attack surfaces with stack traces," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, 2015, pp. 199–208.

[10] K. Herzig, S. Just, and A. Zeller, "The impact of tangled code changes on defect prediction models," *Empirical Software Engineering*, vol. 21, no. 2, pp. 303–336, 2016.

[11] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *15th international symposium on software reliability engineering*. IEEE, 2004, pp. 417–428.

[12] H. Jahanshahi, D. Jothimani, A. Başar, and M. Cevik, "Does chronology matter in jit defect prediction?: A partial replication study," in *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2019, pp. 90–99.

[13] T. Mori and N. Uchihira, "Balancing the trade-off between accuracy and interpretability in software defect prediction," *Empirical Software Engineering*, vol. 24, no. 2, pp. 779–825, 2019.

[14] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, "A benchmark for interpretability methods in deep neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 9737–9748.

[15] C. Molnar, "Interpretable machine learning," *A Guide for Making Black Box Models Explainable*, vol. 7, 2018.

[16] S. Chakraborty, R. Tomsett, R. Raghavendra, D. Harborne, M. Alzantot, F. Cerutti, M. Srivastava, A. Preece, S. Julier, R. M. Rao *et al.*, "Interpretability of deep learning models: a survey of results," in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2017, pp. 1–6.

[17] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "The impact of using regression models to build defect classifiers," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 135–145.

[18] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, 2018.

[19] C. Treude and M. Wagner, "Predicting good configurations for github and stack overflow topic models," in *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 2019, pp. 84–95.

[20] T. Yu, W. Wen, X. Han, and J. Hayes, "Conpredictor: Concurrency defect prediction in real-world applications," *IEEE Transactions on Software Engineering*, 2018.

[21] K. Herzig, "Using pre-release test failures to build early post-release defect prediction models," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 300–311.

[22] A. B. de González, D. R. Cox *et al.*, "Interpretation of interaction: A review," *The Annals of Applied Statistics*, vol. 1, no. 2, pp. 371–385, 2007.

[23] G. Freeman, "The analysis and interpretation of interactions," *Journal of Applied Statistics*, vol. 12, no. 1, pp. 3–10, 1985.

[24] A. Fisher, C. Rudin, and F. Dominici, "All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously," *arXiv preprint arXiv:1801.01489*, 2018.

[25] S. Devlin, C. Singh, W. J. Murdoch, and B. Yu, "Disentangled attribution curves for interpreting random forests and boosted trees," *arXiv preprint arXiv:1905.07631*, 2019.

[26] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," *arXiv preprint arXiv:1802.03888*, 2018.

[27] M. A. Hall, "Correlation-based feature selection for machine learning," 1999.

[28] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data engineering*, vol. 15, no. 6, pp. 1437–1447, 2003.

[29] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *IEEE Transactions on software engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[30] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on software engineering*, vol. 29, no. 6, pp. 481–494, 2003.

[31] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 2008, pp. 531–540.

[32] L. Angelis, I. Stamelos, and M. Morisio, "Building a software cost estimation model based on categorical data," in *Proceedings Seventh International Software Metrics Symposium*. IEEE, 2001, pp. 4–15.

[33] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 284–292.

[34] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 171–180.

[35] T. A. Ghaleb, D. A. da Costa, and Y. Zou, "An empirical study of the long duration of continuous integration builds," *Empirical Software Engineering*, pp. 1–38, 2019.

[36] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 111–120.

[37] F. Calefato, F. Lanubile, and N. Novielli, "An empirical assessment of best-answer prediction models in technical q&a sites," *Empirical Software Engineering*, vol. 24, no. 2, pp. 854–901, 2019.

[38] R. Premraj and K. Herzig, "Network versus code metrics to predict defects: A replication study," in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2011, pp. 215–224.

[39] G. Gay, T. Menzies, M. Davies, and K. Gundy-Burlet, "Automatically finding the control variables for complex system behavior," *Automated Software Engineering*, vol. 17, no. 4, pp. 439–468, 2010.

[40] L. B. Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, and A. D. Brucker, "Time for addressing software security issues: Prediction models and impacting factors," *Data Science and Engineering*, vol. 2, no. 2, pp. 107–124, 2017.

[41] J. Santos and O. Belo, "Estimating risk management in software engineering projects," in *Industrial Conference on Data Mining*. Springer, 2013, pp. 85–98.

[42] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "Mode: automated neural network model debugging via state differential analysis and input selection," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 175–186.

[43] K. El-Emam, D. Goldenson, J. McCurley, and J. Herbsleb, "Modelling the likelihood of software process improvement: An exploratory study," *Empirical Software Engineering*, vol. 6, no. 3, pp. 207–229, 2001.

[44] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 119–125.

[45] O. Malgonde and K. Chari, "An ensemble-based model for predicting agile software development effort," *Empirical Software Engineering*, vol. 24, no. 2, pp. 1017–1055, 2019.

[46] M. Haran, A. Karr, M. Last, A. Orso, A. A. Porter, A. Sanil, and S. Fouche, "Techniques for classifying executions of deployed software to support software engineering tasks," *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 287–304, 2007.

[47] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 345–355.

[48] R. Niedermayr and S. Wagner, "Is the stack distance between test case and method correlated with test effectiveness?" in *Proceedings of the Evaluation and Assessment on Software Engineering*. ACM, 2019, pp. 189–198.

[49] D. Martens and W. Maalej, "Towards understanding and detecting fake reviews in app stores," *Empirical Software Engineering*, pp. 1–40, 2019.

[50] Y. Fan, X. Xia, D. Lo, and S. Li, "Early prediction of merged code changes to prioritize reviewing tasks," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3346–3393, 2018.

[51] L. Bao, X. Xia, D. Lo, and G. C. Murphy, "A large scale study of long-time contributor prediction for github projects," *IEEE Transactions on Software Engineering*, 2019.

[52] T. Dey and A. Mockus, "Are software dependency supply chain metrics useful in predicting change of popularity of npm packages?" in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2018, pp. 66–69.

[53] A. Dehghan, A. Neal, K. Blincoe, J. Linaker, and D. Damian, "Predicting likelihood of requirement implementation within the planned iteration: an empirical study at ibm," in *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 2017, pp. 124–134.

[54] K. Blincoe, A. Dehghan, A.-D. Salaou, A. Neal, J. Linaker, and D. Damian, "High-level software requirements and iteration changes: a predictive model," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1610–1648, 2019.

[55] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM, 2011, p. 11.

[56] T. Menzies and M. Shepperd, "Special issue on repeatable results in software engineering prediction," 2012.

[57] H. K. Dam, T. Tran, and A. Ghose, "Explainable software analytics," in *Proceedings of the 40th International Conference on Software Engi-*

neering: New Ideas and Emerging Results. ACM, 2018, pp. 53–56.

[58] J. Hihn and T. Menzies, "Data mining methods and cost estimation models: Why is it so hard to infuse new ideas?" in *2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. IEEE, 2015, pp. 5–9.

[59] C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn, "Bias in random forest variable importance measures: Illustrations, sources and a solution," *BMC bioinformatics*, vol. 8, no. 1, p. 25, 2007.

[60] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 5, pp. 412–428, 2017.

[61] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146–2189, 2016.

[62] L. Bao, Z. Xing, X. Xia, D. Lo, and S. Li, "Who will leave the company?: a large-scale industry study of developer turnover by mining monthly work report," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 170–181.

[63] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? a case study on free android applications," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 301–310.

[64] S. L. Abebe, N. Ali, and A. E. Hassan, "An empirical study of software release notes," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1107–1142, 2016.

[65] L. C. Briand, J. Daly, V. Porter, and J. Wust, "Predicting fault-prone classes with design measures in object-oriented systems," in *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No. 98TB100257)*. IEEE, 1998, pp. 334–343.

[66] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, 2009.

[67] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 4–14.

[68] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? an empirical case study of windows vista," in *Proceedings of the 31st international conference on software engineering*. IEEE Computer Society, 2009, pp. 518–528.

[69] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Transactions on Software Engineering*, 2020.

[70] A. Gosiewska and P. Biecek, "ibreakdown: Uncertainty of model explanations for non-additive predictive models," *CoRR*, vol. abs/1903.11420, 2019.

[71] C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis, "Conditional variable importance for random forests," *BMC bioinformatics*, vol. 9, no. 1, p. 307, 2008.

[72] E. Candes, Y. Fan, L. Janson, and J. Lv, "Panning for gold:'model-x'knockoffs for high dimensional controlled variable selection," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 80, no. 3, pp. 551–577, 2018.

[73] U. Grömping, "Variable importance assessment in regression: linear regression versus random forest," *The American Statistician*, vol. 63, no. 4, pp. 308–319, 2009.

[74] L. Auret and C. Aldrich, "Empirical comparison of tree ensemble variable importance measures," *Chemometrics and Intelligent Laboratory Systems*, vol. 105, no. 2, pp. 157–170, 2011.

[75] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "Autospearman: Automatically mitigating correlated software metrics for interpreting defect models," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, 2018, pp. 92–103.

[76] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 812–823.

[77] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2016.

[78] ——, "The impact of automated parameter optimization on defect prediction models," *IEEE Transactions on Software Engineering*, 2018.

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

[80] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: The use of machine learning in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 40, no. 6, pp. 603–616, 2014.

[81] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Comments on "researcher bias: the use of machine learning in software defect prediction"," *IEEE Transactions on Software Engineering*, vol. 42, no. 11, pp. 1092–1094, 2016.

[82] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 2015, pp. 789–800.

[83] M. Kuhn, "Variable importance using the caret package," 2012.

[84] A. Avati, K. Jung, S. Harman, L. Downing, A. Ng, and N. H. Shah, "Improving palliative care with deep learning," *BMC medical informatics and decision making*, vol. 18, no. 4, p. 122, 2018.

[85] S. Janitza, C. Strobl, and A.-L. Boulesteix, "An auc-based permutation variable importance measure for random forests," *BMC bioinformatics*, vol. 14, no. 1, p. 119, 2013.

[86] J. Jiarpakdee, "Towards a more reliable interpretation of defect models," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 210–213.

[87] A. Husain, "Understanding how developers work on change tasks using interaction history and eye gaze data," Ph.D. dissertation, 2015.

[88] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "The impact of automated feature selection techniques on the interpretation of defect models," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3590–3638, 2020.

[89] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[90] I. Covert, S. M. Lundberg, and S.-I. Lee, "Understanding global feature contributions with additive importance measures," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[91] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, 2017, pp. 4765–4774.

[92] M. Viggiato and C.-P. Bezemer, "Trouncing in dota 2: An investigation of blowout matches," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 294–300.

[93] G. Esteves, E. Figueiredo, A. Veloso, M. Viggiato, and N. Ziviani, "Understanding machine learning software defect predictions," *Automated Software Engineering*, vol. 27, no. 3, pp. 369–392, 2020.

[94] K. Peng and T. Menzies, "How to improve ai tools (by adding in se knowledge): Experiments with the timelime defect reduction tool," *arXiv preprint arXiv:2003.06887*, 2020.

[95] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ACM, 2018, pp. 286–295.

[96] F. E. Harrell Jr, *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer, 2015.

[97] B. Efron, "Estimating the error rate of a prediction rule: improvement on cross-validation," *Journal of the American statistical association*, vol. 78, no. 382, pp. 316–331, 1983.

[98] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Information and Software Technology*, vol. 76, pp. 135–146, 2016.

[99] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[100] M. Kuhn *et al.*, "Building predictive models in r using the caret package," *Journal of statistical software*, vol. 28, no. 5, pp. 1–26, 2008.

[101] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[102] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "Impact of discretization noise of the dependent variable on machine learning

classifiers in software engineering," *IEEE Transactions on Software Engineering*, 2019.

[103] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 international symposium on software testing and analysis*, 2011, pp. 199–209.

[104] Q. Huang, X. Xia, and D. Lo, "Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 159–170.

[105] C. Tantithamthavorn, "Scottknottesd: The scott-knott effect size difference (esd) test," *R package version*, vol. 2, 2016.

[106] Z. C. Lipton, "The mythos of model interpretability," *arXiv preprint arXiv:1606.03490*, 2016.

[107] M. G. Kendall, "Rank correlation methods." 1948.

[108] B. Kitchenham, S. L. Pfleeger, and N. Fenton, "Towards a framework for software measurement validation," *IEEE Transactions on software Engineering*, vol. 21, no. 12, pp. 929–944, 1995.

[109] A. Bachmann and A. Bernstein, "When process data quality affects the number of bugs: Correlations in software engineering datasets," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 2010, pp. 62–71.

[110] H. Akoglu, "User's guide to correlation coefficients," *Turkish journal of emergency medicine*, 2018.

[111] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[112] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 263–272.

[113] J. H. Friedman, B. E. Popescu *et al.*, "Predictive learning via rule ensembles," *The Annals of Applied Statistics*, vol. 2, no. 3, pp. 916–954, 2008.

[114] M. N. Wright, A. Ziegler, and I. R. König, "Do little interactions get lost in dark random forests?" *BMC bioinformatics*, vol. 17, no. 1, p. 145, 2016.

[115] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[116] J. Cahill, J. M. Hogan, and R. Thomas, "Predicting fault-prone software modules with rank sum classification," in *2013 22nd Australian Software Engineering Conference*. IEEE, 2013, pp. 211–219.

[117] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[118] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[119] M. P. Muller, G. Tomlinson, T. J. Marrie, P. Tang, A. McGeer, D. E. Low, A. S. Detsky, and W. L. Gold, "Can routine laboratory tests discriminate between severe acute respiratory syndrome and other causes of community-acquired pneumonia?" *Clinical infectious diseases*, vol. 40, no. 8, pp. 1079–1086, 2005.

[120] T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, "Deepjit: an end-to-end deep learning framework for just-in-time defect prediction," in *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 2019, pp. 34–45.

[121] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2012.

[122] M. T. Ribeiro, S. Singh, and C. Guestrin, "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*. ACM, 2016, pp. 1135–1144.

[123] L. S. Shapley, "A value for n-person games," *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 307–317, 1953.

[124] C. Halimu, A. Kasem, and S. S. Newaz, "Empirical comparison of area under roc curve (auc) and mathew correlation coefficient (mcc) for evaluating machine learning algorithms on imbalanced datasets for binary classification," in *Proceedings of the 3rd international conference on machine learning and soft computing*, 2019, pp. 1–6.

[125] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 8–17.

[126] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2009, pp. 121–130.

Gopi Krishnan Rajbahadur is a Senior Researcher at the Centre for Software Excellence at Huawei, Canada. He holds a PhD in computer science Queen's University, Canada. He received his BE in computer Science and Engineering from SKR Engineering college, Anna University, India. His research interests include Software Engineering for Artificial Intelligence (SE4AI), Artificial Intelligence for Software Engineering (AI4SE), Mining Software Repositories (MSR) and Explainable AI (XAI).

Shaowei Wang Shaowei Wang is an assistant professor in the Department of Computer Science at University of Manitoba. He obtained his Ph.D. from Singapore Management University and his BSc from Zhejiang University. His research interests include software engineering, machine learning, data analytics for software engineering, automated debugging, and secure software development. He is one of four recipients of the 2018 distinguished reviewer award for the Springer EMSE (SE's highest impact journal). More information at: https://sites.google.com/site/wswshaoweiwang.

Gustavo A. Oliva is a Research Fellow at the School of Computing of Queen's University in Canada under the supervision of professor Dr. Ahmed E. Hassan. Gustavo leads the blockchain research team at the Software Analysis and Intelligence Lab (SAIL), where he investigates blockchain technologies through the lens of Software Engineering. In addition, Gustavo also conducts research on software ecosystems, explainable AI, and software maintenance. Gustavo received his MSc and PhD degrees from the University of São Paulo (USP) in Brazil under the supervision of professor Dr. Marco Aurélio Gerosa. More information at: https://www.gaoliva.com

Yasutaka Kamei is an associate professor at Kyushu University in Japan. He received his BE degree in informatics from Kansai University, and PhD degree in information science from the Nara Institute of Science and Technology. He was a research fellow of the JSPS (PD) from July 2009 to March 2010. From April 2010 to March 2011, he was a postdoctoral fellow at Queen's University in Canada. His research interests include empirical software engineering, open source software engineering, and mining software repositories (MSR). He serves on the editorial boards of Springer Journal of Empirical Software Engineering. He is a senior member of the IEEE. More information at http://posl.ait.kyushu-u.ac.jp/~kamei

Ahmed E. Hassan is an IEEE Fellow, an ACM SIGSOFT Influential Educator, an NSERC Steacie Fellow, the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. He received a PhD in Computer Science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. He also serves/d on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, and PeerJ Computer Science. More information at: http://sail.cs.queensu.ca

Appendix A
Case Study: Additional Information

A.1 Studied Datasets

Table 5 shows various basic characteristics about each of the studied datasets.

TABLE 5: Overview of the datasets studied in our case study

Project	DR	#Files	#Fts	#FACRA	EPV
Poi-3.0	63.5	442	20	12	14.05
Camel-1.2	35.53	608	20	10	10.8
Xalan-2.5	48.19	803	20	11	19.35
Xalan-2.6	46.44	885	20	11	20.55
Eclipse34_debug	24.69	1065	17	10	15.47
Eclipse34_swt	43.97	1485	17	9	38.41
Pde	13.96	1497	15	6	13.93
PC5	27.53	1711	38	13	12.39
Mylyn	13.16	1862	15	7	16.33
Eclipse-2.0	14.49	6729	32	9	30.47
JM1	21.49	7782	21	7	79.62
Eclipse-2.1	10.83	7888	32	9	26.69
Prop-5	15.25	8516	20	12	64.95
Prop-4	9.64	8718	20	12	42
Prop-3	11.49	10274	20	12	59
Eclipse-3.0	14.8	10593	32	9	49
Prop-1	14.82	18471	20	13	136.9
Prop-2	10.56	23014	20	13	121.55

DR: Defective Ratio, FACRA: Features After Correlation and Redundancy Analysis, Fts: Features

A.2 Studied Classifiers

Table 6 shows our studied classifiers, the machine learning families to which they belong, and the *caret* function that was used to build the classifiers.

TABLE 6: Studied classifiers and their hyperparameters

Family	Classifier	Caret method	Hyperparameters
Statistical Techniques	Regularized Logistic Regression	glmnet	alpha, lambda
Rule-Based Techniques	C5.0 Rule-Based Tree	C5.0Rules	None
Neural Networks	Neural Networks (with model averaging)	avNNet	size, decay, bag
Decision Trees	Recursive Partitioning and Regression Trees	rpart	K, L, cp
Ensemble methods-Bagging	Random Forest	rf	mtry
Ensemble methods-Boosting	Extreme Gradient Boosting Trees	xgbTree	nrounds, max_depth, eta, gamma, colsample_bytree, min_child_weight, subsample

A.3 Classifier Specific Feature Importance (CS) methods

Table 7 provides a brief explanation about the inner working of the CS methods that we study in this paper. For a more detailed explanation we refer the readers to Kuhn [83].

A.4 Classifier Agnostic Feature Importance (CA) methods

In the following, we describe the two CA methods that we employ in this study.

Permutation feature importance (Permutation). We use the same permutation feature importance method used by Rajbahadur et al. [17]. First, the performance of a classifier is calculated on the non-permuted dataset (we use AUC for measuring the performance in our study). Then each feature in the training set is randomly permuted one by one and this permuted feature along with the other non-permuted features are used to measure the performance of the classifier. If the permutation of a feature decreases the performance of the classifier, compared to the classifier built on the non-permuted dataset, then that feature is considered important. The magnitude of performance changes decides the importance scores of features in a given dataset.

SHapley Additive ExPlanations (SHAP). We use the method outlined by Lundberg and Lee [91]. SHAP uses the game-theory based Shapley values [123]to fairly distribute the credit for a classifier's output among its features. To do so, for each data point in the train set, SHAP first transforms all the features into a space of simplified binary features. Then SHAP estimates how the output probability of the classifier for the given data point can be expressed as a linear combination of these simplified binary features. These computed feature importance scores are theoretically guaranteed to be optimal, additive, and locally accurate for each data point. Therefore, the overall feature importance scores for each studied feature in the classifier can be given by simply summing the feature importance score of each feature across all the data points in the train set [90]. For more details about how SHAP computes the feature importance ranks we refer the readers to studies by Lundberg and Lee [91], Esteves et al. [93]. We use the *vip*[5] R package for computing the SHAP feature importance scores in our study.

A.5 Out-of-sample Bootstrap

The out-of-sample bootstrap is a model validation technique that aims to create a test set that has a similar distribution to that of the training set in each validation iteration. In the following, we briefly describe how it works:

1) For a given dataset, N (where N=size of the given dataset) data points are randomly sampled with replacement. These N data points are used as the *train* set for the classifiers.
2) As the data points were sampled with replacement in the prior step, on an average, 36.8% of the data points do not appear as a part of the *train* set [97]. We use these data points as the *test* set for measuring the performance of the constructed classifiers.

The aforementioned process should be repeated k times for each of the studied dataset. Typical values for k include 100, 500, and 1,000.

A.6 Performance Computation

In the following, we briefly describe each of our adopted performance measures and the rationale for choosing them:

AUC. The ROC curve plots the True Positive Rate (TPR = TP / (TP + FN)) against the False Positive Rate (FPR = FP / (FP + TN) for all possible classification thresholds (i.e., from 0 to 1). The use of the AUC measure has several advantages [82, 95, 101, 102]. First, AUC does not require one to preemptively select a classification threshold. Hence, the AUC measure prevents our study from being influenced by the choice of classification thresholds [17].

5. https://cran.r-project.org/web/packages/vip/index.html

TABLE 7: Brief explanation about the working of caret's CS methods that are used in our study.

CS method	Brief explanation
Logistic Regression FI (LRFI)	Classifier coefficient's t-statistic is reported as the feature importance score
C5.0 Rule-Based Tree FI (CRFI)	The number of training data points that are covered by the leaf nodes, created from the split of a feature is given as the feature importance score for that feature. For instance, the feature that is split in the root node will have a 100% importance as all the training samples will be covered by the terminal nodes leading from it.
Neural Networks (with model averaging) FI (NNFI)	The feature importance score is given by combining the absolute weights used in the neural network
Recursive Partitioning and Regression Trees FI (RFI)	The feature importance score is given by the sum of the reduction in loss function that is brought about by each feature at each split in the tree.
Random Forest FI (RFFI)	Average of difference between the Out-of-Bag (OOB) error for each tree in the forest where none of the features are permuted and the OOB error where each of the features is permuted one by one. The feature permutation's impact on the overall OOB error is reported as the feature importance score
Extreme Gradient Boosting Trees FI (XGFI)	Feature importance score is given by counting the number of times a feature is used in all the boosting trees of the xgboost tree.

For instance, when measuring the performance of a classifier with other performance measures like precision and recall, a threshold (e.g., 0.5) needs to be set up to discretize the classification probability into the "Defective" and "Non-Defective" outcome classes. However, setting this threshold is challenging in practice. The AUC measure circumvents this problem by measuring the TPR and FPR at all possible thresholds. Following this, the performance of the classifier is given as the area under this curve. Another commonly used measure to evaluate the performance of a classifier is the Matthews Correlation Coefficient (MCC). We use AUC instead of MCC because (i) MCC is threshold-dependent (similarly to precision and recall) and (ii) there is empirical evidence showing that AUC is generally more discriminative than MCC (even though AUC and MCC tend to be statistically consistent with each other) [124]. AUC is robust to imbalanced class [101] (even more so than other performance measures like MCC [124]). As a consequence, class imbalances that are typically inherent to defect datasets [125, 126] are automatically accounted for. The AUC measure ranges from 0 to 1. An AUC measure close to 1 indicates the classifier's performance is very high. Conversely, an AUC measure close to 0.5 indicates that the classifier's performance is no better than a random guess.

IFA. The IFA measures the number of false alarms that are encountered before the first defective module is detected by a classifier [69, 104]. We choose the IFA measure in lieu of others (e.g., False Alarm Rate) because several prior studies in fault localization argued that practitioners tend to distrust a classifier whose top recommendations are false alarms. In particular, Parnin and Orso [103], found that developers tend to avoid automated tools if the first few recommendations given by them are false alarms. Therefore, we argue that the number of false alarms that are encountered before the first defective module is detected is a better measure to evaluate the usability of our constructed defect classifiers in practice. The IFA measure is calculated by first sorting the modules in the descending order of their risk as given by the classifiers (i.e, the probability of a module being defective). Then the number of non-defective modules that are predicted as defective before identifying the first true positive (i.e., defective module) is the IFA of a classifier. The IFA measure ranges from 1 (best) to the number of modules in the dataset.

APPENDIX B
FRIEDMAN H-STATISTIC PER DATASET

TABLE 8: No. of features per dataset with Friedman H-Statistic ≥ 0.3 and ≥ 0.5

Dataset	#F with H ≥ 0.3	#F with H ≥ 0.5
Poi-3.0	3	0
Camel-1.2	5	4
Xalan-2.5	4	3
Xalan-2.6	2	0
Eclipse34_debug	6	3
Eclipse34_swt	3	0
Pde	5	4
PC5	4	0
Mylyn	4	4
Eclipse-2.0	6	4
JM1	7	5
Eclipse-2.1	6	3
Prop-5	5	4
Prop-4	5	3
Prop-3	5	3
Eclipse-3.0	6	5
Prop-1	6	2
Prop-2	8	4

F - Features