On the Use of Evaluation Measures for Defect Prediction Studies

Rebecca Moussa
University College London
London, United Kingdom
rebecca.moussa.18@ucl.ac.uk

Federica Sarro
University College London
London, United Kingdom
f.sarro@ucl.ac.uk

ABSTRACT

Software defect prediction research has adopted various evaluation measures to assess the performance of prediction models. In this paper, we further stress on the importance of the choice of appropriate measures in order to correctly assess strengths and weaknesses of a given defect prediction model, especially given that most of the defect prediction tasks suffer from data imbalance.

Investigating 111 previous studies published between 2010 and 2020, we found out that over a half either use only one evaluation measure, which alone cannot express all the characteristics of model performance in presence of imbalanced data, or a set of binary measures which are prone to be biased when used to assess models especially when trained with imbalanced data.

We also unveil the magnitude of the impact of assessing popular defect prediction models with several evaluation measures based, for the first time, on both statistical significance test and effect size analyses. Our results reveal that the evaluation measures produce a different ranking of the classification models in 82% and 85% of the cases studied according to the Wilcoxon statistical significance test and $\hat{A}_{12}$ effect size, respectively. Further, we observe a very high rank disruption (between 64% to 92% on average) for each of the measures investigated. This signifies that, in the majority of the cases, a prediction technique that would be believed to be better than others when using a given evaluation measure becomes worse when using a different one.

We conclude by providing some recommendations for the selection of appropriate evaluation measures based on factors which are specific to the problem at hand such as the class distribution of the training data, the way in which the model has been built and will be used. Moreover, we recommend to include in the set of evaluation measures, at least one able to capture the full picture of the confusion matrix, such as MCC. This will enable researchers to assess whether proposals made in previous work can be applied for purposes different than the ones they were originally intended for. Besides, we recommend to report, whenever possible, the raw confusion matrix to allow other researchers to compute any measure of interest thereby making it feasible to draw meaningful observations across different studies.

CCS CONCEPTS

• **Software and its engineering** → **Software defect analysis**;

KEYWORDS

Software Defect Prediction, Evaluation Measures

ACM Reference Format:
Rebecca Moussa and Federica Sarro. 2022. On the Use of Evaluation Measures for Defect Prediction Studies. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3533767.3534405

1 INTRODUCTION

Software bugs are costly. The most common cost of bugs is poor user experience, which causes software abandonment. For example, one of the most prominent reasons mobile users delete an application, often after a single use, is due to the app crashing [28]. In the worst-case scenario, their cost can be life-threatening as in the case of the IT glitch of the UK National Health Service which put 10,000 patients at risk of being given the wrong medication in 2018 [8]. The earlier a bug is found and fixed, the less it costs.

Software defect prediction research aims to support engineers in identifying defective components, early in the development process. An ideal prediction model is the one able to unveil as many defects as possible without raising false alarms (i.e., flagging clean components as defective).

However, as shown by Zhang et al. [56], achieving this in practise is challenging as optimising for one aspect often compromises the other (especially for problems such as the ones with large negative vs. positive ratios). On the other hand, defect prediction models with high detect capabilities and high false alarm, or vice-versa models with low detective capabilities but high precision, can still considered effective depending on the business context [32, 33]. For example, a defect prediction model for safety critical software can be considered effective if it exhibits a high probability of detecting defects, even if it does so at the cost of generating a large number of false alarms. Similarly, a model that sacrifices certain detective capabilities in order to achieve a better precision can be desirable when there is, for example, a high cost in checking false alarms.

The use of appropriate evaluation measures guides practitioners and researchers to understand whether a given prediction model is fit for their purposes [17].

However, previous work has raised alarms about the way researchers have employed these measures to assess the effectiveness of the prediction models proposed in their work, especially in the presence of imbalanced data [4, 6, 12, 17, 20, 48]. The importance of adopting suitable measures has been often overlooked, thereby leading to discordant empirical results (i.e., conclusion instability) and hindering meta-analysis across different studies [9, 34, 47].

In this paper we invite the community to reflect better on the selection of appropriate evaluation measures to support the scientific conclusions that are drawn on the effectiveness of defect prediction models.

To this end, we first analyse how researchers have selected and used these measures over the last decade by examining 111 defect prediction studies published between 2010 and 2020. We find that 59% of these studies do not properly motivate the use of their evaluation measures depending on the business context, and less than half acknowledge that their findings might change if the results are assessed by using a different evaluation measure. Furthermore, we find that, despite the warnings raised in previous studies [4, 17, 24], the use of some problematic measures has become more frequent with time. On the other hand, no growth in the adoption of more robust measures has been seen.

We also unveil and quantify the impact using different measures might have in practice by carrying out a comprehensive empirical study. Specifically, we investigate the use of six of the most widely used evaluation measures in literature to assess and compare the performance of seven popular defect prediction models in predicting defects for 15 different real-world software systems (for total of 24 datasets), under three different prediction scenario.

We find that there is no case where all the measures agree on a same ranking of prediction models. Moreover, in 118 (83%) and in 122 (85%) out of the 144 cases analysed, the ranking produced by a given evaluation measure varies from the rankings produced by all the other evaluation measures, according to the Wilcoxon statistical significance test and the $\hat{A}_{12}$ the effect size measure, respectively. Besides, we find that assessing model performance based on a given measure would have changed the rank of a specific technique between 61% and 90% of the time, on average, depending on the measure used.

Overall, these results highlight the dramatic impact on the ability to draw meaningful conclusions across studies using different measures often not relevant or suitable to the business context.

We encourage researchers to select evaluation measures that fit the study's specific aim, model and data; as well as to include a more comprehensive and balanced measure to give an overall view of the performance of the proposed approach. This enables researchers and practitioners to assess and decide whether proposals made in previous work can be applied for purposes different than the ones they were originally intended for. The rest of the paper is organised as follows. We first provide the reader with some background on previous work discussing the matters arising from the use of different evaluation measures in defect prediction studies (Section 2), and with a comprehensive overview of the different metrics used in the literature (Section 3). The core contributions of our study are presented in Sections 4 to 6. In Section 4 we report our findings on the use of evaluation measures in 111 defect prediction studies published over the last decade. Whereas in Section 5 we describe the design of the empirical study we conducted and discuss its results in Section 6. Section 7 discusses possible threats to the validity of our study. Section 8 concludes the paper and presents some recommendation for the selection of evaluation measures in future defect prediction studies.

2 RELATED WORK

A few studies have highlighted possible differences resulting from the use of different evaluation measures, however no previous study has provided empirical evidence on the magnitude of such differences nor its statistical significance, and the effect it can potentially have on findings across various studies. In the following, we discuss these studies and highlight further differences with ours.

In 2008, Jiang et al. [24] provided a review of evaluation measures for defect prediction commonly used at that time, as well as an initial comparison of their use on the NASA data. They highlighted possible threats coming from the use of certain/different measures and suggested that evaluation measures should be carefully chosen and interpreted based on the specific needs of the project. Twelve years later, our analysis of the work published between 2010 and 2020 reveals that the threat has remained un-tackled by the community.

Moreover, our empirical study includes measures proposed more recently, as well as a more diverse and recent set of data and for the first time, the comparison is entirely based on statistical significant tests and the effect size measure.

Subsequently, Jingxiu and Shepperd [54] performed a meta-analysis of eight papers on defect prediction in order to understand the differences resulting from the use of F-measure as opposed to MCC. They illustrated potential biases by using confusion matrices that portray different scenarios and found that the use of F-measure is problematic. However they did not quantify the differences resulting from the comparison, in fact as they state, their study "captures a change in direction of the effect, it does not, however, capture the magnitude of the effect"[54]). In our study, we specifically study the magnitude of the effect of using six different evaluation measures (including F-measure and MCC) based on both statistical and effect size analyses. This analysis is crucial to provide solid empirical evidence on whether the use of a measure over another significantly changes the way model performance is interpreted with respect to the business needs.

Other studies not directly targeting this issue yet highlighting that the problem exists are those by Arisholm et al. [4], Xuan et al. [51], and Hall et al. [17]. Arisholm et al. [4] investigated and compared the use of different classifiers and features to predict cross-release defects of an industrial legacy system. To this end, they used different evaluation measures (including Accuracy, Precision, Recall and ROC), and observed that what is considered the best model depends on the criteria that are used to evaluate and compare the models.

Xuan et al. [51] came to a similar conclusion by investigating the performance of several classifiers for within-project defect prediction in 10 open-source software systems, based on a large number of evaluation measures in order to find the best performing classifier. The comprehensive literature survey by Hall et al. [17] reviewed defect prediction studies published up to 2010. Despite the fact that the primary goal of their survey was not to investigate bias resulting from the use of different evaluation measures, they do observe that this is an issue and provide some guidelines to prevent it. As this study was published in 2012, one would have expected subsequent research to adopt/follow these guidelines, however, our analysis of the work published in the last decade shows that this has not been the case as we further articulate in Section 4.

3 A HITCHHIKER'S GUIDE TO DEFECT PREDICTION EVALUATION MEASURES

In this section we describe the most common binary classification evaluation measures, highlighting strengths and weaknesses of their use for defect prediction.

The formulae of these measures are reported in Table 1 and they originate from the confusion matrix (see Table 2). This matrix describes four types of instances: TP, defective modules correctly classified as defective; FP, non-defective modules wrongly classified as defective; FN, defective modules wrongly classified as non-defective; TN, non-defective modules correctly classified as non-defective.

The Accuracy measure has been one of the first measures used to assess defect prediction models performance, but it is nowadays widely recognised that this is a biased measure for defect prediction models and thus should not be used. The reason is that this measure is very sensitive to class imbalance and defect prediction data is very often imbalanced [24]. A simple trick to maximize accuracy when data is imbalanced is to always predict the instances as non-defective.

Subsequently, Precision and Recall and their harmonic mean, the F-measure (F1), have been adopted in numerous studies. These measures consider the positive (i.e., defective) class as the only class of interest. Similarly, False Positive Rate (FPR), which is also known as Type I Error Rate, is also a single-focus measure and, as such, it explains only one aspect of a classifier.

While, this can be acceptable in some domains, like information retrieval, where the number of irrelevant documents not correctly retrieved is hard to quantify (i.e., it is essentially unbounded), it would not be acceptable in the medical domain, for example, where classifying a sick person as healthy is just as important as classifying a healthy person as sick. The latter can also be the case for defect prediction since the correct identification of negative classes (non-defective) becomes important when the cost of inspecting the components incorrectly classified as defective (i.e., false alarms) is high. These examples suggest that the choice of assessing a model with Precision, Recall, F1 or (FPR) might vary according to the business needs.

The knowledge of the business domain can indeed guide in choosing the most appropriate way to evaluate whether a given prediction model is effective for the problem at hand. In other words, the relative importance assigned to precision and recall is an aspect of the problem. Weighted measure can be used to control such a delicate balance. An example is the Fβ measure, which can be used to control the balance of precision and recall by setting a β coefficient: $F\beta = ((1+\beta^2) * Precision * Recall)/(\beta^2 * Precision + Recall)$. However, determining meaningful weights is not trivial in practice [24].

Using only measures that give importance to only one class, might lead to biased evaluation when assessing prediction models in presence of highly imbalanced data [32]. In such a context, a good classifier is expected to produce high accuracy in detecting the defect class without significantly degrading the accuracy of detecting the non-defect class [20]. In other words, probability of detection (or Recall) and probability of false alarm (or FPR) are important performance metrics in the class imbalance context, and it is usually recommended the use of balanced measures, such as G-measure and G-mean2, to properly assess model's performance. These measures give equal importance to the positive and the negative class, i.e., they are formulated based taking into account both probability of detection and probability of false alarm, and thereby show how much balance between two metrics is achieved overall.

Similarly, the distance to heaven (d2h) measure (a.k.a. Balance = 1 - d2h) computes the distance of FPR and Recall to the ideal (heaven) values of FPR=0 and Recall=1. Therefore, the smaller d2h (higher Balance), the better the performance of the classifier. Based on the definition of d2h, it is clear that a high Recall leads to a lower d2h; whereas a high FPR results in a higher d2h.

Ranking measures have also been used to assess prediction models. The most popular one is the Area Under the ROC[1] Curve (AUC), which can be interpreted as the probability that a model ranks a random positive observation higher than a random negative observation. The AUC is a reliable heuristic to evaluate and compare the overall performance of classification models as it is scale-invariant (i.e., it measures how well predictions are ranked, rather than their

[1]The Receiver Operating Characteristic (ROC) is a graph showing model's performance in terms of True Positive Rate and False Positive Rate at all classification thresholds.

Evaluation Measure	Definition
AUC	Area under the Receiver Operating Characteristic Curve
Recall (PD)	$\frac{TP}{TP+FN}$
Precision	$\frac{TP}{TP+FP}$
F1	$2 \times \frac{Precision \times PD}{Precision+PD}$
FPR (a.k.a PF)	$\frac{FP}{FP+TN}$
G-measure	$\frac{2 \times PD \times (1-PF)}{PD+(1-PF)}$
Balance	$1 - \frac{\sqrt{(0-FPR)^2+(1-PD)^2}}{\sqrt{2}}$
MCC	$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
G-mean	$\sqrt{\text{PD} \times (1-\text{FPR})}$

Table 1: The definition of the measures.

	Predicted Value	
Actual Value	Defective	Non-Defective
Defective	True Positive (TP)	False Negative (FN)
Non-Defective	False Positive (FP)	True Negative (TN)

Table 2: Confusion Matrix for Binary Classification.

absolute values) and threshold-invariant (i.e., it measures the quality of the model's predictions irrespective of the exact classification threshold chosen). However, it is not applicable in practice since a deployed classifier must have a particular classification threshold [18]. Thus, unless a classifier outperforms another for all possible threshold values, AUC cannot be used to compare and rank classifiers [36, 45].

A different approach based on statistics, is the phi-coefficient (ϕ) [13], also known as Matthews Correlation Coefficient (MCC) when applied to classifiers [31]. This approach considers the true class and the predicted class as two (binary) variables, and computes their correlation coefficient, in a similar way to the computation of the correlation coefficient between any two variables. The higher the correlation between true and predicted values, the better the prediction. MCC has the nice property of being perfectly symmetric, i.e., no class is more important than the other, and switching the positive and negative class yields to a same MCC value. Since MCC takes into account all four values in the confusion matrix, a coefficient close to 1 means that both the positive and negative classes are predicted accurately, even when the data is imbalanced.

The use of MCC allows to capture overall model performance, and thus it is often recommended for evaluating the performance of classification models across different tasks or in the presence of imbalanced data. Previous studies have argued that using MCC is more appropriate than using F1, (or AUC), alone for defect prediction studies [46, 54], yet our survey, (described in Section 4), shows that MCC has not been widely adopted yet (see Figure 2). A detailed comparison of MCC, F1 and AUC can be found elsewhere [12, 15, 46, 54].

The above measures are easy to compute, and this has generated the use of a variety of different measures over the years, even if they were not always related to the specific model usage requirements as we further articulate in the next section.

4 INVESTIGATING THE USE OF EVALUATION MEASURES IN THE DEFECT PREDICTION LITERATURE

As discussed in Section 2, the use of different evaluation measures has raised a concern within the defect prediction community that dates back to over 10 years ago [24], but *what has changed since?* and *how have we handled this crucial aspect in our studies*? In the following we aim at answering these questions by reviewing prior studies published over the last decade.

4.1 Search Methodology

We used Scopus to search for research articles published between 2010 and 2020 (June 2nd) by using the query: ("Defect" || "Fault" || "Bug") & ("Prediction" || "Prone"). This search resulted in 242 papers. Among these, we manually filter out irrelevant publications by following the three step process adopted in previous surveys [22, 30] as described below:

(1) **Title:** First, we exclude all publications whose title clearly does not match our inclusion criteria;
(2) **Abstract:** Second, we examine the abstract of every remaining publications. Publications whose abstract does not meet our inclusion criteria are excluded at this step;
(3) **Body:** We then read, in full, all publications that had passed the previous two steps. Manuscripts are excluded if their content does not satisfy the inclusion criteria or contribute to this survey.

To ensure that the publications included in this survey are relevant to the context of binary defect prediction, at each of the above steps we apply the following inclusion criteria:

- The publication should investigate an experimental study of software defect prediction models, metrics or data.
- The publication predicts a dichotomous outcome (i.e., defect or not defect-prone).

Based on the above three-stage process and inclusion criteria, we iteratively reduce the amount of publications obtained from the Scopus search, until we end up with a set of 111 publications investigated herein. For the sake of space, we provide the full list of papers in our online appendix [1].

We then manually examine the 111 papers to extract relevant information. In particular, in order to identify which evaluation measures were used in each study, and the rational for them, we analyze the section discussing the evaluation measure/procedure.

We consider a study to have explained its use of measures when it provides a clear reasoning for the choice of measures as opposed to others, or when otherwise, the reason is obvious from the research context.

Finally, we proceed by checking whether the study mention any challenge arising from the choice of the evaluation measure(s), and hence whether any mitigation was put in place. We consider a study to have acknowledged and mitigated the importance of choosing proper evaluation measures if it discusses, for example, limitations of the used measure, the existence of measures other than the ones used, the fact that different measures might yield different results, the necessity to complement the used measures with additional ones, etc.

4.2 Results

We report the number of papers that use a specific measure in Table 3 as well as the number of studies that use one or multiple measures in Figure 1, and the frequency of measure use over time in Figure 2.

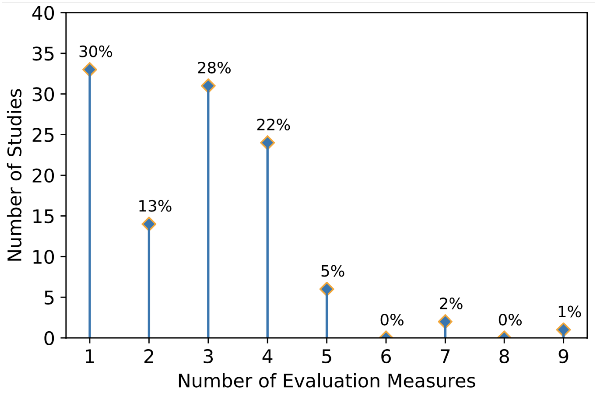


Figure 1: Number of measures used in prior studies.

Evaluation Measure	No. of Studies (%)
AUC	60 (54%)
F1	59 (53%)
Recall (PD)	57 (51%)
Precision	43 (39%)
FPR (a.k.a. PF)	19 (17%)
G-measure	12 (11%)
Balance	10 (9%)
MCC	13 (12%)
Accuracy	12 (11%)
G-mean	9 (8%)
Others	10 (9%)

Table 3: Number of studies using a given measure.

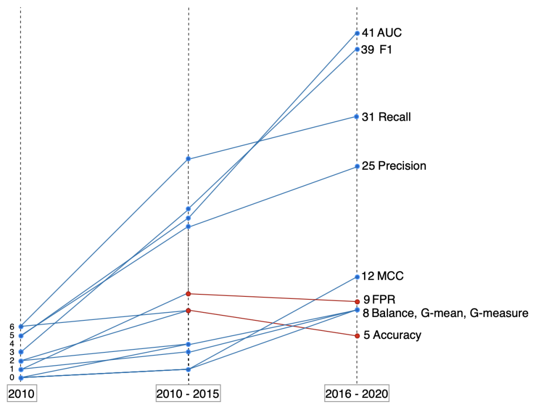


Figure 2: The frequency of measure use for 2010-2020.

We can observe, from Table 3, that more than nine different evaluation measures have been used to assess defect prediction models over the past 10 years.

The most used measures are AUC and F1 (used in 60% and 59% of papers, respectively), followed by Recall and Precision (used in 57% and 43% of papers, respectively). As explained in Section 3, measures such as AUC and F1 do not portray the full confusion-matrix and can lead to biased results when used for assessing classifier performance on unbalanced datasets, which is often the case for defect prediction data. A variety of symmetric and robust measures exist [45], however we found (see Table 3) that they have been adopted by much fewer studies (e.g., MCC 12%, G-measure 11%, G-mean 8%, Balance 9%).

Moreover, as shown in Figure 1, 30% of the papers analysed use only one measure to evaluate defect prediction models, among which AUC is the most common (53%), followed by F1 (33%).

Using only one measure might affect the validity of these studies especially if such measure is among those that have been shown to be problematic (see Section 3). This is the case of AUC and F1, which might be biased when imbalanced data is used, as further explained in Section 3.

The threat might still be present in past studies that use more than one measure. Indeed, among the 14 studies (13%) which use two measures, four of them (29%) use AUC and F1 together; and among the 31 papers (28%) using three measures, 13 of them (42%) use F1 and its constituent components (i.e., Recall and Precision). Furthermore, when analysing the studies that use four measures, 10 out of 24 (42%) use a set that may still be biased as it includes a subset of Accuracy, F1, Precision, Recall, and AUC. This is somehow worrying since these measures are known to be problematic [14, 18, 24, 41], as explained in Section 3. Their use should be complemented with the use of balanced measures such as G-measure or MCC.

Together, these results highlight that 56 out of the 111 papers examined (51%) might have drawn different conclusions had they considered a more comprehensive set of measures.

Further, when observing the use of the evaluation measures over the years, presented in Figure 2, results show that despite the warnings raised in previous studies about the use of AUC, F1, and its constituents [14, 18, 24, 41], their use has actually become more frequent with time. On the other hand, we do not observe as big of a growth in the adoption of more robust measures advocated in previous work [45]. For example, only one study has used MCC before 2016 and it has then been used at an average of less than three times per year since.

Despite the warnings raised in previous work [4, 12, 17, 48], the results of our literature review reveal that the choice of evaluation measures is still a major concern in the community and that it has only been partially tackled so far. Over half of the studies (59%) do not justify the use of the measures, based on the aim of the study, neither the models investigated or the data at hand. 52% of studies also do not acknowledge that using different measures could lead to different results. Among those studies which acknowledge the threat (48%), 41 put in place some form of mitigation. However, only 15 out of the 41 studies recommend the use of unbiased measures, while the majority propose mitigation which might be perceived as unsatisfactory since they recommend, for example, that future work should investigate the use of other measures, or the use of measures widely seen in previous work, or the use of threshold independent measures, which has been criticised for not being applicable in practice since to deploy a classifier one must use a specific classification threshold [18, 45] as we explain in Section 3.

5 EMPIRICAL STUDY DESIGN

In this section, we present a thorough empirical study based on both, statistical tests and effect size to investigate the extent to which the conclusion of a study comparing the performance of defect prediction techniques may change based on the evaluation measure used to make the comparison.

This study investigates two research questions (RQs) and involves the use of seven classification techniques, 15 publicly available datasets, and six evaluation measures validated on three different defect prediction scenarios, as detailed in the following subsections.

5.1 Research Questions

We pose two research questions investigating the use of the evaluation measures most used in prior studies (i.e., AUC, Balance, F1, G-measure, MCC, and FPR according to the literature [2] to measure the performance of seven different prediction techniques, described in Section 5.5.

Our first research question investigates, precisely, whether each of the six evaluation measures would rank the seven techniques in the same order. This is crucial because if these measures turn out to yield different rankings, then the conclusions made by previous studies using certain measures would change at a significant level. Therefore, we first ask:

RQ1. Ranking Disagreement: *How often would the ranking of techniques produced by a given measure differ from the ranking produced by another measure?*

To address RQ1, we first produce a ranking of the prediction techniques per measure according to the Wilcoxon Signed-Rank test and the Vargha and Delaney's $\hat{A}_{12}$ non-parametric effect size [3]. Then, based on these rankings, we compute the *ranking disagreement* for each measure by counting the number of times it produce a same ranking with respect to using other measures.

The *ranking disagreement* tells us the extent to which a measure agrees with others. If the rank disagreement of a given measure is high, this would suggest that there is a non-trivial error when using different measures.

Our second research question analyses this error at a finer grain to gather further insights on the extent to which these rankings differ:

RQ2. Rank Disruption: *What is the percentage of cases in which a rank of a specific technique, based on a given measure, changes when assessed using the other measures?*

To assess the *rank disruption* of a ranking provided by a given evaluation measure, we assess the rank change, i.e., a prediction technique that would be believed to be better than others when using a given evaluation measure becomes worse when using a different measure. If the *rank disruption* proves to be high, some scientific conclusions drawn in previous studies could be reversed when accounting for the threat to validity posed by the choice of the evaluation measure.

We explain in details the way we compute the *rank disagreement* and *rank disruption* in Section 5.2.

5.2 Ranking Disagreement and Rank Disruption

To compute the ranking disagreement and the rank disruption, we run the prediction models on each dataset and evaluate their performance with each of the six evaluation measures under investigation. Then we apply the Wilcoxon test and the Vargha and Delaney's $\hat{A}_{12}$ non-parametric effect size, and rank the prediction models, per measure, based on the results of these analyses. Once the rankings are obtained we compute:

- the *ranking disagreement* of a given measure by analysing the number of times its ranking varies from the rankings produced by each of the other evaluation measures per dataset. The *ranking disagreement* of a given measure varies from 0 (all measures agree on the same ranking) to 1 (all measures disagree, thus each ranking is unique).
- the *rank disruption* of a given measure by counting the number of times the rank of a specific technique changes when assessed according to each of the other measures.

[2]We do not want to inflate our results, therefore we do not include Accuracy, Recall and Precision given they would generate opposing rankings due to the way they are defined.

	M_1	M_2	M_3
Rank 1	T_a	T_c	T_c
Rank 2	T_b, T_c	T_a	T_b
Rank 3	-	T_b	T_a

Table 4: An example of the procedure used to compute the ranking disagreement and the rank disruption.

In the following, we explain in detail how we computed the rankings based on the Wilcoxon test and $\hat{A}_{12}$ effect size.

Given an evaluation measure, we perform the Wilcoxon Signed-Rank test ($\alpha < 0.05$) on the results obtained by each pair of techniques by testing the null hypothesis: "The results achieved by $predictionModel_x$ in terms of a given evaluation measure m are worse than those achieved by $predictionModel_y$". We, then, summarise the results of this test by using the following win-tie-loss procedure [42]:

We count the number of times a prediction model scored a p−value<0.05 (win), p−value>0.95 (loss), and 0.05≤ p−value ≥0.95 (tie). Finally, we rank the techniques based on the highest number of wins, where any ties are broken based on the number of losses. For example, based on the win-tie-loss procedure, Table 4 shows that according to measure M_1, technique T_a is the best performing one since it achieves the highest number of wins when compared to techniques T_b and T_c. Whereas, technique T_c is said to be the best performing one according to measures M_2 and M_3 given that this technique obtains the highest number of wins. However, all three measures (i.e., M_1, M_2 and M_3) disagree on the second and third rank. We also produce a ranking of the prediction models based on the Vargha and Delaney's $\hat{A}_{12}$ non-parametric effect size [3] to validate whether the differences highlighted by Wilcoxon test are worthy of interest.

Similarly to the procedure described above, we compute the $\hat{A}_{12}$ effect size for each pair of techniques resulting in a statistically significant difference according to the Wilcoxon Signed-Rank test ($\alpha < 0.05$). The $\hat{A}_{12}$ statistic measures the probability that an algorithm A yields better values for a given performance measure M than that of another algorithm B. If the two algorithms are equivalent, then $\hat{A}_{12} = 0.5$. Given the first algorithm performing better than the second, $\hat{A}_{12}$ is considered small for $0.6 \leq \hat{A}_{12} < 0.7$, medium for $0.7 < \hat{A}_{12} < 0.8$, and large for $\hat{A}_{12} \geq 0.8$, although these thresholds are somewhat arbitrary [43]. Based on these thresholds, we count the number of small, medium and large effect sizes obtained by each prediction model, and rank the techniques based on the highest number of large effect size, where ties are broken based on the number of medium and small effect sizes in this order.

Following the above procedure, for each dataset we obtain a set of six rankings (i.e., one per evaluation measure) based on the Wilcoxon test and a set of six rankings based on the $\hat{A}_{12}$. For each of these sets, we manually inspect the rankings produced by each of the measures to compute the ranking disagreement and rank disruption, as explained above.

5.3 Datasets

To answer RQ1, we carry out a thorough empirical study using a total of 15 datasets available from two public repositories: NASA [40] and Realistic [55]. The former has been made publicly available in the early 2000 and contains software defect data about industrial NASA software systems, which have been widely used by the defect prediction community since. The latter was made publicly available in 2019 and consists of open-source projects characterised by a wider set of features than the ones extracted from the NASA data.

All data used herein was curated in previous work by following rigorous processes and made publicly available [40, 55]. In particular, we use six NASA datasets from the work of Petrić et al. [40], who applied rules to clean erroneous data from the original NASA repository [44], and data about nine open-source software systems, collected by Yatish et al. to ensure robust defect counts [55].

These datasets have different nature and size: They vary in application domain, age, total number of modules (ranging from 94 to 8,847) and features (ranging from 38 to 65), and percentage of defective modules they contain as summarised in Table 5.

5.4 Validation Scenarios

Our experimental design tries to reflect as much as possible the scenarios where defect prediction has been applied so far. To this end we consider three common validation scenarios: Within-Project Defect Prediction, Cross-Version Defect prediction and Cross-Project Defect Prediction.

Repository	Dataset	Modules (% defective)
NASA	CM1	296 (12.84%)
	KC3	123 (13.00%)
	MW1	253 (10.67%)
	PC1	661 (7.87%)
	PC3	1043 (12.18%)
	PC5	94 (19.15%)
Realistic	Activemq_5.3.0	2367 (10.90%)
	Activemq_5.8.0	3420 (6.02%)
	Camel_2.10.0	7914 (2.91%)
	Camel_2.11.0	8846 (2.17%)
	Derby_10.3.1.4	2206 (30.33%)
	Derby_10.5.1.1	2705 (14.16%)
	Groovy_1_6_beta_1	821 (8.53%)
	Groovy_1_6_beta_2	884 (8.60%)
	Hbase_0.95.0	1669 (22.95%)
	Hbase_0.95.2	1834 (26.34%)
	Hive_0.10.0	1560 (11.28%)
	Hive_0.12.0	2662 (8.00 %)
	Jruby_1.5.0	1131 (7.25%)
	Jruby_1.7.0	1614 (5.39%)
	Lucene_3.0	1337 (11.59%)
	Lucene_3.1	2806 (3.81%)
	Wicket_1.3.0-beta2	1763 (7.37%)
	Wicket_1.5.3	2578 (4.07%)

Table 5: Datasets used in our empirical study.

For the NASA datasets, given that each system consists of a single version, we investigate the Within-Project scenario applying the Hold-Out validation process where the data is randomly split into 80% training set and the other 20% which is used for testing. This procedure is repeated 30 times in order to reduce any possible bias resulting from the validation splits [3].

For the Realistic datasets, given that each system consists of multiple releases we investigate both the Cross-Version and Cross-Project scenarios. In the Cross-Version scenario, for each of the software systems, we train on one release and test on a different one, i.e., we train on version v_x and test on version v_y, where $x < y$ as done in previous work (see e.g., [19]). In the Cross-Project scenario, for each of the software systems, we consider the version with the higher release number as the test set and train the model on the union of the versions of the other datasets with a lower release number. In both scenarios, the versions used as train and test sets are not subsequent releases nor are they the system's most recent ones. In addition, there is always a window of at least five months between these releases. This reduces the likelihood of the snoring effect or unrealistic labelling as noted in previous studies [2, 5, 25].

5.5 Techniques

In order to study the magnitude of any discrepancy resulting from the use of different evaluation measures, we consider a variety of techniques, namely Dummy, Logistic Regression (LR), Naïve Bayes (NB), Random Forest (RF) and Support Vector Machine (SVM) implemented in `Scikit-learn 0.20.2` (`Python 3.6.8`), belonging to different classifier families. These are widely studied techniques in the defect prediction literature [17, 21]. Indeed, the chosen classifiers are also representative of those employed by the surveyed papers (see Section 4), as we found that LR is used in 67% of the surveyed papers (74/111), NB in 62% (69/111), RF in 47% (52/111) and SVM in 38% (42/111). We briefly describe them below:

Dummy is a simple baseline which generates predictions uniformly at random. We use the Scikit-learn's `dummy` class.

Naïve Bayes is a statistical classifier that uses the combined probabilities of the different attributes to predict the target variable, based on the principle of Maximum a Posteriori [50]. We use the Scikit-learn's `naive_bayes` class.

Logistic Regression is a statistical technique, introduced as an extension to linear regression, which models the probabilities for problems with a dichotomous dependent variable [29]. We use the Scikit-learn's `linear_model` class.

Random Forest is a decision tree-based classifier consisting of an ensemble of tree predictors [10]. It classifies a new instance by aggregating the predictions made by its decision trees. Each tree is constructed by randomly sampling N cases, with replacement, from the N sized training set. RF is known to perform well on large datasets and to be robust to noise. We use the RF classifier from the `ensemble` class in Scikit-learn. We also perform hyperparameter tuning, denoted as RF_T, where we explore different values for the following parameters: min_samples_leaf = [1, 20], min_samples_split = [2, 20], max_leaf_nodes = [default=None, 10, 50], and n_estimators = [50, 75, 100, 125, 150].

Support Vector Machine is a widely known classification technique [11]. A linear model of this technique uses a hyperplane in order to separate data points into two categories. However, in many cases there might be several hyperplanes that can correctly separate the data. Hence, SVM seeks to find the hyperplane that has the largest margin, in order to achieve a maximum separation between the two categories. When the data is not linearly separable, SVM does the mapping from input space to feature space by using a kernel function instead of an inner product. In this work, we use Scikit-learn's `svm` class. We also perform hyperparameter tuning for this technique, denoted as SVM$_T$, where we try different values for C and γ with the Radial Basis Function kernel. Specifically, we explore the following grids: C = [0.01, 0.5, 1, 512, 8000, 32000] and γ = [1/n_features, 0.000001, 0.0001, 0.001, 0.01, 0.1, 1, 2, 4, 8].

6 EMPIRICAL STUDY RESULTS

In this section we report and discuss the results we obtained carrying out the empirical study described in Section 5.

6.1 RQ1. Ranking Disagreement

To address RQ1 we compare the performance of Dummy, LR, NB, RF, RF$_T$, SVM, SVM$_T$ experimenting with the six most widely used evaluation measures in literature (shown in Table 3). The results of this comparison, based on both statistical significance test and effect size, are summarised in Tables 6 (Within-Project scenario) and 7 (Cross-Version and Cross-Project scenarios), and discussed in detail in the rest of the section. Each table reports whether a given measure disagrees with *"All"*, *"More than half*, *"Less than a half"*, or *"None"* of the other measures considered on a given dataset in our study[3].

When looking at the ranking of techniques obtained by each evaluation measure, based on the Wilcoxon test, our results show that across all scenarios (i.e., out of the 24 datasets) examined there is no case where the use of each measure leads to a same ranking of the techniques, i.e., the ranking disagreement is never 0. On the other hand, the number of times the ranking disagreement = 1 (i.e., a measure provides a unique ranking) is quite high as it adds up to a total of 119 out of the 144 cases analysed (83%). In the remaining cases, the ranking disagreement is always either greater or equal to 0.60.

Similar observations hold when analysing the results based on the Vargha and Delaney's $\hat{A}_{12}$ non-parametric effect size measure. These results also show that there is no case where all the measures produce the same ranking (i.e., ranking disagreement is never 0). When computing the number of times the ranking disagreement = 1, results show that this happens very frequently, specifically on a total of 123 out of the 144 cases studied (85%). The ranking disagreement on the remaining cases is always either equal or greater than 0.6. Below we discuss the specific results obtained in each of the scenarios.

When looking at the results obtained for the Within-Project scenario based on the Wilcoxon test, we observe that each of the measures produces a unique ranking on two out of the six NASA datasets (see Table 6). Whereas, four out of six measures produce a unique ranking on the other four datasets and only two (i.e., Balance and G-measure) agree with each other on the same ranking.

[3]Note that for the sake of space, we omit the columns *"None"* and *"Less than a half"* as our results show that there are no such cases.

		>Half		*All*	
		Wilcoxon	**Eff. Size**	**Wilcoxon**	**Eff. Size**
CM1	F1			✓	✓
	G-meas.			✓	✓
	MCC			✓	✓
	Balance			✓	✓
	AUC			✓	✓
	FPR			✓	✓
KC3	F1		✓	✓	
	G-meas.	✓	✓		
	MCC			✓	✓
	Balance	✓			✓
	AUC			✓	✓
	FPR			✓	✓
MW1	F1		✓	✓	
	G-meas.		✓	✓	
	MCC			✓	✓
	Balance			✓	✓
	AUC			✓	✓
	FPR			✓	✓
PC1	F1			✓	✓
	G-meas.	✓			✓
	MCC			✓	✓
	Balance	✓			✓
	AUC			✓	✓
	FPR			✓	✓
PC3	F1			✓	✓
	G-meas.	✓			✓
	MCC			✓	✓
	Balance	✓			✓
	AUC			✓	✓
	FPR			✓	✓
PC5	F1			✓	✓
	G-meas.	✓			✓
	MCC			✓	✓
	Balance	✓			✓
	AUC			✓	✓
	FPR			✓	✓

Table 6: RQ1. Ranking disagreement results for the Within-Project scenario. For each dataset, we report whether a given evaluation measure disagrees with more than a half, or all the other measures, based on statistical significance and effect size analyses

An even stronger discordance than that shown by the Wilcoxon test is observed when analysing the results based on the $\hat{A}_{12}$ non-parametric effect size measure for this scenario. We found an agreement between two measures only (i.e., F1 and G-measure) on only two datasets out of the six under study. Whereas, each measure produces a unique ranking in all other cases (i.e., 34 out of 36) considered.

Based on the results for the Cross-Version scenario, shown in Table 7, we observe that, according to the Wilcoxon statistical significance test, each of the measures produces a unique ranking for four out of out nine datasets, while on the remaining ones, only two measures agree (G-measure and Balance on three datasets, F1 and Balance on one, and F1 and MCC on the other one). In the case of the analysis based on the Â12 effect size measure, each of the evaluation measures studied produces a unique ranking on five datasets. Similarly to the results observed by the Wilcoxon test, a maximum of two evaluation measures (the same ones denoted by the Wilcoxon test) agree on the remaining datasets. Specifically, G-measure and Balance agree on three of the remaining four datasets while F1 and Balance agree on another dataset and F1 and MCC on the remaining one.

The results based on the Wilcoxon test, in the Cross-Project scenario (shown in Table 7) are more discordant compared to those observed in the cross-version scenario. A unique ranking is produced by each measure on six out of the nine dataset studied. On two out of the remaining three datasets, only two measures agree on a single ranking (i.e., Balance and G-measure). Whereas on the remaining dataset, Balance, F1 and G-measure agree on a ranking while the other measures provide unique ones. In addition, the $\hat{A}_{12}$ effect size, in line with the results based on the Wilcoxon Test, shows more discordant results compared to the cross-version scenario. While no measure agrees on any ranking for six of the datasets (i.e., each of the measures provides a unique ranking), at most two evaluation measures (i.e., Balance and G-measure) agree on a ranking for two of the remaining three datasets. As for the remaining dataset, results show that Balance, F1 and G-measure agree on the same ranking whereas the other evaluation measures produce unique rankings.

		Cross-Version				Cross-Project			
		>Half		*All*		*>Half*		*All*	
		Wilcoxon	**Eff. Size**	**Wilcoxon**	**Eff. Size**	**Wilcoxon**	**Eff. Size**	**Wilcoxon**	**Eff. Size**
ActiveMQ	F1			✓	✓			✓	✓
	G-meas.			✓	✓			✓	✓
	MCC			✓	✓			✓	✓
	Balance			✓	✓			✓	✓
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓
Camel	F1			✓	✓			✓	✓
	G-meas.			✓	✓			✓	✓
	MCC			✓	✓			✓	✓
	Balance			✓	✓			✓	✓
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓
Derby	F1			✓	✓	✓	✓		
	G-meas.	✓	✓			✓	✓		
	MCC			✓	✓			✓	✓
	Balance	✓	✓			✓	✓		
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓
Groovy	F1			✓	✓			✓	✓
	G-meas.	✓	✓					✓	✓
	MCC			✓	✓			✓	✓
	Balance	✓	✓					✓	✓
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓
HBase	F1			✓	✓			✓	✓
	G-meas.			✓	✓			✓	✓
	MCC			✓	✓			✓	✓
	Balance			✓	✓			✓	✓
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓
Jruby	F1	✓	✓					✓	✓
	G-meas.			✓	✓			✓	✓
	MCC	✓	✓					✓	✓
	Balance			✓	✓			✓	✓
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓
Lucene	F1			✓	✓			✓	✓
	G-meas.	✓	✓			✓	✓		
	MCC			✓	✓			✓	✓
	Balance	✓	✓			✓	✓		
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓
Hive	F1	✓	✓			✓	✓		
	G-meas.			✓	✓			✓	✓
	MCC			✓	✓			✓	✓
	Balance	✓	✓			✓	✓		
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓
Wicket	F1			✓	✓			✓	✓
	G-meas.			✓	✓			✓	✓
	MCC			✓	✓			✓	✓
	Balance			✓	✓			✓	✓
	AUC			✓	✓			✓	✓
	FPR			✓	✓			✓	✓

Table 7: RQ1. Ranking disagreement results for the Cross-Version and Cross-Project scenarios. For each dataset, we report whether a given evaluation measure disagrees with *more than a half*, or *all* the other measures, based on statistical significance and effect size analyses.

	Dataset	AUC	Balance	F1	FPR	G-meas.	MCC
Within-Project	**CM1**	69%	69%	74%	66%	66%	86%
	KC3	80%	71%	86%	71%	86%	91%
	MW1	63%	63%	69%	71%	71%	91%
	PC1	57%	49%	74%	49%	66%	71%
	PC3	77%	66%	91%	66%	83%	80%
	PC5	71%	66%	69%	66%	86%	77%
	Average	70%	64%	77%	65%	76%	83%
Cross-Version	**ActiveMQ**	69%	80%	71%	71%	80%	86%
	Camel	83%	69%	80%	71%	80%	86%
	Derby	86%	63%	80%	63%	89%	94%
	Groovy	94%	71%	77%	71%	91%	80%
	Hbase	71%	83%	74%	94%	91%	100%
	Hive	60%	66%	60%	60%	71%	100%
	JRuby	69%	80%	69%	74%	83%	89%
	Lucene	86%	63%	74%	63%	77%	100%
	Wicket	80%	80%	69%	80%	74%	86%
	Average	77%	73%	73%	72%	82%	91%
Cross-Project	**ActiveMQ**	86%	71%	77%	80%	89%	100%
	Camel	69%	66%	71%	71%	89%	80%
	Derby	51%	51%	83%	51%	89%	86%
	Groovy	77%	66%	77%	69%	89%	91%
	Hbase	71%	66%	69%	66%	97%	100%
	Hive	77%	71%	69%	71%	94%	91%
	JRuby	57%	83%	60%	66%	66%	86%
	Lucene	69%	63%	86%	63%	100%	94%
	Wicket	89%	74%	91%	80%	97%	89%
	Average	72%	68%	76%	69%	90%	91%

Table 8: RQ2. Percentage of rank disruption per measure based on statistical significance analysis.

Answer to RQ1: There is no case where all evaluation measures produce the same ranking. In fact, the evaluation measures produce a unique ranking in 83% and 85% of the cases according to the analysis based on the Wilcoxon test and $\hat{A}_{12}$ effect size, respectively. There are only very few cases where at most three (less than 1% according to both statistical and effect size analyses) or two evaluation measures (8% and 5% according to the Wilcoxon test and $\hat{A}_{12}$ effect size, respectively) provide the same ranking, but even in those cases it is not always the same set of evaluation measures that agree on a ranking.

6.2 RQ2. Rank Disruption

Tables 8 and 9 report the rank disruption values obtained per measure per dataset for all scenarios when analysed based on the Wilcoxon Test and Vargha and Delaney's $\hat{A}_{12}$ effect size results, respectively.

For the Within-Project scenario, the average rank disruption across the NASA datasets varies between 64% (Balance) and 83% (MCC) on average when basing the results on the Wilcoxon test

	Dataset	AUC	Balance	F1	FPR	G-meas.	MCC
Within-Project	**CM1**	94%	83%	74%	74%	80%	80%
	KC3	63%	63%	86%	63%	89%	89%
	MW1	66%	66%	74%	69%	74%	91%
	PC1	77%	77%	80%	71%	77%	91%
	PC3	83%	74%	83%	77%	83%	97%
	PC5	60%	57%	57%	69%	100%	69%
	Average	74%	70%	76%	70%	84%	86%
Cross-Version	**ActiveMQ**	69%	86%	71%	71%	77%	83%
	Camel	83%	69%	80%	71%	80%	86%
	Derby	74%	63%	74%	63%	83%	94%
	Groovy	86%	69%	77%	69%	83%	86%
	Hbase	71%	83%	74%	94%	91%	100%
	Hive	57%	69%	60%	57%	74%	100%
	JRuby	66%	89%	66%	77%	83%	89%
	Lucene	69%	57%	80%	57%	77%	100%
	Wicket	77%	71%	71%	74%	77%	91%
	Average	72%	73%	73%	70%	81%	92%
Cross-Project	**ActiveMQ**	83%	77%	77%	86%	86%	94%
	Camel	71%	66%	60%	63%	91%	77%
	Derby	49%	49%	77%	49%	89%	89%
	Groovy	71%	57%	71%	71%	86%	83%
	Hbase	71%	66%	71%	71%	94%	100%
	Hive	80%	66%	80%	66%	91%	97%
	JRuby	54%	83%	57%	63%	69%	80%
	Lucene	69%	63%	86%	63%	100%	94%
	Wicket	89%	74%	91%	80%	97%	89%
	Average	71%	67%	75%	68%	89%	89%

Table 9: RQ2. Percentage of rank disruption per measure based on effect size analysis.

and between 70% (Balance) and 86% (MCC) when the analysis is based on $\hat{A}_{12}$, depending on the evaluation measure used.

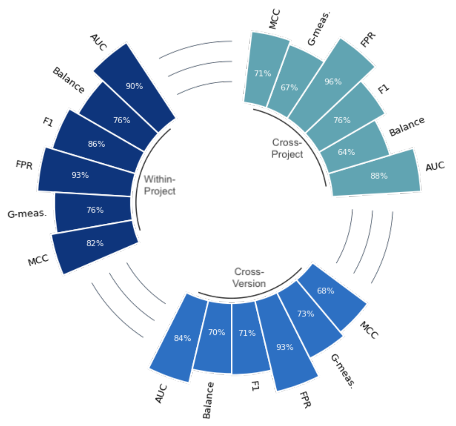
Similarly for the Cross-Version scenario the average rank disruption across the nine Realistic datasets ranges from 72% (FPR) to 91% (MCC) based on the Wilcoxon test and from 70% (FPR) to 92% (MCC) based on $\hat{A}_{12}$.

The rank disruption is still very high when looking at the Cross-Project scenario, with average results across the nine systems ranging from 68% (Balance) to 91% (MCC) based on the Wilcoxon test and from 67% (Balance) to 89% (G-measure and MCC) based on $\hat{A}_{12}$.

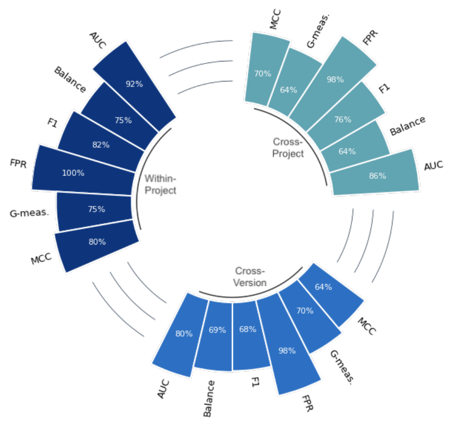
We can conclude that both analyses (i.e., Wilcoxon test and $\hat{A}_{12}$) provide similar rank disruption results for all the three scenarios.

We also perform a more fine grained analysis of our results to get a better understanding of the impact the use of these different evaluation measures can have on the conclusion validity of a study. We therefore examine the rank disruption of each evaluation measure when considering the top three ranked techniques, then considering the top two ranked and then the first ranked technique, for each scenario investigated. Figures 3 and 4 show the average rank disruptions of the measures, across all datasets, for each scenario based on statistical test and effect measure analyses, respectively.

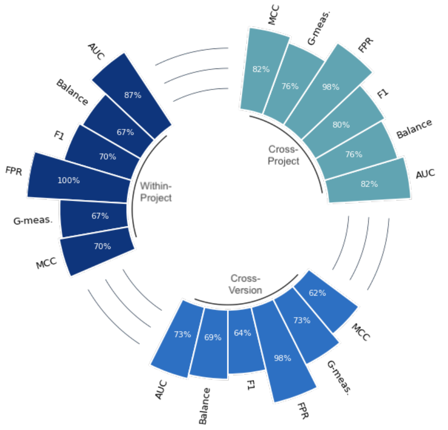
Top Three Ranked Techniques: We can observe from Figure 3 that the rank disruption remains very high when considering the top three ranked techniques for each scenario. Specifically, according to the Wilcoxon test (shown in Figure 3c), the rank disruption ranges from 76% (Balance and G-meas.) to 93% (FPR) when evaluating



(a) Rank disruption of top three ranked techniques

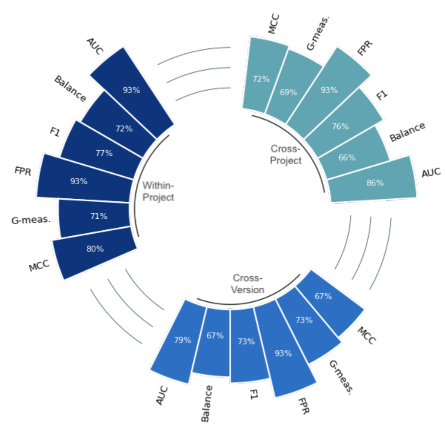


(b) Rank disruption of top two ranked techniques

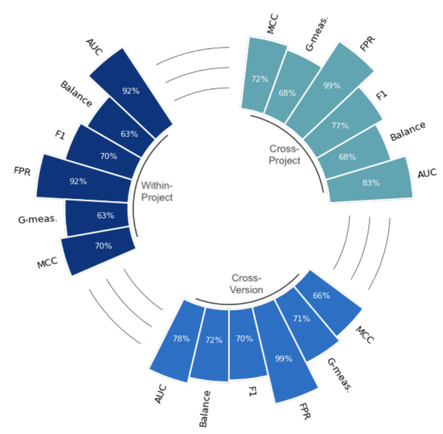


(c) Rank disruption of top ranked technique

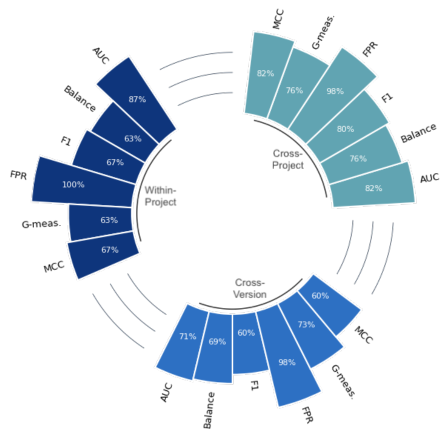
Figure 3: RQ2. Rank disruption (average across all datasets) of each evaluation measure for the top three techniques (a), top two (b) and first ranked technique (c) for each scenario investigated based on statistical significance analysis.



(a) Rank disruption of top three ranked techniques



(b) Rank disruption of top two ranked techniques



(c) Rank disruption of top ranked technique

Figure 4: RQ2. Rank disruption (average across all datasets) of each evaluation measure for the top three techniques (a), top two (b) and first ranked technique (c) for each scenario investigated based on effect size analysis.

the Within-Project scenario, from 68% (MCC) to 93% (FPR) in the Cross-Version scenario and from 64% (Balance) to 96% (FPR) in the Cross-Project scenario. Similar observations hold when analysing the rank disruption based on $\hat{A}_{12}$, with its values ranging between 71% (G-meas.) and 93% (AUC and FPR) for the Within-Project scenario, between 67% (Balance and MCC) and 93% (FPR) when evaluating predictions across versions (i.e., Cross-Version scenario) and between 66% (Balance) and 93% (FPR) in the Cross-Project scenario.

Top Two Ranked Techniques: This high level of rank disruption is also consistent when considering the top two ranked techniques only. Figure 3b shows that, based on the Wilcoxon text, the rank disruption ranges between 75% (Balance and G-meas.) to 100% (FPR) when evaluating the Within-Project scenario, between 64% (MCC) and 98% (FPR) in the Cross-Version scenario, and between 64% (Balance and G-meas.) and 98% (FPR) when evaluating predictions across projects (i.e., Cross-Project scenario). The analysis based on $\hat{A}_{12}$ also agrees with that of the statistical significance test. Figure 4b reports that the average rank disruption, across all datasets, ranges between 63% (Balance and G-meas.) and 92% (AUC and FPR) in the Within-Project scenario. When considering the Cross-Version and Cross-Project scenarios, the average rank disruption is a bit higher, with its values ranging between 66% (MCC) and 99% (FPR) and between 68% (Balance and G-meas.) and 99% (FPR), respectively.

Top Ranked Technique: High values of rank disruption are also be observed when only the top ranked technique is considered. The average rank disruption based on the the Wilcoxon test (see Figure 3a) varies between 67% (Balance and G-meas.) and 100% (FPR) in the Within-Project scenario, between 62% (MCC) and 98% (FPR) in the Cross-Version scenario, and between 76% (Balance and G-meas.) and 98% (FPR) in the Cross-Project scenario. Similar to previous analyses, the results based on the effect size measure agree with those based on the Wilcoxon test. As shown in Figure 4c, the rank disruption based on the $\hat{A}_{12}$ ranges between 63% (Balance and G-meas.) and 100% (FPR) in the Within-Project scenario, between 60% (F1 and MCC) and 98% (FPR) when evaluations are carried out on predictions of different versions within a same software project (i.e., Cross-Version scenario), and between 76% (Balance and G-meas.) and 98% (FPR) in the Cross-Project scenario.

Answer to RQ2: The rank disruption for each of the measures investigated is high on average, ranging from 64% to 92% depending on the measure and validation scenario. Our results also show that high disruption is also present when investigating the top ranked techniques only, with an average rank disruption ranging between 60% and 100% depending on the validation scenario, the measure and the number of top ranked techniques considered.

7 THREATS TO VALIDITY

We discuss below how we mitigate possible threats to the internal, construct and external validity of this study.

Internal Validity: To mitigate the threat of missing relevant work or information in our literature review, we have specified the query we used for our search, defined clear inclusion criteria and followed a rigours procedure recommended and used in previous work to filter out irrelevant articles [23, 30]. Although we cannot and do not claim that the set of 111 studies we investigates is exhaustive, it is reasonable to believe that it is representative of the current state-of-the-art. As a matter of fact, we note that two prominent defect prediction literature surveys [17, 45] published in IEEE TSE in 2012 and 2014, reviewed, respectively, 36 and 42 papers, while in this study we analyse 111 articles. The gathering and filtering procedure was performed by both authors, to ensure reliability and reduce researcher bias.

Construct Validity: We carefully calculated the performance measures used in the study, and applied the statistical tests, verifying all the required assumptions.

External Validity: As happens with most empirical studies, the subjects used in our study might not be representative of the whole population. However, we have designed our study aiming at using measures, techniques, datasets, and validation scenarios, which are as representative as possible of the defect prediction literature. First of all, we focused on binary evaluation measure as these are the most common in literature [17] and we strove to consider the measures most used in previous work (see Section 4). To increase the relevance to the defect prediction literature we also investigated class level defect binary prediction models in three validation scenarios (i.e., within-project, cross-release and cross-project) widely investigated in the literature [27, 47] as opposed, for example, to more recent (and therefore less studied) ones such as effort-aware models [7, 35, 37, 53] or binary prediction at line level [49], method level [39] or commit/change level [16, 26, 38, 52]. Similarly, we considered traditional classification techniques widely used in previous studies [17] as described in Section 5.5.

Moreover, we used a publicly available implementation of these techniques provided by the `Scikit-learn` library, to reduce possible biases and errors arising from the use of ad-hoc implementations. We also used publicly available datasets previously investigated in the literature, which are of different nature and size, and which have been carefully curated in previous work as explained in Section 5.3. We plan to make our scripts and data publicly available on-line [1] upon acceptance, to facilitate the replication and extension of our work.

8 CONCLUSIONS

In this paper, we investigate the effect of using different evaluation measures for comparing the performance of software defect prediction models.

Our review of previous work published over the last decade, has revealed that the majority of the studies do not provide rationale for the measures used with regard to the characteristic of the datasets and/or aim of their study. Moreover, they often use measures which only partially reflect the performance of defect prediction models, and that the measures used are often the most susceptible ones to data imbalance. Further, our empirical study reveals that different evaluation measures provide unique rankings in 82% and 85% of the cases studied according to the Wilcoxon test and effect size measure, respectively. Moreover, the rank disruption for each of the measures investigated is high (ranging from 61% to 90% on average depending on the measure and validation scenario). These results suggest that in the majority of the cases, a prediction technique that would be believed to be better than others when using a given evaluation measure becomes worse when using a different one. Our results also show that the percentage of disruption when only considering the top ranked techniques is just as significant as when all ranks are studied, rendering the results even more striking.

We hope that the empirical evidence provided herein on the significant differences that arise from the use of evaluation measures will encourage the community to act upon this matter, and carefully select the measures based on factors which are specific to the problem at hand [17]. These include (1) the class distribution of the training data; (2) the way in which the model will be used; (3) the way in which the model has been built. Moreover, we recommend to include in the set of evaluation measures, at least one able to capture the full picture of the confusion matrix (i.e., the correctly and incorrectly classified instances), such as MCC, so that it is possible to assess whether proposals made in previous work can be applied for purposes different than the ones they were originally intended for. Besides, we recommend to report, whenever possible, the raw confusion matrix from which the results were extracted as this can enable other researchers to compute any measure of interest and facilitate them to draw meaningful observations across different studies.

ACKNOWLEDGMENTS

Rebecca Moussa and Federica Sarro are supported by the ERC Advanced fellowship grant EPIC (741278) and the Department of Computer Science of University College London.

REFERENCES

[1] [n. d.]. On-line appendix - On the Use of Evaluation Measures for Defect Prediction Models. https://github.com/SOLAR-group/dpevalmeasures
[2] Aalok Ahluwalia, Massimiliano Di Penta, and Davide Falessi. 2020. On the Need of Removing Last Releases of Data When Using or Validating Defect Prediction Models. *arXiv preprint arXiv:2003.14376* (2020).
[3] Andrea Arcuri and Lionel Briand. 2014. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *STVR* 24, 3 (2014), 219–250.
[4] Erik Arisholm, Lionel C Briand, and Eivind B Johannessen. 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *JSS* 83, 1 (2010), 2–17.
[5] Abdul Ali Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. 2020. On the time-based conclusion stability of cross-project defect prediction models. *Empirical Software Engineering* 25, 6 (2020), 5047–5083.
[6] Mohamed Bekkar, Hassiba Kheliouane Djemaa, and Taklit Akrouf Alitouche. 2013. Evaluation measures for models assessment over imbalanced data sets. *J Inf Eng Appl* 3, 10 (2013).
[7] K. E. Bennin, K. Toda, Y. Kamei, J. Keung, A. Monden, and N. Ubayashi. 2016. Empirical Evaluation of Cross-Release Effort-Aware Defect Prediction Models. In *Procs. of QRS*. 214–221.
[8] Henry Bodkin. 2019. https://www.telegraph.co.uk/news/2018/10/19/nhs-blunder-put-10000-patients-risk-wrong-prescription/
[9] David Bowes, Tracy Hall, and David Gray. 2012. Comparing the performance of fault prediction models which report multiple performance measures: recomputing the confusion matrix. In *Procs. of PROMISE*. 109–118.
[10] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
[11] Christopher JC Burges. 1998. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery* 2, 2 (1998), 121–167.
[12] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the MCC over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21, 1 (2020), 6.
[13] Harald Cramir. 1946. Mathematical methods of statistics. *Princeton U. Press* (1946).
[14] Peter Flach and Meelis Kull. 2015. Precision-recall-gain curves: PR analysis done right. In *Advances in neural information processing systems*. 838–846.

[15] George Forman and Martin Scholz. 2010. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD* 12, 1 (2010), 49–57.

[16] Takafumi Fukushima, Yasutaka Kamei, Shane McIntosh, Kazuhiro Yamashita, and Naoyasu Ubayashi. 2014. An Empirical Study of Just-in-Time Defect Prediction Using Cross-Project Models. In *Procs. of MSR*. 172–181.

[17] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2011. A systematic literature review on fault prediction performance in software engineering. *IEEE TSE* 38, 6 (2011), 1276–1304.

[18] David J Hand. 2009. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine learning* 77, 1 (2009), 103–123.

[19] Mark Harman, Syed Islam, Yue Jia, Leandro L Minku, Federica Sarro, and Komsan Srivisut. 2014. Less is more: Temporal fault predictive performance over multiple hadoop releases. In *Procs. of SSBSE*. Springer, 240–246.

[20] Haibo He and Edwardo A. Garcia. 2009. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering* 21, 9 (2009), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

[21] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2017. A comparative study to benchmark cross-project defect prediction approaches. *IEEE TSE* 44, 9 (2017), 811–833.

[22] Max Hort, Maria Kechagia, Federica Sarro, and Mark Harman. 2021. A Survey of Performance Optimization for Mobile Applications. *IEEE TSE* (2021).

[23] Max Hort, Maria Kechagia, Federica Sarro, and Mark Harman. 2021. A Survey of Performance Optimization for Mobile Applications. *IEEE TSE* (2021).

[24] Yue Jiang, Bojan Cukic, and Yan Ma. 2008. Techniques for evaluating fault prediction models. *EMSE* 13, 5 (2008), 561–595.

[25] Matthieu Jimenez, Renaud Rwemalika, Mike Papadakis, Federica Sarro, Yves Le Traon, and Mark Harman. 2019. The importance of accounting for real-world labelling when predicting software vulnerabilities. In *Procs. of ESEC/FSE*. 695–705.

[26] Yasutaka Kamei, Takafumi Fukushima, Shane McIntosh, Kazuhiro Yamashita, Naoyasu Ubayashi, and Ahmed E. Hassan. 2016. Studying just-in-time defect prediction using cross-project models. *ESE* 21, 5 (2016), 2072–2106.

[27] Y. Kamei and E. Shihab. 2016. Defect Prediction: Accomplishments and Future Challenges. In *Procs. of SANER*. 33–45.

[28] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan. 2015. What Do Mobile App Users Complain About? *IEEE Software* 32, 3 (2015), 70–77.

[29] David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. 2002. *Logistic regression*. Springer.

[30] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2016. A survey of app store analysis for software engineering. *IEEE TSE* 43, 9 (2016), 817–847.

[31] Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *BBA-Protein Structure* 405, 2 (1975), 442–451.

[32] Tim Menzies, Alex Dekhtyar, Justin Distefano, and Jeremy Greenwald. 2007. Problems with Precision: A Response to "Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'". *IEEE Transactions on Software Engineering* 33, 9 (2007), 637–640. https://doi.org/10.1109/TSE.2007.70721

[33] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering* 33, 1 (2007), 2–13. https://doi.org/10.1109/TSE.2007.256941

[34] Tim Menzies and Martin J. Shepperd. 2011. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering* 17 (2011), 1–17.

[35] Mariam El Mezouar, Feng Zhang, and Ying Zou. 2016. Local versus Global Models for Effort-Aware Defect Prediction. In *Procs. of CASCON*. 178–187.

[36] Sandro Morasca and Luigi Lavazza. 2020. On the assessment of software defect prediction models via ROC curves. *ESE* 25, 5 (2020), 3977–4019.

[37] C. Ni, X. Xia, D. Lo, X. Chen, and Q. Gu. 2020. Revisiting Supervised and Unsupervised Methods for Effort-Aware Cross-Project Defect Prediction. *IEEE TSE* (2020).

[38] Luca Pascarella, Fabio Palomba, and Alberto Bacchelli. 2019. Fine-grained just-in-time defect prediction. *JSS* 150 (2019), 22–36.

[39] Luca Pascarella, Fabio Palomba, and Alberto Bacchelli. 2020. On the performance of method-level bug prediction: A negative result. *JSS* 161 (2020), 110493.

[40] Jean Petrić, David Bowes, Tracy Hall, Bruce Christianson, and Nathan Baddoo. 2016. The jinx on the NASA software defect data sets. In *Procs. of EASE*. 1–5.

[41] David Martin Powers. 2011. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. (2011).

[42] Federica Sarro, Mark Harman, Yue Jia, and Yuanyuan Zhang. 2018. Customer rating reactions can be predicted purely using app features. In *Procs. of RE*. IEEE, 76–87.

[43] Federica Sarro, Alessio Petrozziello, and Mark Harman. 2016. Multi-objective software effort estimation. In *Procs. of ICSE*. 619–630.

[44] J. Sayyad Shirabad and T.J. Menzies. 2005. The PROMISE Repository of Software Engineering Databases. http://promise.site.uottawa.ca/SERepository

[45] Martin Shepperd, David Bowes, and Tracy Hall. 2014. Researcher bias: The use of machine learning in software defect prediction. *IEEE TSE* 40, 6 (2014), 603–616.

[46] Q. Song, Y. Guo, and M. Shepperd. 2019. A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction. *IEEE TSE* 45, 12 (2019), 1253–1269.

[47] B. Turhan, T. Zimmermann, F. Shull, L. Layman, A. Marcus, A. Butcher, D. Cok, and T. Menzies. 2013. Local versus Global Lessons for Defect Prediction and Effort Estimation. *IEEE TSE* 39, 06 (2013), 822–834.

[48] Mauno Vihinen. 2012. How to evaluate performance of prediction methods? Measures and their interpretation in variation effect analysis. In *BMC genomics*, Vol. 13.

[49] Supatsara Wattanakriengkrai, Patanamon Thongtanunam, Hideaki Tantithamthavorn, Chakkrit Hata, and Kenichi Matsumoto. 2020. Predicting Defective Lines Using a Model-Agnostic Technique. In *IEEE IEEE TSE*.

[50] Ian H Witten, Eibe Frank, and Mark A Hall. 2005. Practical machine learning tools and techniques. *Morgan Kaufmann* (2005), 578.

[51] Xiao Xuan, David Lo, Xin Xia, and Yuan Tian. 2015. Evaluating Defect Prediction Approaches Using a Massive Set of Metrics: An Empirical Study. In *Procs. of ACM SAC*. 1644–1647.

[52] M. Yan, X. Xia, Y. Fan, A. E. Hassan, D. Lo, and S. Li. 2020. Just-In-Time Defect Identification and Localization: A Two-Phase Framework. *IEEE TSE* (2020), 1–1.

[53] Meng Yan, Xin Xia, Yuanrui Fan, David Lo, Ahmed E. Hassan, and Xindong Zhang. 2020. *Effort-Aware Just-in-Time Defect Identification in Practice: A Case Study at Alibaba*. ACM, 1308–1319.

[54] Jingxiu Yao and Martin Shepperd. 2020. Assessing software defection prediction performance: why using the Matthews correlation coefficient matters. (2020), 120–129.

[55] Suraj Yatish, Jirayus Jiarpakdee, Patanamon Thongtanunam, and Chakkrit Tantithamthavorn. 2019. Mining software defects: should we consider affected releases?. In *Procs. of ICSE*. 654–665.

[56] Hongyu Zhang and Xiuzhen Zhang. 2007. Comments on "Data Mining Static Code Attributes to Learn Defect Predictors". *IEEE Transactions on Software Engineering* 33, 9 (2007), 635–637. https://doi.org/10.1109/TSE.2007.70706