The Impact of Using Regression Models to Build Defect Classifiers

Gopi Krishnan Rajbahadur, Shaowei Wang
Queen's University, Canada
{krishnan, shaowei}@cs.queensu.ca

Yasutaka Kamei
Kyushu University, Japan
kamei@ait.kyushu-u.ac.jp

Ahmed E. Hassan
Queen's University, Canada
ahmed@cs.queensu.ca

***Abstract*—It is common practice to discretize continuous defect counts into defective and non-defective classes and use them as a target variable when building defect classifiers (discretized classifiers). However, this discretization of continuous defect counts leads to information loss that might affect the performance and interpretation of defect classifiers. Another possible approach to build defect classifiers is through the use of regression models then discretizing the predicted defect counts into defective and non-defective classes (regression-based classifiers).**

In this paper, we compare the performance and interpretation of defect classifiers that are built using both approaches (i.e., discretized classifiers and regression-based classifiers) across six commonly used machine learning classifiers (i.e., linear/logistic regression, random forest, KNN, SVM, CART, and neural networks) and 17 datasets. We find that: i) Random forest based classifiers outperform other classifiers (best AUC) for both classifier building approaches; ii) In contrast to common practice, building a defect classifier using discretized defect counts (i.e., discretized classifiers) does not always lead to better performance.

Hence we suggest that future defect classification studies should consider building regression-based classifiers (in particular when the defective ratio of the modeled dataset is low). Moreover, we suggest that both approaches for building defect classifiers should be explored, so the best-performing classifier can be used when determining the most influential features.

***Index Terms*—Classification via regression; Random forest; Bug prediction; Discretization; Non-Discretization; Model Interpretation;**

I. Introduction

Finding and fixing defects consume more than 80% of the total budget of a software project. These costs can be reduced significantly if the defects are identified and fixed early on [4], [9], [14], [26], [27], [36].

Defect classifiers assist in software quality assurance efforts and in prioritizing process improvement efforts. In particular, defect classifiers can identify defect-prone modules [17], [20], [40], [44], in turn helping quality assurance teams allocate their limited resources to these modules (e.g., packages, files, or classes). Moreover, the trained defect classifiers can be used to understand the impact of the various features (e.g., process or product metrics) on the defect-proneness of a module, in turn helping practitioners (through process improvement efforts) avoid pitfalls that have led to defective modules in the past.

The most common approach to build a classifier is through the discretization of the continuous defect counts into "defective" and "non-defective" classes and using these classes as a target variable (i.e., *discretized defect classifiers*) [8], [21], [25]. However, the discretization of continuous variables (i.e., defect counts in this case) into two classes often leads to a significant loss of information [2], [11], [34] and introduces undesired false positives or false negatives [5].

To avoid the information loss because of discretization, one possible solution is to perform classification via regression [18], [37], [42]. Classification via regression first builds a regression model using the non-discretized defect counts then uses the predicted defect counts to identify the presence or absence of defect (i.e., *regression-based defect classifiers*.) However, it is not clear which classifier building approach leads to better performing defect classifiers. It is also not clear whether these two approaches would produce classifiers which are influenced by different set of features (e.g., product versus process metrics).

In this paper, we examine the use of regression models to build defect classifiers in terms of performance (i.e., Area Under the receiver operator characteristic Curve (AUC)) and model interpretation (i.e., influential features that impact the defect-proneness of a module). We conduct our study on six commonly used classifiers (i.e., linear/logistic regression, random forest, KNN, SVM, CART, and neural networks) and using 17 Tera-PROMISE defect datasets [1]. We conduct our study through the following research questions:

- **RQ1. How well do regression-based classifiers perform?**
  In contrast to current practices in our field, building classifiers using discretized defect counts does not always lead to better performance. Regression-based classifiers outperform discretized classifiers when the defective ratio of the modeled dataset is low ($< 15\%$) and the pattern reverses when the defective ratio is high ($> 35\%$) for the random forest classifier.
- **RQ2. Are discretized and regression-based classifiers influenced by the same set of features?**
  The most influential features (i.e., Rank 1 features) do not vary significantly between both approaches for building a defect classifier (when using a random forest based classifier). However, we observe significant variances for features at lower ranks.

Thus we suggest that future defect classification studies should consider building regression-based classifiers (in par-

ticular when the defective ratio of the modeled dataset is low). Moreover, we suggest that both approaches for building defect classifiers should be explored, so the best-performing classifier can be used when determining the most influential features.

Paper organization. Section II presents the background and related work. In section III, we describe the data collection and overall approach of our study and present a preliminary study in Section IV, The results of our RQs and their implications are discussed in Section V. In Section VI, we delve deeper into some of our findings and discuss the threats to validity in Section VII. Finally, we conclude our paper in Section VIII.

II. Background and Related Work

A. Defect Prediction

Defect classifier can be used to identify modules that are likely to be defective (i.e., defect-prone) [20], [24], [29], [35], [40], [44]. Quality assurance teams can then effectively focus their limited testing and code review resources on such defect-prone modules. For instance, Kim et al. [20] propose an approach to predict defective modules by leveraging software change history data. Second, defect classifiers can be used to plan process improvement efforts by focusing on controlling the most influential features in the classifiers (i.e., model interpretation) [6], [8], [13], [23], [25], [32], [33]. For example, Cataldo et al. [8] employ a logistic classifier to identify that defects are influenced by workflow dependencies over syntactic dependencies. In this paper, we propose the use of regression models to build defect classifiers in lieu of the common use of machine learning classifiers. We structure our analysis in terms of performance (i.e., AUC) and model interpretation (i.e., influential features that impact the defect-proneness of a module).

B. Discretization of a Continuous Variable

Discretization is referred as the process of transferring a continuous variable into its discrete classes. Prior studies point out that discretization increases the risk of information loss and introduction of false positives [2], [5], [11], [22], [34], which is due to the loss of variance that happens to continuous target variable (defect counts) during discretization [11].

Traditionally, prior defect studies usually do not delve into such information loss. These studies discretize the defect counts into "defective" and "non-defective" classes then use the discretized defect counts to build a classifier. To alleviate the information loss problem, one solution is to build a classifier through regression, which uses the non-discretized defect counts to build a regression model first, then performs classification based on the results from the regression models [18], [37], [42].

In classification via regression the continuous defect counts are used as a target variable directly and the resulting prediction is used for classification based on a threshold. However such an approach is discouraged in the machine learning literature as it is much more sensitive to the fluctuation in data than traditional classification techniques [30].

TABLE I: Overview of the studied datasets.

Project	DR(%)	#Files	#Features	#FACRA	EPV
Eclipse-2.0	14.5	6,729	32	12	30
Eclipse-2.1	10.8	7,888	32	12	30
Eclipse-3.0	14.8	10,593	32	12	49
Camel-1.2	35.5	608	20	12	11
Mylyn	13.2	1,862	15	8	16
PDE	14.0	1,497	15	9	14
Prop-1	14.8	18,471	20	15	137
Prop-2	10.6	23,014	20	14	122
Prop-3	11.5	10,274	20	15	59
Prop-4	9.6	8,718	20	15	42
Prop-5	15.3	8,516	20	14	65
Xalan-2.5	48.2	803	20	14	19
Xalan-2.6	46.4	885	20	13	21
Lucene-2.4	59.7	340	20	13	10
Poi-2.5	64.4	385	20	12	12
Poi-3.0	63.6	442	20	13	14
Xerces-1.4	74.3	588	20	11	22

DR - Defective Ratio; FACRA - Features After Correlation and Redundancy Analysis

In summary, both discretized and regression-based defect classifiers have inherent flaws and it is not clear which approach is better. In this paper, we compare the traditional classification approach against the classification via regression approach for building defect classifiers in order to better understand both approaches in the context of software defect data.

III. Experimental Setting

This section describes the data collection, and gives an overview of our study approach.

A. Data collection

We use data from the Tera-PROMISE Repository [1]. Tera-PROMISE contains 101 software projects data, and the types of these projects are diverse. Using data from Tera-PROMISE helps us draw more general observations across different datasets. We select datasets based on the following two criteria which are similar to a prior study [38]:

Criterion 1: Remove datasets with an EPV that is larger than 10. Events Per Variable (EPV) is defined as the ratio of the frequency of the least occurring class in the outcome variable to the number of features that are involved in training of a classifier. Prior studies show that the EPV value has a significant influence on the performance of defect classifiers [31], [39]. In particular, defect classifiers trained with datasets with a low EPV value yield unstable results [38], [39]. To ensure the stability of our results, we select datasets with an EPV value that is larger than 10 [31]. We calculate the EPV for our datasets using the steps that are provided by Tantithamthavorn et al. [39].

Criterion 2: Remove datasets that have more than 80% defective modules. We choose datasets that have less than 80% defective modules, because it is highly unlikely for any software project to have modules with defects that are considerably more than clean modules.

Among the 101 Tera-PROMISE datasets, we excluded 78 datasets since they had an EPV value that is less than 10. To satisfy criterion 2, we eliminate the Xalan-2.7 project and

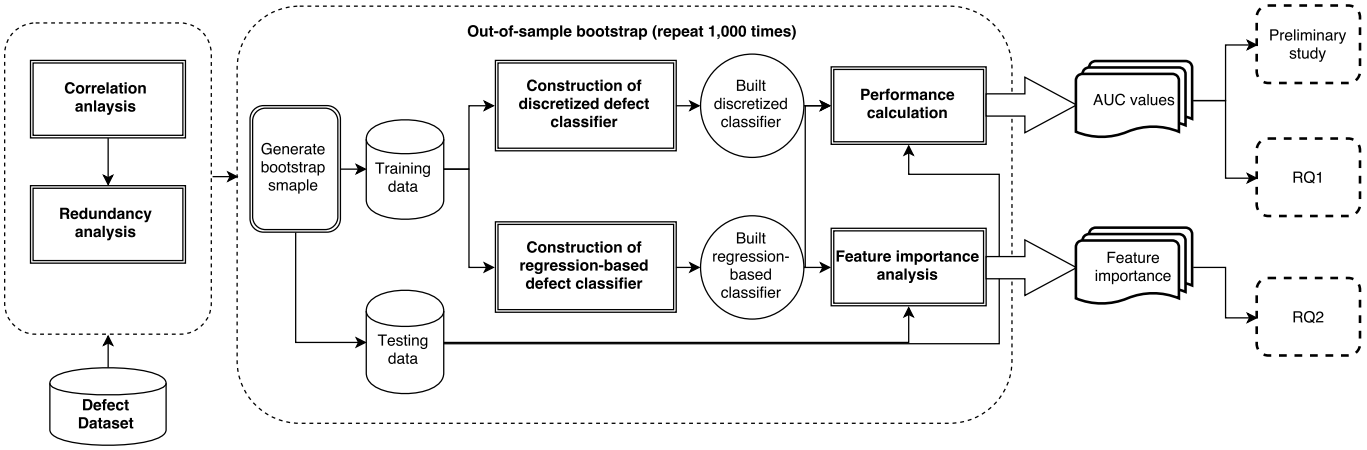

Fig. 1: An overview of our study approach.

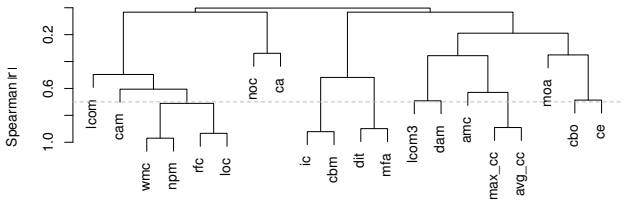

Fig. 2: Clustered features after correlation analysis for Poi-3.0. The dotted line represents the 0.7 threshold and we select only one feature from the sub-hierarchy below for inclusion in our analysis.

end up with 22 datasets that satisfy our criteria. We had to eliminate another 5 datasets as they did not have the actual defect counts for each module (they only had the module class, i.e., buggy or not buggy). We end up with 17 datasets for our analysis. Table I shows the selected datasets for our study along with basic characteristics about each dataset.

B. Overall approach

An overview approach of our study is presented in Figure 1. First, we perform correlation analysis and redundancy analysis. Then, we build two defect classifiers, one using the non-discretized defect counts and the other using the discretized defect count (i.e., "defective" or "non-defective"), respectively. After the classifiers are built, we calculate their performance using the *Area Under the receiver operator characteristic Curve (AUC)* and compute the feature importance for each classifier. We repeat this process 1,000 times using out-of-sample bootstrap validation to ensure that our drawn conclusions are statistically robust as suggested by Tantithamthavorn et al. [39]. In each iteration of the bootstrap, we compute the AUC values and feature importance for the discretized and regression-based classifiers. We use the computed AUC and feature importances to conduct our preliminary study in Section IV and answer our research questions in Section V.

The individual steps of our approach are explained in detail below.

C. Correlation analysis & Redundancy analysis

Correlation analysis: To avoid multicollinearity problems in our classifiers, we perform a correlation analysis to remove highly correlated features. We use a feature clustering analysis to construct a hierarchical overview of the Spearman correlations among features. For sub-hierarchies of features with correlations larger than 0.7, we select only one feature from the sub-hierarchy for inclusion into our classifiers. When selecting the feature for inclusion, we select the feature that is simplest to interpret and compute. We use the **varclus** function from the **Hmisc** R package in this paper. For example, Figure 2 shows the hierachical clustering of the features of the Poi-3.0 project. We observe that the features "wmc", "npm", "rfc", and "loc" have correlation values larger than 0.7. We choose "loc" for inclusion in our classifier as it is relatively easy to compute and explain. We repeat a similar process for other correlated features.

Redundancy analysis: The Correlation analysis handles multicollinearity, but it does not remove redundant features, which are features that do not add additional information with respect to other features [43]. The presence of these features distorts the relationship between features and the target variable. Hence, it is important to remove redundant features prior to classifier construction. We use the **redun** function from **rms** R package to remove redundant features. The function drops features iteratively until either no previously constructed model of features achieved an R^2 above a chosen cutoff threshold (0.9 in our case). Table I shows the number of remaining features in our datasets after employing feature selection on each dataset.

D. Classifier construction

In our experiments, we use two approaches to build defect classifiers to predict whether a module has defects or not:

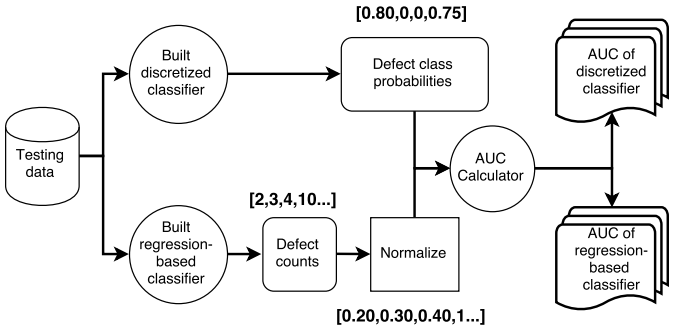

Fig. 3: An overview of performance evaluation.

a traditional defect classifier that is built with discretized defect counts (referred as a *discretized defect classifier*) and a classifier that is built using a regression model that is built with non-discretized defect counts (referred as a *regression-based defect classifier*).

Construction of discretized defect classifiers: The continuous defect counts are discretized to defect classes "defective" and "non-defective" based on the condition: If a module's defect count is greater than or equal to 1, it is classified as "defective"; otherwise it is classified as "non-defective". During the training phase, the defect classes are then treated as the target variable and are feed to a classification technique (e.g., random forest) along with the collected features to build the discretized classifier. During the testing phase, the trained discretized classifier is tested on unseen testing data to compute the performance (i.e., AUC) of the classifier and its most influential features.

Construction of regression-based defect classifiers: Different from the construction of a discretized classifier, during the training phase, we use the non-discretized defect counts (i.e., the actual values) to build a regression model then the predicted counts of the model are transformed into two classes ("defective" and "non-defective") based on a threshold which is not necessary to be 1. Then the model performance and feature importance are computed.

E. Performance calculation

We use the *Area Under the receiver operator characteristic Curve (AUC)* as the measure of the performance when comparing between the discretized and regression-based defect classifiers [21]. AUC is computed by plotting the ROC curve, which maps the relation between True Positive Rate (TPR) and False Positive Rate (FPR) at all thresholds. We choose AUC because of following reasons:

1) AUC measures the performance across all the thresholds. When calculating the performance (e.g., precision and recall) of both discretized and regression-based defect classifiers, a threshold needs to be set up to classify the outcome as "defective" if the predicted value is above that threshold and "non-defective" otherwise. It is challenging to decide this threshold. To avoid the problem of threshold setting, we select AUC since AUC measures the performance on all the thresholds (i.e., from 0 to 1) and precludes our analysis from the peculiarities of setting up thresholds.
2) The AUC is insensitive to cost and class distributions [21] so that data imbalances that are inherent to the software datasets is automatically accounted for and provides a score that is objective. An AUC score close to 1 means the classifier's performance is very high and a classifier with an AUC score of 0.5 is no better than random guessing.

We now briefly discuss the calculation of the AUC for the regression-based defect classifier. Figure 3 depicts how the performance calculation component of Figure 1 works. The AUC calculation is accomplished by normalizing the predicted defect counts to fall within the 0 and 1 range to mimic the class probabilities that are generated by a discretized defect classifier. We use the normalized score along with the actual classes to compute the AUC. In this way, we compare the discretized defect and regression-based defect classifiers on a common ground. This normalized score also ensures that the regression-based defect classifier is tested for its classification prowess rather than its regression performance.

F. Feature importance analysis

We use permutation feature importance [3] as a means of measuring the importance of a given feature. Understanding the importance of each feature on a classifier, helps practitioners in their process improvement activities for avoiding future defects. We use permutation importance in lieu of the built-in the feature importance algorithm of each classifier and regression technique since permutation importance gives us a way of conducting feature importance estimation in an unbiased setting.

The permutation importance works by randomly permuting the values of one feature at a time so that the original relationship between the feature and target variable is disturbed. Then this permuted feature along with the other non-permuted features is used to classify the testing data, and performance of the classifier is computed. If the computed performance of the classifier that is built using the permuted feature decreases significantly from the classifier that is built with non-permuted feature, then such performance decrease signifies the importance of this feature. This process is repeated for each of the features and they are ranked based on the degree of decrease in the performance once that particular feature is permuted.

G. Out-of-sample bootstrap

In order to ensure that the conclusions that we draw about our classifiers are robust, we use the out-of-sample bootstrap validation technique, which has been shown to yield the best balance between the bias and variance in a recent study [39]. The out-of-sample bootstrap is conducted along the following steps:

TABLE II: Classifiers that are selected from each family.

Family	Classifier	Regression classifier
Statistical	Logistic regression (Log-Reg)	Linear regression (Lin-Reg)
Random forest	Random forest classification (RF-C)	Random forest regression (RF-R)
Neural networks	Neural Networks classifier (NN-C)	Neural Networks regression (NN-R)
Decision tree	Classification tree (CT)	Regression tree (RT)
Support-Vector machines	SVM classifier (SVM-C)	SVM regression (SVM-R)
Nearest neighbor	K-NN classification (KNN-C)	K-NN regression (KNN-R)

1) A bootstrap sample of size *N* is randomly drawn with replacement from the original dataset, which is also of size *N*.
2) Discretized and regression-based defect classifiers are trained using the bootstrap sample (i.e., training data). On average, 36.8% of the data points will not appear in the bootstrap sample, since it is drawn with replacement [39].
3) We calculate the AUC value and feature importance for each classifier on unseen testing data that are data points that do not appear in the bootstrap sample.

The out-of-sample bootstrap process is repeated 1,000 times. After the bootstrap, 1,000 AUC values and 1,000 lists of the feature importances are generated. We perform further analysis on these AUC values and feature importance to answer our research questions.

IV. Preliminary Study

Prior studies have compared the performance of different machine learning classifiers when building discretized defect classifiers [16], [21]. However, there is no knowledge about the performance of regression-based classifiers. Hence we focus our preliminary study to examine the performance of regression-based classifiers. Our goals are two folds: 1) To replicate prior findings in order to understand whether prior findings for discretized defect classifiers would hold for regression-based classifiers, 2) To help focus our analysis in the following sections on the top performing regression-based classifiers.

Approach: We choose one representative classifier from each widely used machine learning families that are listed by previous studies [15], [21] by satisfying following criteria:

1) A classifier could be built based on both discretized and non-discretized defect counts.
2) One is widely used in prior defect prediction studies.

Table II shows the classifiers chosen for our analysis. All of these classifiers are used at their default settings.

We use the overall experiment setup that is outlined in Section III with all the six families of chosen classifiers. We start with data collection, then correlation and redundancy analysis on the datasets. We then build the discretized and regression-based defect classifiers using the six families of chosen classifiers. The generated classifiers are validated and the performance of each classier is evaluated using the AUC that is obtained from the out-of-sample bootstrap as explained in the Section III.

Once the AUC values are computed, we use a Scott-Knott Effect size clustering (SK-ESD) [39] to rank the classifiers based on the AUC values. SK-ESD uses the effect size as computed by Cohen's Δ [12] to merge statistically similar groups into the same rank. These ranks are obtained for both discretized and regression based defect classifiers on all 17 datasets for each of the six families of chosen classifiers. The average ranks for each classifier over 17 datasets are calculated and the classifier with the lowest rank across both the approaches is considered as the best classifier.

Results: The random forest family has the best performance across both discretized and regression-based defect classifiers. The ranks for each classifier are listed in Table III. Random forest family has the best performance across both discretized (i.e., average rank is 1.17) and regression-based defect classifiers (i.e., average rank is 1.52). This is compatible with the findings of prior studies that suggest that a random forest classifier outperforms other classifiers for building discretized defect classifiers [15], [21]. More specifically, the random forest family performs the best in 14 out of 17 studied datasets for regression-based defect classification and in 11 out of 17 studied dataset for discretized defect classification.

The next best classifier is K-Nearest Neighbor family which has an average rank of 2.52 for both the regression-based and discretized defect classifier, followed by the Statistical classifiers (i.e., Linear and Logistic regressions) which have an average rank of 3.12 for discretized defect classifiers and 1.82 for the regression-based defect classifiers.

We also report the average AUC for each classifier on the studied datasets in Table III. We find that random forest family has an average AUC of 0.78 across both discretized and regression-based defect classifiers, which is the highest among all considered classifiers. The average AUC for linear and logistic regression is 0.73 for regression-based defect classifiers and 0.76 for discretized defect classifiers, which is followed by the KNN family at 0.74 for both types of classifiers.

The random forest family performs the best across both discretized and regression-based defect classifiers. Hence, we primarily focus our analysis in the following sections on the random forest family.

V. Case Study Results

A. *RQ1. How well do regression-based classifiers perform?*

Motivation: In this research question, we investigate the performance of regression-based defect classifiers. Both regression-based and discretized classifiers can identify defect-prone modules. Prior research in software engineering has primarily used discretized classifiers for identifying defect-prone modules. However the discretization that is performed by discretized classifiers leads to information loss. Hence, regression-based classifiers might be a viable option.

TABLE III: Average ranks of various discretized and regression-based classifiers

Project	Lin-Reg		Log-Reg		RF-R		RF-C		NN-R		NN-C		RT		CT		SVM-R		SVM-C		KNN-R		KNN-C	
	R	A	R	A	R	A	R	A	R	A	R	A	R	A	R	A	R	A	R	A	R	A	R	A
Eclipse-2.0	2	0.80	2	0.83	1	0.84	1	0.84	6	0.63	3	0.71	5	0.71	6	0.71	4	0.75	5	0.76	3	0.79	4	0.78
Eclipse-2.1	1	0.79	1	0.79	2	0.78	2	0.77	3	0.73	4	0.67	3	0.73	4	0.67	5	0.62	5	0.62	4	0.72	3	0.71
Eclipse-3.0	1	0.80	1	0.80	1	0.80	1	0.80	5	0.62	2	0.67	3	0.73	4	0.67	4	0.71	5	0.71	2	0.75	3	0.74
Camel-1.2	3	0.60	2	0.61	2	0.63	1	0.64	5	0.55	4	0.56	3	0.55	4	0.56	1	0.64	5	0.64	2	0.59	3	0.58
Mylyn	3	0.68	1	0.70	1	0.74	2	0.68	6	0.54	2	0.60	5	0.62	4	0.60	4	0.65	3	0.65	2	0.70	1	0.70
PDE	2	0.69	1	0.72	1	0.71	2	0.71	5	0.56	5	0.64	4	0.64	4	0.64	4	0.64	6	0.65	3	0.66	3	0.65
Prop-1	4	0.71	2	0.74	1	0.79	1	0.77	5	0.62	4	0.61	5	0.62	4	0.61	3	0.73	3	0.72	2	0.76	1	0.76
Prop-2	4	0.66	3	0.71	1	0.84	1	0.81	5	0.57	5	0.50	5	0.57	5	0.50	3	0.68	4	0.68	2	0.76	2	0.75
Prop-3	3	0.68	2	0.71	1	0.72	4	0.69	6	0.54	1	0.50	5	0.58	6	0.50	4	0.64	5	0.64	2	0.70	3	0.70
Prop-4	2	0.73	1	0.75	1	0.77	2	0.72	5	0.63	5	0.58	5	0.63	5	0.58	4	0.67	4	0.66	3	0.71	3	0.71
Prop-5	3	0.66	2	0.71	1	0.73	3	0.70	5	0.58	1	0.51	4	0.63	6	0.51	3	0.66	5	0.66	2	0.69	4	0.69
Xalan-2.5	5	0.63	3	0.65	1	0.75	1	0.76	6	0.62	3	0.66	4	0.64	3	0.66	2	0.72	4	0.72	3	0.70	2	0.70
Xalan-2.6	3	0.78	2	0.80	1	0.82	1	0.84	5	0.66	3	0.77	4	0.77	3	0.77	2	0.80	4	0.81	3	0.79	2	0.81
Lucene-2.4	2	0.75	2	0.74	1	0.77	1	0.77	5	0.50	5	0.67	4	0.67	4	0.67	3	0.72	6	0.73	3	0.72	3	0.71
Poi-2.5	5	0.76	3	0.81	2	0.85	1	0.89	6	0.50	3	0.80	4	0.79	3	0.80	1	0.86	4	0.86	3	0.82	2	0.84
Poi-3.0	5	0.75	2	0.84	1	0.82	1	0.89	6	0.50	4	0.82	4	0.75	3	0.82	3	0.80	5	0.85	2	0.81	2	0.85
Xerces-1.4	5	0.86	1	0.94	1	0.91	1	0.96	6	0.50	4	0.91	5	0.86	3	0.91	3	0.90	5	0.91	2	0.90	2	0.92
Avg.	3.12	0.73	1.82	0.76	**1.17**	**0.78**	**1.52**	**0.78**	5.29	0.58	3.41	0.66	4.23	0.68	4.18	0.66	3.11	0.72	4.59	0.72	2.52	0.74	2.52	0.74

R - Rank; A - AUC

TABLE IV: Performance Comparison of discretized and regression-based random forest classifiers.

Project	**Avg. AUC of DRFC**	**Avg. AUC of RBRFC**	*p*-**Value**	**Cohen's** d	**DR(%)**
Prop-4	0.72	**0.77**	0	4.27 (L)	9.6
Prop-2	0.81	**0.84**	0	4.32 (L)	10.5
Eclipse-2.1	0.77	**0.78**	0	0.46 (S)	10.8
Prop-3	0.69	**0.72**	0	2.67 (L)	11.5
Mylyn	0.68	**0.74**	0	2.45 (L)	13.2
PDE	0.71	0.71	0	0.12 (N)	14.0
Eclipse-2.0	0.84	0.84	0	0.22 (S)	14.5
Eclipse-3.0	0.80	0.80	0.04	-0.03 (N)	14.8
Prop-1	0.77	**0.79**	0	2.93 (L)	14.8
Prop-5	0.70	**0.73**	0	2.97 (L)	15.3
Camel-1.2	**0.64**	0.63	0	-0.39 (S)	35.5
Xalan-2.6	**0.84**	0.82	0	-0.96 (L)	46.4
Xalan-2.5	**0.76**	0.75	0	-0.39 (S)	48.2
Lucene 2.4	0.77	0.77	0	-0.16 (N)	59.7
Poi 3.0	**0.89**	0.82	0	-2.42 (L)	63.6
Poi 2.5	**0.89**	0.85	0	-1.46 (L)	64.4
Xerces 1.4	**0.96**	0.91	0	-2.43 (L)	74.3

L- Large, S- Small, N- Negligible, DR - Defective Ratio

Approach: To answer this research question, we construct discretized and regression-based classifiers on the 17 studied datasets. Based on our observations in Section IV, we use random forest classifiers since they outperform other types of classifiers.

We then compare the performance of the discretized random forest classifiers (DRFC) and regression-based random forest classifiers (RBRFC) using the AUC values. To measure the differences between the two types of classifiers, we use a Wilcoxon signed-rank test [41] since it does not need the data to follow a normal distribution and it tests paired results. To quantify the magnitude of the performance differences between DRFC and RBRFC, we use Cohen's d effect size test [12]. The threshold for analyzing the magnitude is as follows: $|d| \leq 0.2$ means magnitude is negligible, $|d| \leq 0.5$ means small, $|d| \leq 0.8$ means medium and $|d| > 0.8$ means large.

Results: In contrast to prior studies, building a defect classifier using discretized defect counts does not usually lead to better performance. The comparison of DRFC and RBRFC of all datasets are provided in Table IV. Overall, the Wilcoxon signed-rank test results show that the differences between DRFC and RBRFC are significant on all datasets. Cohen's d results show that the performance differences between DRFC and RBRFC are not negligible for 14 datasets (82%). More specifically, on 7 out of these 14 datasets, DRFC outperforms RBRFC; while, in contradiction to the intuition, RBRFC outperforms DRFC on another 7 datasets.

Regression-based random forest classifiers outperform discretized random forest classifiers when the defective ratio of the dataset data is less than 15% and this trend is reversed when the defective ratio is greater than 35%. To understand how the performance varies among different datasets, we plot the ratio of AUCs of DRFC and RBRFC on the y-axis of Figure 4 and we sort the datasets from low defective ratio to high defective ratio on the x-axis. We observe a consistent trend of RBRFC outperforming DRFC for datasets with a low defective ratio ($< 15\%$) and DRFC outperforms RBRFC when the defective ratio is greater than 35%. It should also be noted that though the difference in average AUC is only a few percentage points, it is statistically significant as highlighted in Table IV.

One possible reason for DRFC having poorer performance than RBRFC on datasets with low defective ratios is that discretized random forest classifiers are known to be impacted by the imbalance in the dataset [10]. It is the fact in our case, findings are suggestive of the fact that, these imbalanced datasets can be better handled by using a regression-based random forest defect classifier in lieu of the traditionally-used discretized classifiers.

The trend of discretized classifiers outperforming regression-based defect classifiers is unique to random forest classifiers. No similar trend is observed for other

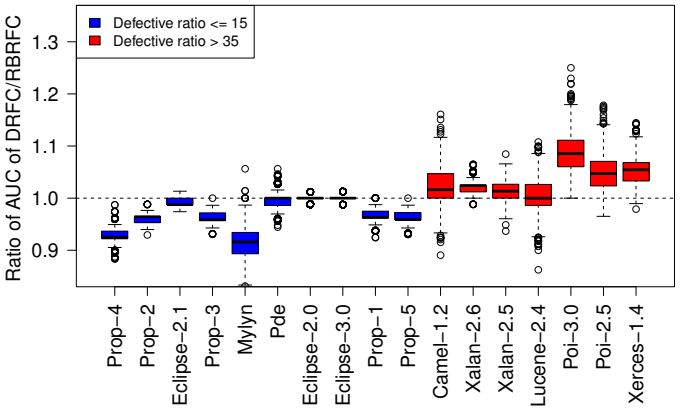

Fig. 4: Ratio of AUC of discretized/regression-based random forest classifiers across different datasets. The datasets are ordered based on their defective ratio from low to high.

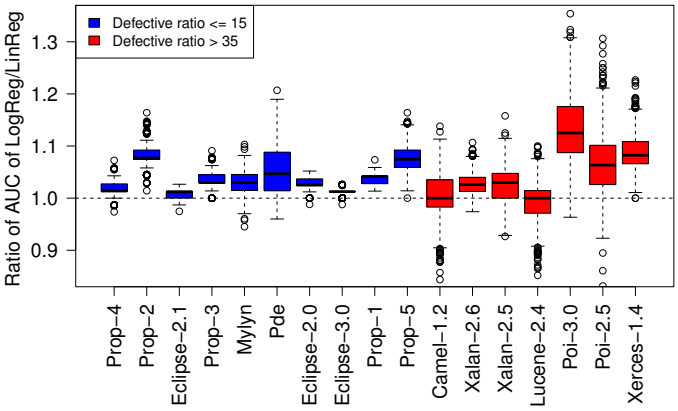

Fig. 5: Ratio of AUC of discretized/regression-based statistical defect classifiers across different datasets. The datasets are ordered based on their defective ratio from low to high.

types of classifiers (e.g., LogReg, LinReg, and KNN). When using other types of classifiers, the discretized classifiers either outperform or perform as good as the regression-based classifiers. For example, Figure 5 shows the ratio of AUCs between discretized and regression-based statistical classifiers. We observe that the medians of the ratios are all above the dashed line which indicates that discretized logistic classifiers always outperform or perform similar toh regression-based linear classifier. More detailed results are available in Table III.

In summary, the common intuition of building a classifier using discretized defect counts is not always correct. To achieve high performance, we advise the use of RBRFC instead of its discretized alternative on datasets with a low defective ratio (i.e., less than 15%) and the use of DRFC on datasets with a high defective ratio. For datasets with a defective ratio between 15% and 35%, we cannot provide suggestions on which classifier to use, since we do not have datasets in that range of defective ratio. To alleviate this problem, we revisit this point in Section VI where we simulate datasets with different defective ratios.

In constrast to the common practice, building a defect classifier using discretized defect counts does not always lead to better performance. RBRFC outperform DRFC when the defective ratio of the dataset is low ($< 15\%$) and the pattern reverses when the defective ratio is high ($> 35\%$).

*B. **RQ2. Are discretized and regression-based classifiers influenced by the same set of features?***

Motivation: Prior studies use defect classifiers to understand the impact of various features (e.g., software metrics) on the likelihood of a module containing a defect [8], [23], [25]. Understanding the most influential features helps practitioners identify process improvement plans and act on them quickly so that the defects could be avoided in future version of a software system. In this paper, we introduce an approach to build defect classifiers using regression. In RQ1, we find that such an approach might lead to better performing classifiers than the traditionally used approach for building classifiers (i.e., discretized classifiers). Hence, in this RQ we wish to examine if these different approaches to build classifiers might produce conflicting information about the most influential features that impact the quality of a software module.

Approach: Similar to the previous research question, we focus on exploring this RQ with DRFC and RBRFC. However, we will provide insights about the other machine learning classifiers whenever appropriate. We follow the approach in Figure 1. We build DRFC and RBRFC on the 17 studied datasets as in RQ1. But instead of generating the AUC values of the classifiers, we generate the feature importance for each feature in each dataset to understand the importance of each feature in identifying the defect-proneness of modules. We use a permutation importance method for generating feature importance scores for both DRFC and RBRFC as outlined in section III. Once the feature importance scores are generated, we rank features using the Scott-Knott ESD test [39]. We then compare the feature importance ranks of DRFC and RBRFC.

To estimate the impact of DRFC and RBRFC on model interpretation, we compute the shifts in the ranks of the features that appear in the top three ranks for both the DRFC and RBRFC classifiers on each dataset. We define rank shift as the amount that a feature shifts its rank between the two classifier in relation to the total number of features in the dataset. Suppose $DRFC(k) = \{var_1, var_2, ..., var_n\}$ and $RBRFC(k) = \{var_1, var_2, ..., var_m\}$ are the features that appear at rank k of $DRFC$ and $RBRFC$. Let PN be the number of features in the given dataset under consideration. We compute the $Shifts(k)$ of a features on rank k between two classifier for a given dataset using the equation (1).

TABLE V: Rank shifts between discretized and regression-based classifiers on various classifiers.

Technique	Rank	Average Shifts	Variance
Random forest	Rank 1	0.04	0.004
	Rank 2	0.07	0.007
	Rank 3	0.16	0.03
Statistical	Rank 1	0.11	0.22
	Rank 2	0.07	0.005
	Rank 3	0.22	0.039
KNN	Rank 1	0.01	0.001
	Rank 2	0.02	0.002
	Rank 3	0.07	0.008

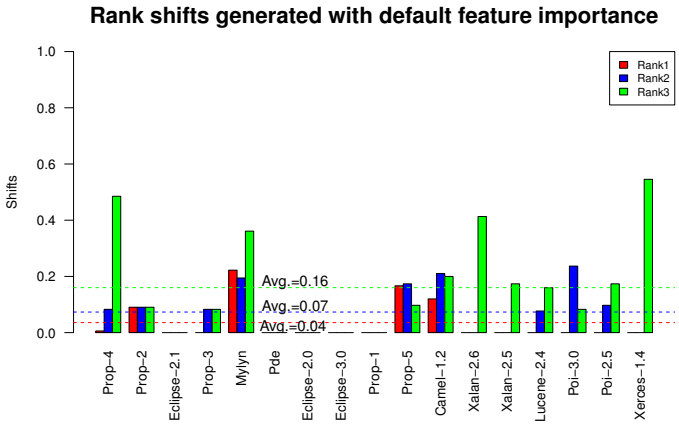

Fig. 6: Rank shifts between DRFC and RBRFC in terms of permutation feature importance across the datasets ordered by defective ratio of the dataset. The mean values of rank shifts are marked with dashed lines.

$$Shifts(k) = (\sum_{var \in DRFC(k)} |k - Rank_{RBRFC}(var)| + \sum_{var \in RBRFC(k)} |k - Rank_{DRFC}(var)|)/PN \quad (1)$$

where $Rank_{RBRFC}(var)$ denotes that rank of var from RBRFC and $Rank_{DRFC}(var)$ denotes the rank of var from DRFC. For example, if the Rank 1 features in the RBRFC are $RBRFC(1) = \{cbo, loc\}$ (i.e., Coupling Between Objects and Lines Of Code), Rank 1 features for $DRFC$ is $DRFC(1) = \{loc\}$ and $DRFC(2) = \{cbo\}$, hence $Rank_{DRFC}(cbo)$ is 2 and NP is 13 for the dataset under consideration, we then compute the $Shifts(1)$ between both classifiers as $1/13 = 0.076$, since only the feature cbo has different ranks across both classifiers. We compute the feature importance shifts for all datasets in a similar fashion. These rank shifts capture the difference in the influential features across the two approaches for building classifiers.

Results: Rank 1 features do not vary significantly between the DRFC and RBRFC classifiers, however the influential features vary significantly at the lower ranks. Figure 6 shows the rank differences between he DRFC and RBRFC classifiers. The DRFC and RBRFC classifiers have exactly the same rank 1 features in 12 out of the 17 datasets (80%). The features at rank 2 and 3 vary drastically since only 8 (47%) and 5 (29%) datasets have the same features at rank 2 and 3, respectively. In terms of rank shift, we observe that the shift between features in rank 1 (i.e., average shift is 0.04) is small and the feature importance varies slightly at rank 2 (i.e., average shift is 0.07). But from rank 3, the shifts start becoming drastic (i.e., 0.16). The dashed horizontal lines that represent the average shift in each rank between the features in Figure 6). We also performed a paired Wilcoxon signed-rank test between the observed shifts and ideal no shift case in which each shifts value is 0 for all datasets. The results show that the shifts at rank 1 are not statistically significant (p-value > 0.05) and shifts at rank 2 and 3 are significant.

We also investigate the rank shifts between discretized and regression-based classifiers for families other than random forest. We present the findings of statistical and KNN classifiers in Table V as they are the next best classifiers after random forests in terms of performance. We find that KNN classifiers exhibit a similar pattern as the random forest classifiers. The feature importance of Rank 1 features does not vary significantly, nevertheless the feature importances start to vary significantly from Rank 2. However, when using statistical classifiers the importance of features shift significantly even at Rank 1.

In summary, although the rank shifts of features appear to be family dependent, random forest has the most stable ranks for its features across both approaches for building defect classifiers. Nevertheless, we recommend that feature importance of the best performing classifier should be used instead of relying solely on the feature importance results that are produced by the discretized classifiers (since such classifiers might fail to accurately capture the studied datasets as observed in some case in RQ1).

The importance of Rank1 features does not vary significantly between discretized and regression-based random forest classifiers. However, lower ranked features vary considerably between types of classifiers. Thus, we suggest practitioners to employ caution on the feature importance variation and use the classifiers with superior performance for model interpretation.

VI. Discussion

This discussion section is evaluated in the context of random forest based discretized and regression-based defect classifiers.

A. How does the performance of discretized and regression-based random forest classifiers vary across different defective ratio?

In RQ1, we observe that the RBRFC outperforms DRFC on data with defective ratio larger than 35% and the observations reverses on the data with defective ratio less than 15%. However, we do not know how the performance of DRFC and RBRFC varies on the data with defective ratio between 15% and 35%. To fill this gap and better understand the relation between the defective ratio and the performance difference

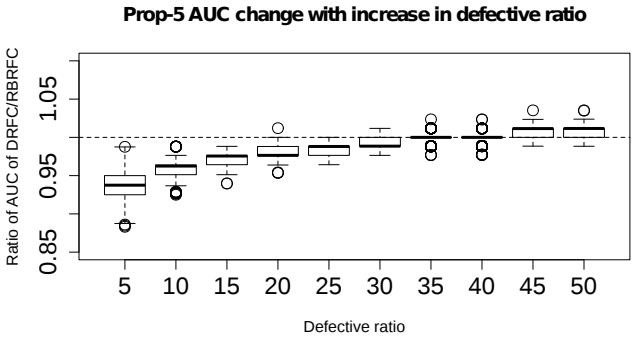

Fig. 7: Boxplot of the ratio of AUC of DRFC/RBRFC on Prop-5.

TABLE VI: Correlation between defective ratio and ratio of AUC of DRFC/RBRFC.

Dataset	Correlation	Dataset	Correlation
Eclipse-2.0	0.90	Eclipse-2.1	0.81
Eclipse-3.0	0.84	Camel-1.2	0.89
Mylyn	0.79	Pde	0.78
Prop-1	0.90	Prop-2	0.88
Prop-3	0.92	Prop-4	0.91
Prop-5	0.95	Xalan-2.5	0.39
Xalan-2.6	0.86	Lucene-2.4	0.51
Poi-2.5	0.78	Poi-3.0	0.56
Xerces-1.4	0.54		

between both approaches for building defect classifiers, we generate datasets with defective ratio ranging from 5% to 50% with 5% interval by re-sampling the studied datasets with replacement (i.e. while keeping the data size fixed). We do this by repeatedly and randomly sampling the datasets with replacement until the datasets have the required defective ratio for our simulation study. Once the datasets at all defective ratios are generated, we followed the experiment setup of RQ1 and analyzed the performance of the generated DRFC and RBRFC for various defective ratios.

We find that as a dataset's defective ratio increases the DRFC classifiers start to outperform the RBRFC classifiers. Table VI shows the spearman correlation (ρ) between the ratio of $AUC of DRFC(dataset)/AUC of RBRFC(dataset)$ and the defective ratio of each dataset. For most datasets (94% – 16 out of 17 datasets), there is indeed a positive and strong correlation (i.e., > 0.5) between the defective ratio and the ratio of the AUC in 16 out of 17 datasets.

For example, we show how the DRFC outperforms RBRFC as the defective ratio of the dataset increases using the "Prop-5" dataset in Figure 7. After the defective ratio crosses 40%, the DRFC outperforms the RBRFC. This study of variation in performance of AUC between DRFC and RBRFC reaffirms our findings in the RQ1 that RBRFC perform better for datasets with low defective ratio whereas the DRFC perform better as the defective ratio of the dataset increases.

Also, we find that the specific point where DRFC start outperforming RBRFC is dataset specific. But as a general rule of thumb, DRFC outperform RBRFC as the defective ratio in the dataset increases. Finally, for datasets with defective ratio between 15% and 35% we suggest practitioners to try both DRFC and RBRFC and use the classifier with superior performance.

TABLE VII: Correlation between AUC and R^2 for the RBRFC classifiers.

Dataset	Correlation	Avg. R^2	Dataset	Correlation	Avg. R^2
Eclipse-2.0	0.14	0.29	Eclipse-2.1	0.33	0.19
Eclipse-3.0	0.19	0.31	Camel-1.2	0.32	0.04
Mylyn	0.28	0.05	Pde	0.21	0.02
Prop-1	0.08	0.05	Prop-2	0.17	0.12
Prop-3	0.21	-0.05	Prop-4	0.18	0.11
Prop-5	0.21	0.10	Xalan-2.5	0.41	0.16
Xalan-2.6	0.45	0.34	Lucene-2.4	0.13	0.33
Poi-2.5	0.37	0.41	Poi-3.0	0.11	0.32
Xerces-1.4	0.07	0.44			

B. Does the R^2 regression fit score impact the performance of regression-based classifiers?

The R^2 regression score explains the variability in the data that is captured by the regression model. Prior studies consider that a higher R^2 is usually associated with better performance and more accurate model interpretation [28]. But as we are using the regression-based classifier for the purposes of classification (regression-based classifier), we find that this assumption no longer holds.

Low R^2 scores for the regression model does not imply that a regression-based classifier will have a low AUC. There is no correlation between R^2 and a classifier's predictive power (i.e., AUC). A low R^2 score does not indicate poor classification performance. Table VII presents the values of R^2 and the correlation between the AUC and R^2 of the RBRFC classifier. Overall, the average AUC across all dataset is good (i.e., 0.78), while the average R^2 is poor (i.e., 0.19). We find that in most of the datasets (88%), the correlation between AUC and R^2 is considered weak (i.e., less than 0.4) [7]. Only two datasets (i.e., Prop-1 and Xerces-1.4), the correlation between AUC and R^2 is considered as moderate [7]. For example, the RBRFC classifier achieves a high AUC 0.84 on Prop-2, while its R^2 is only 0.12 and the correlation between the AUC and the R^2 is 0.17.

C. Permutation feature importance vs. Default feature importance?

In RQ2, we propose permutation feature importance as the method of choice for generating feature importance scores. However, researchers primarily use the default feature importance method that comes along with the implementation of their classifiers. We examine here whether our findings for RQ2 would hold when the default feature importance method is used. We use random forest classifiers for our investigation here. However, the observations hold for all the studied family of classifiers.

Figure 6 and 8 show the rank shifts between permutation and default feature importance methods. We observe that the

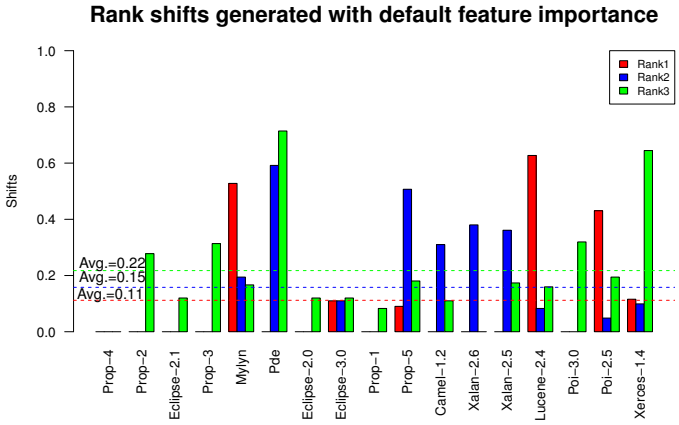

Fig. 8: Rank shifts between DRFC and RBRFC in terms of default feature importance across the datasets. The mean values of rank shifts are marked with dashed lines.

rank shifts of Rank 1 between DRFC and RBRFC are low (i.e., average shift is 0.11) with default feature importance. However, compared with the rank shifts from permutation feature importance as shown in Figure 6, the default feature importance has a higher average rank shift at rank 1. We also conduct the Wilcoxon signed-rank test and it suggests that the feature importances that are computed with default method vary significantly from rank 1. The findings that are observed from permutation and default feature importance methods are still hold from rank 2, although the findings are different at rank 1.

VII. Threats To Validity

We discuss the threats to the validity of our study.

Construct Validity. Threats to construct validity relates to the suitability of our evaluation measures. We have used AUC to evaluate the performance of defect classifiers in our study. While we have explained our reasons for choosing this metric, other evaluation measures may lead different conclusions. For instance, we need to evaluate recall after reading 20% of the lines of code [19] but we present it as an avenue of future research. However, AUC is a well-known metric to evaluate classification models and also widely used in prior studies [21], [39]. We have also performed statistical tests and effect size tests to check if the performance differences between different classifiers are significant and substantial.

In this study, we did not optimize the parameters for the studied classifiers except Neural networks, and CART, as most studied classifiers do not get a significant performance boost with parameters optimization [38]. However, to reduce this threats, future studies should examine the impact of optimized parameters on our findings.

Internal Validity. Prior work shows that incorrect data influences the conclusions drawn from software defect classifiers and potentially biases the results [15]. Even though we tried to control the purity of datasets by imposing conditions on data collection, we cannot ensure that our datasets are correct. To reduce the internal validity, future studies should investigate the correctness of the data further.

External Validity. Threats to external validity relate to the generalizability of our results. In this study, we study 17 datasets and our results may not generalize to other datasets. However, the goal of this paper is not to show a result that generalizes to all datasets, but rather to show that there are datasets where regression-based classifiers would outperform the commonly used discretized classifiers. Nonetheless, additional replication studies may prove fruitful.

VIII. Conclusion

Defect classifiers support software quality assurance efforts in identifying defect-prone modules and allocating quality improvements resources in a timely and cost effective fashion. Traditionally, software defect classifiers are built by discretizing the continuous defect counts of modules into "defective" and "non-defective" classes. However discretization of continuous variables leads to a considerable loss of information. To avoid such information loss, we consider a regression-based classifiers which use the continuous defect counts as the target variable for identifying defect-prone modules.

In this study, we compare discretized and regression-based defect classifiers by applying six machine learning classifiers on 17 open datasets from Tera-PROMISE. We observe that in contrast to current practices in our field, building classifiers using discretized defect counts does not always lead to better performance. Hence future studies should explore both approaches for building classifiers – Given the simplicity of building both types of classifiers, we believe that our suggestion is a rather simple and low-cost suggestion to follow. Moreover, the most influential features vary between the different approaches to build classifiers. Hence future studies should examine the influential factors using the best performing classifier (i.e., discretized or regression-based) instead of simply using discretized classifiers.

IX. Acknowledgment

This study would not have been possible without the High Performance Computing (HPC) systems provided by the Compute Canada and Centre for Advanced Computing (CAC). The research was partially supported by JSPS KAKENHI Grant numbers 15H05306.

References

[1] The promise repository of empirical software engineering data, 2015. http://openscience.us/repo. North Carolina State University, Department of Computer Science.

[2] D. G. Altman and P. Royston. The cost of dichotomising continuous variables. *Bmj*, 332(7549):1080, 2006.

[3] A. Altmann, L. Toloşi, O. Sander, and T. Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

[4] Ö. F. Arar and K. Ayan. Software defect prediction using cost-sensitive neural network. *Applied Soft Computing*, 33:263–277, 2015.

[5] P. C. Austin and L. J. Brunner. Inflation of the type I error rate when a continuous confounding variable is categorized in logistic regression analyses. *Statistics in medicine*, 23(7):1159–1178, 2004.

[6] G. Bavota, M. Linares-Vasquez, C. E. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. The impact of api change-and fault-proneness on the user ratings of android apps. *IEEE Transactions on Software Engineering*, 41(4):384–407, 2015.

[7] S. Boslaugh and P. Watters. *Statistics in a Nutshell: A Desktop Quick Reference*. In a Nutshell (O'Reilly). O'Reilly Media, 2008.

[8] 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*, 35(6):864–878, 2009.

[9] E. Ceylan, F. O. Kutlubay, and A. B. Bener. Software defect identification using machine learning techniques. In *EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 240–247, 2006.

[10] C. Chen, A. Liaw, and L. Breiman. Using random forest to learn imbalanced data. *University of California, Berkeley*, 110, 2004.

[11] J. Cohen. The cost of dichotomization. *Applied psychological measurement*, 7(3):249–253, 1983.

[12] J. Cohen. Statistical power analysis for the behavioral sciences lawrence earlbaum associates. *Hillsdale, NJ*, pages 20–26, 1988.

[13] D. Di Nucci, F. Palomba, S. Siravo, G. Bavota, R. Oliveto, and A. De Lucia. On the role of developer's scattered changes in bug prediction. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 241–250, 2015.

[14] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Syst. J.*, 38(2-3):258–287, 1999.

[15] 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 (ICSE)*, pages 789–800, Piscataway, NJ, USA, 2015. IEEE Press.

[16] L. Guo, Y. Ma, B. Cukic, and H. Singh. Robust prediction of fault-proneness by random forests. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, pages 417–428, 2004.

[17] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 78–88, 2009.

[18] C. Hou, F. Nie, D. Yi, and Y. Wu. Efficient image classification via multiple rank regression. *IEEE Transactions on Image Processing*, 22(1):340–352, 2013.

[19] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan. Revisiting common bug prediction findings using effort-aware models. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[20] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller. Predicting faults from cached history. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 489–498, 2007.

[21] 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*, 34(4):485–496, 2008.

[22] R. C. MacCallum, S. Zhang, K. J. Preacher, and D. D. Rucker. On the practice of dichotomization of quantitative variables. *Psychological methods*, 7(1):19, 2002.

[23] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the Qt, VTK, and ITK projects. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, pages 192–201, 2014.

[24] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann. Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on Software Engineering*, 39(6):822–834, 2013.

[25] A. Mockus. Organizational volatility and its effects on software defects. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, pages 117–126, 2010.

[26] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 181–190, 2008.

[27] R. E. Mullen and S. S. Gokhale. Software defect rediscoveries: a discrete lognormal model. In *Proceedings of International Symposium on Software Reliability Engineering (ISSRE)*, pages 10–pp, 2005.

[28] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 452–461, 2006.

[29] J. Nam and S. Kim. Clami: Defect prediction on unlabeled datasets (t). In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 452–463, 2015.

[30] A. Ng. Lecture notes. cs 229: Machine learning, 2003.

[31] P. Peduzzi, J. Concato, E. Kemper, T. R. Holford, and A. R. Feinstein. A simulation study of the number of events per variable in logistic regression analysis. *Journal of clinical epidemiology*, 49(12):1373–1379, 1996.

[32] D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov. Dual ecological measures of focus in software development. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 452–461, 2013.

[33] F. Rahman and P. Devanbu. How, and why, process metrics are better. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 432–441, 2013.

[34] P. Royston, D. G. Altman, and W. Sauerbrei. Dichotomizing continuous predictors in multiple regression: a bad idea. *Statistics in medicine*, 25(1):127–141, 2006.

[35] G. Scanniello, C. Gravino, A. Marcus, and T. Menzies. Class level fault prediction using software clustering. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 640–645, 2013.

[36] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz. What we have learned about fighting defects. In *Proceedings of International Symposium on Software Metrics (METRICS)*, pages 249–258, 2002.

[37] R. Singh, T. Jaakkola, and A. Mohammad. 6.867 machine learning. fall 2006., 2006.

[38] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 321–332, 2016.

[39] 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 (TSE)*, 2016.

[40] S. Wang and X. Yao. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, 62(2):434–443, 2013.

[41] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[42] S. Xiang, F. Nie, and C. Zhang. Semi-supervised classification via local spline regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(11):2039–2053, 2010.

[43] L. Yu and H. Liu. Efficient feature selection via analysis of relevance and redundancy. *Journal of machine learning research*, 5(Oct):1205–1224, 2004.

[44] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for Eclipse. In *Proceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE)*, page 9, 2007.