On the Reliability of the Area Under the ROC Curve in Empirical Software Engineering

Luigi Lavazza, Sandro Morasca and Gabriele Rotoloni

June 7, 2023

On the Reliability of the Area Under the ROC Curve in Empirical Software Engineering

LUIGI LAVAZZA, SANDRO MORASCA, and GABRIELE ROTOLONI, Università degli Studi dell'Insubria, Italy

Binary classifiers are commonly used in software engineering research to estimate several software qualities, e.g., defectiveness or vulnerability. Thus, it is important to adequately evaluate how well binary classifiers perform, before they are used in practice. The Area Under the Curve (*AUC*) of Receiver Operating Characteristic curves has often been used to this end. However, *AUC* has been the target of some criticisms, so it is necessary to evaluate under what conditions and to what extent *AUC* can be a reliable performance metric.

We analyze *AUC* in relation to ϕ (also known as Matthews Correlation Coefficient), often considered a more reliable performance metric, by building the lines in the ROC space with constant value of ϕ, for several values of ϕ, and computing the corresponding values of *AUC*.

By their very definitions, *AUC* and ϕ depend on the prevalence ρ of a dataset, which is the proportion of its positive instances (e.g., the defective software modules). Hence, so does the relationship between *AUC* and ϕ. It turns out that *AUC* and ϕ are very well correlated, and therefore provide concordant indications, for balanced datasets (those with $\rho \simeq 0.5$). Instead, *AUC* tends to become quite large, and hence provide over-optimistic indications, for very imbalanced datasets (those with $\rho \simeq 0$ or $\rho \simeq 1$).

We use examples from the software engineering literature to illustrate the analytical relationship linking *AUC*, ϕ, and ρ. We show that, for some values of ρ, the evaluation of performance based exclusively on *AUC* can be deceiving. In conclusion, this paper provides some guidelines for an informed usage and interpretation of *AUC*.

CCS Concepts: • **General and reference → Empirical studies**; **Measurement**; **Estimation**; • **Software and its engineering →** *Risk management*.

Additional Key Words and Phrases: Binary classifiers, predictors, accuracy, performance metrics, Pearson ϕ, Matthews Correlation Coefficient.

1 INTRODUCTION

The usage of binary classifiers is increasingly frequent in software engineering research. For instance, binary classifiers are commonly used in research aiming to define new methods for predicting faulty modules[1], spotting code that is hard to maintain, identifying vulnerabilities, etc. In many cases, the effectiveness of such research depends on the degree to which modules are correctly classified, e.g., how well a technique separates faulty modules from not faulty ones.

Classification activities have practical consequences. For instance, failing to correctly identify a module that is actually faulty leads to classifying it as if it were not faulty and therefore not subjecting it to additional development activities to identify and remove defects from it before the module is delivered, at which time potentially serious consequences may ensue for the software development company and the customers. Conversely, failing to correctly classify a module that is not faulty as such leads to dealing with it as it it were faulty and thus having it go through unnecessary development activities, with unnecessary added costs for the software development organization.

[1]In this paper, by the term "module," we denote any piece of software (e.g., routine, method, class, package, subsystem, system).

Authors' address: Luigi Lavazza, luigi.lavazza@uninsubria.it; Sandro Morasca, sandro.morasca@uninsubria.it; Gabriele Rotoloni, rotoloni.gabriele@gmail.com, Università degli Studi dell'Insubria, Varese, Italy.

Since perfect classification is hardly ever attained, real-life classifiers will classify some modules incorrectly. The evaluation of a classifier needs to take into account both kinds of misclassifications, so overall performance metrics are needed to have a comprehensive evaluation of a classifier.

To this end, Receiver Operating Characteristic (ROC) curves are widely used; specifically, the Area Under the Curve (*AUC*) is used as an indicator of the extent to which binary classifiers make correct predictions [7, 12, 16, 21].

However, the usage of *AUC* as a performance[2] indicator for binary classifiers has been criticized (as described in Section 2.4 below). Hence, we are facing a potentially quite critical situation: if evaluations based on *AUC* are not reliable, a significant fraction of software engineering research could have been incorrectly evaluated.

In this paper, we address the problem of assessing the reliability of *AUC* by establishing quantitative relationships between *AUC* and ϕ (alias Matthews Correlation Coefficient), a performance metric that is considered more reliable and whose usage is recommended by several researchers [1, 9, 23].

The results of our analysis show that the relationship between *AUC* and ϕ depends strongly on the prevalence of the "positive" class in the set of modules, which is the proportion of those modules that belong to the "positive" class (for instance, the proportion of defective modules). Accordingly, this paper contributes to the current research practices in software engineering by investigating under what conditions *AUC* is a reliable indicator of performance and when it should be regarded as possibly misleading.

The remainder of the paper is organized as follows: Section 2 provides some background, by illustrating ROC curves, *AUC*, and reporting the main criticisms to *AUC*. Section 3 discusses the variation of *AUC* when ϕ is constant, depending on the prevalence of the testing dataset. In Section 4, the obtained results are discussed and recommendations concerning the usage of the considered performance metrics are given. Section 5 discusses the related work. Finally, Section 6 draws some conclusions and outlines future work.

2 BACKGROUND

We here concisely illustrate the basic concepts that are used throughout the paper.

2.1 The Confusion Matrix

The performance of a binary classifier on a set of n modules[3] is usually assessed based on a 2×2 matrix called "confusion matrix" (also known as "contingency table") that shows how many of those n modules are correctly and incorrectly classified. As Table 1 shows, the cells of a confusion matrix contain the numbers of modules that are: correctly estimated negative (True Negatives *TN*); incorrectly estimated negative (False Negatives *FN*); incorrectly estimated positive (False Positives *FP*); and correctly estimated positive (True Positives *TP*).

The column totals *AN* and *AP*, i.e., the number of actually negative and positive modules, respectively, depend exclusively on the dataset and not on the specific binary classifier. Instead, the row totals *EN* and *EP*, i.e., the number of estimated negative and positive modules, respectively, depend on the binary classifier as well.

An especially important characteristic of a dataset is the prevalence of the positive class, i.e., $\rho = \frac{AP}{n}$. Prevalence ρ is in the [0,1] range and is closely related to the notion of class imbalance as quantified by *IR* (Imbalance Ratio), which is

[2]Throughout this paper, we use the term "performance," borrowed from the Machine Learning field, to indicate accuracy. This choice is also motivated by the fact that "Accuracy" is the name of a specific performance metric.
[3]The concept of confusion matrix, as well as several others in the paper, is defined for sets of "instances," and not just sets of "modules." However, since our goal is to use these concepts in the software engineering domain, we use the term "module" throughout the paper.

Table 1. A confusion matrix.

	Actual Neg.	Actual Pos.	
Est. Negative	TN	FN	$EN{=}TN{+}FN$
Est. Positive	FP	TP	$EP{=}FP{+}TP$
	$AN{=}TN{+}FP$	$AP{=}FN{+}TP$	$n{=}AN{+}AP$ $={EN}{+}EP$

the ratio of the number of elements of the majority class to number of the elements of the minority class

$$IR = \max\left\{\frac{AN}{AP}, \frac{AP}{AN}\right\} = \max\left\{\frac{1-\rho}{\rho}, \frac{\rho}{1-\rho}\right\} \tag{1}$$

$IR{=}1 \Leftrightarrow \rho{=}\frac{1}{2}$ indicates that the dataset is perfectly balanced, as it includes as many positive modules as negative ones. The greater $\left|\rho-\frac{1}{2}\right|$, the more imbalanced the dataset is. In software defect prediction, there is a majority of negative modules, so, for instance, Song, Guo, and Shepperd [18] take $IR = \frac{AN}{AP} = \frac{1-\rho}{\rho}$. In this paper, we are interested in showing that prevalence ρ itself is always relevant in the evaluation of binary classifiers, even when the dataset is perfectly balanced, so we use prevalence ρ instead of imbalance throughout the paper. As a matter of fact, some shortcomings of performance metrics are more serious for balanced datasets.

2.2 Performance Metrics

Performance metrics are computed based on the cells of a confusion matrix to obtain an overall evaluation of how well a binary classifier classifies the modules in a dataset or focus on a specific aspect of performance.

Table 2 lists the performance metrics that are used in the paper. They are among the most relevant performance metrics and the most frequently used in Empirical Software Engineering [15].

Table 2. Performance Metrics

Term	Formula	Name(s)
TPR	$\frac{TP}{AP}$	True Positive Ratio, Recall, Sensitivity
FPR	$\frac{FP}{AN}$	False Positive Ratio, Fall-out
ϕ	$\frac{TP\ TN - FP\ FN}{\sqrt{EN\ EP\ AN\ AP}}$	ϕ, Matthews Correlation Coefficient [13]

TPR and FPR range between 0 and +1. For TPR, larger values indicate better performance, while for FPR is the opposite. High values of TPR basically mean that the binary classifier is able to estimate actually positive modules correctly in many cases. This is important in Empirical Software Engineering because the cost per false negative is usually quite high. Conversely, low values of FPR mean that the binary classifier can correctly estimate actually negative modules. So, there is a small proportion of actually negative modules that may undergo unnecessary additional actions because they are incorrectly believed to be positive.

ϕ is in the $[-1, 1]$ range, with $\phi = 1$ if and only if $FP{=}FN{=}0$, i.e., in the perfect classification case. $\phi = 0$ is the expected performance of the random binary classifier that estimates a module positive with any given probability p. $\phi = -1$ if and only if $TP{=}TN{=}0$, i.e., in the perfect misclassification case. In general, $\phi < 0$ means that a binary classifier appears to be better at misclassifying modules than at classifying them correctly. By simply inverting the estimations, we obtain a binary classifier that instead is better at classifying modules correctly—and better than random too. In general, when $\phi \geq 0$, higher values of binary classifier are preferable.

In essence, ϕ is an effect size measure, which quantifies how far the estimates given by a binary classifier are from being random. A commonly cited interpretation guideline proposal [2] for ϕ uses $\phi=0.1$, $\phi=0.3$, and $\phi=0.5$ respectively to denote a weak, a medium, and a large effect size. ϕ is also related to the χ^2 statistic, since $|\phi| = \sqrt{\frac{\chi^2}{n}}$. The usage of ϕ has been recommended by several researchers [1, 9, 23].

2.3 The ROC Curve

A Receiver Operating Characteristic (ROC) curve is built by taking a scoring function f that associates a value (e.g., a probability) with each module, and estimating the target class of all modules for all possible thresholds that can be set on f. Each of these thresholds on f leads to the definition of a binary classifier. For instance, with a specified threshold t, a module is estimated positive if its value of f is greater than or equal to t and negative otherwise. This defines a proper binary classifier, which will have a value of *FPR* and a value of *TPR* when applied to a dataset. The application of all possible binary classifiers obtained for all values of t to a dataset results in obtaining a set of pairs $\langle FPR, TPR \rangle$. The ROC curve is the graphical representation obtained when conjoining these pairs in order of threshold in a plane (the so-called "ROC space" [3]) with *FPR* as the x-axis and *TPR* as the y-axis, as shown in Figure 1.

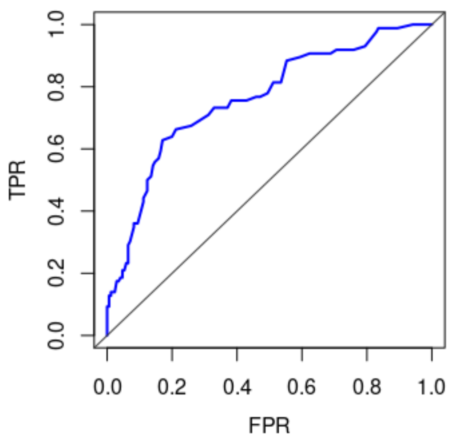

Fig. 1. An example of ROC curve.

It can be shown that a ROC curve is monotonically increasing and goes through points (0, 0) and (1, 1).

2.4 AUC

The Area Under the Curve (*AUC*) of a ROC curve is a very commonly used performance metric for assessing the discriminating ability of a *family* of binary classifiers.

The ideal classification is obtained by a classifier with *FPR*=0 and *TPR*=1, i.e., with no false positives and no false negatives. Thus, a scoring function is good if the binary classifiers that can be derived from it with all possible thresholds are represented by points close to the (0, 1) point. This happens when the ROC curve lingers near the y-axis (i.e., the

FPR=0 line) and then near the *TPR*=1 horizontal straight line. As a consequence, the value of the area under the ROC curve will be close to 1. Roughly speaking, the farther the points from the ideal classification point, the lower the performance of their associated binary classifiers, the lower the overall performance of the scoring function, and the lower the *AUC*. This is the rationale for selecting *AUC* as a measure for the overall performance of a scoring function.

Note that the minimum value of *AUC* is 0.5, in practice. When *AUC* is below 0.5, class estimations tend to be more incorrect than correct, so we can always obtain a binary classifier with *AUC* above 0.5 by taking the original binary classifier and simply inverting the estimations, in the same way as we described for values of ϕ less than zero.

Hosmer at al. [6] propose the intervals in Table 3 as interpretation guidelines for *AUC* as a measure of how well f discriminates between positives and negatives for all values of t.

Table 3. Evaluation of *AUC*

AUC range	Evaluation
$AUC = 0.5$	totally random, as good as tossing a coin
$0.5 < AUC < 0.7$	poor, not much better than a coin toss
$0.7 \leq AUC < 0.8$	acceptable
$0.8 \leq AUC < 0.9$	excellent
$0.9 \leq AUC$	outstanding

Several authors criticized AUC as too general, because it includes unrealistic decision thresholds [10, 11]. In fact, it accounts for the entire ROC curve, i.e., for all thresholds, including those very close to zero (when all modules are classified as positive) and one (when all modules are classified negative).

3 *AUC* AND ϕ

3.1 Iso-ϕ Curves

It has been shown that, in the triangle above and to the left of the diagonal in the ROC space, the iso-ϕ curves, i.e., the lines characterized by a constant value of $\phi = \overline{\phi}$, are arcs of ellipses [14]. Iso-ϕ curves are described by Formula (2) in implicit form.

$$\phi = \frac{\sqrt{\rho(1-\rho)}(TPR - FPR)}{\sqrt{(\rho TPR + (1-\rho) FPR)(\rho(1-FPR) + (1-\rho)(1-TPR))}} \tag{2}$$

Figure 2 shows the iso-ϕ curves for values of $\phi = \overline{\phi}$ multiples of 0.1, when ρ=0.5. The diagonal is the line with ϕ=0: the greater ϕ, the higher the iso-ϕ curve in the ROC space. In the extreme case in which $\phi = 1$, the corresponding iso-ϕ curve in the ROC space reduces to the single point (0, 1), the one that corresponds to perfect estimation; it can be proven that this is true regardless of the value of ρ.

Formula (2) indicates that the iso-ϕ curve is a parametric curve that depends on ρ. For instance, Figures 3 and 4 show the iso-ϕ curves when ρ is 0.1 and 0.9, respectively: these curves are quite different from those of Figure 2.

Also, Figures 3 and 4 graphically depict the case of two iso-ϕ curves with complementary values of ρ, i.e., 0.1 and 0.9. These figures show a symmetry between these two curves, which also holds in a more general case. Starting from Formula (2), it can be proven that the iso-ϕ curve with $\phi = \overline{\phi}$ when $\rho = \overline{\rho}$ is the symmetrical of the line with $\phi = \overline{\phi}$ when $\rho = 1 - \overline{\rho}$, with respect to the $y = 1 - x$ line (which is the diagonal of the ROC space passing through points (0,1) and (1,0)). This symmetry implies that two iso-ϕ curves with complementary values of ρ have the same *AUC*.

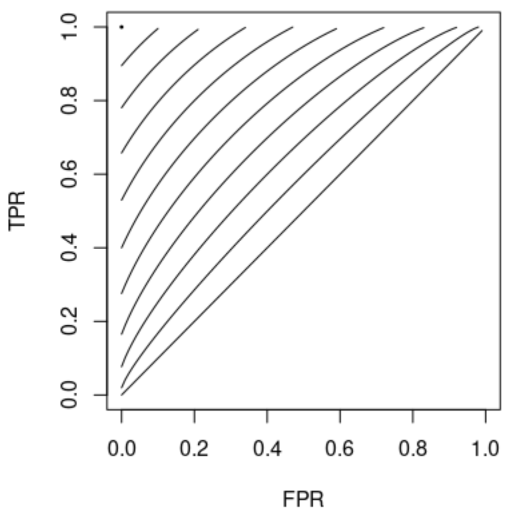

Fig. 2. Iso-ϕ curves for various values of ϕ, when ρ=0.5.

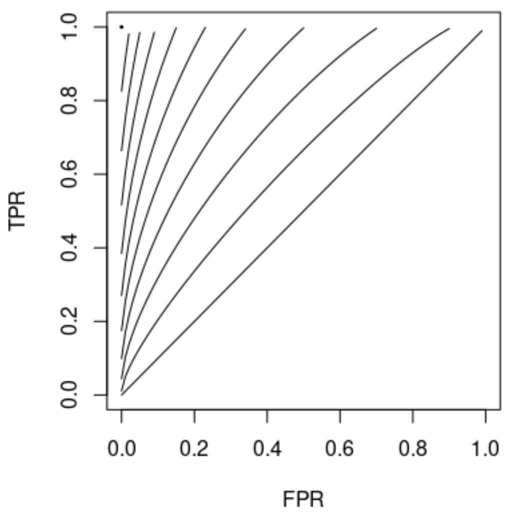

Fig. 3. Iso-ϕ curves for various values of ϕ, when ρ=0.1.

A few special cases occur depending on the values of ρ and ϕ. When $\rho = 0$ (and, therefore, when $\rho = 1$), the dataset contains modules belonging to only one of the two target classes, i.e., either only negative modules, so $\rho = 0$, or only positive modules, so $\rho = 1$. It can be shown that the arc of ellipse representing the iso-ϕ curve for any constant value of $\phi > 0$ chosen degenerates into the union of two segments: one from (0, 0) to (0, 1) and the other from (0, 1) to (1, 1). In the special sub-case in which $\rho = 0$ or $\rho = 1$ and also $\phi = 0$, the iso-ϕ curve in the ROC space is undefined.

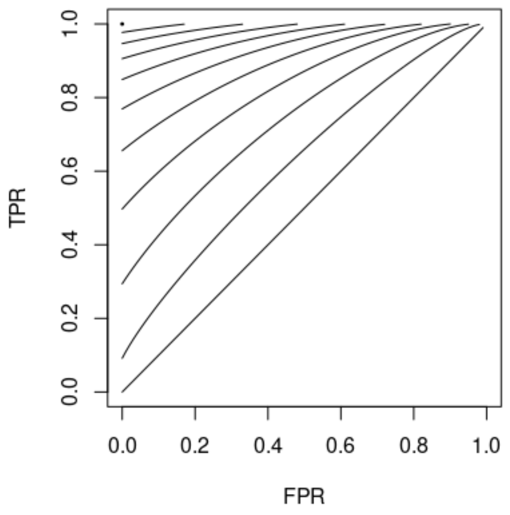

Fig. 4. Iso-ϕ curves for various values of ϕ, when ρ=0.9.

3.2 *AUC* for Iso-ϕ Curves

We computed *AUC* for iso-ϕ curves for various values of ϕ and ρ. We solved the equation in Formula (2) for *TPR* to have an explicit representation of *TPR* as a function of *FPR* for any given values of ϕ and ρ. Then, we computed *AUC* via the trapezoidal numerical integration applied to the explicit representation. Specifically, we divided the $[0, 1]$ interval of *FPR* into 1,000 equally wide intervals and computed the approximation of the integral by summing the areas of the resulting trapezoids. A finer-grain subdivision of the $[0, 1]$ interval of *FPR* may lead to results that differ from ours only starting from the fourth decimal digit.

Table 4 reports the results for a subset of values of ϕ and ρ. As for ϕ, we used all multiples of 0.1 in the $[0, 1]$ interval. As for ρ , we report *AUC* for values of ρ multiples of 0.1 in the $[0, 0.5]$ interval, because of the symmetry existing in the iso-ϕ lines for complementary values of ρ; for instance, the row of Table 4 reporting the results $\rho = 0.2$ also reports the results for $\rho = 0.8$. Since $\rho = 0$ represents a special case and all the *AUC* values are equal to 1, we also used $\rho = 0.01$ as a representative case of values very close to $\rho = 0$. On a final note, the cell corresponding to $\rho = 0$ and $\phi = 0$ is void because the corresponding iso-ϕ curve is undefined and so is the value of *AUC*.

To visually illustrate how ρ affects the relationship between *AUC* and ϕ, Figure 5 shows the dependence of *AUC* on ϕ, for all ρ values that are multiple of 0.01 in the [0.01, 0.99] range. The highest line corresponds to the extreme values of ρ, i.e., 0.01 and 0.99; the lowest line corresponds to ρ=0.5.

Table 4 and Figure 5 provide a first quite interesting insight in the relationship between *AUC* and ϕ. When ρ is not far from 0.5 (i.e., for not very unbalanced datasets), *AUC* and ϕ provide coherent indications. For instance, with ρ=0.5, when ϕ=0.3 (which indicates barely acceptable performance according to Cohen [2]), we have *AUC* just above 0.7 (the acceptability threshold according to Hosmer at al. [6]). Likewise, when ϕ=0.5 (which indicates good performance), *AUC* is between 0.8 and 0.9 (indicating good performance as well). Instead, when ρ is very low or very high, *AUC* appears much more optimistic than ϕ: when ρ=0.1, ϕ=0.2, which indicates poor performance, corresponds to *AUC* well above

Table 4. *AUC* for constant ϕ, depending on ρ.

	ϕ										
ρ	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0		1	1	1	1	1	1	1	1	1	1
0.01	0.5	0.824	0.936	0.971	0.985	0.992	0.996	0.998	0.999	1	1
0.1	0.5	0.63	0.745	0.834	0.895	0.936	0.963	0.981	0.992	0.998	1
0.2	0.5	0.598	0.692	0.776	0.845	0.899	0.939	0.967	0.986	0.997	1
0.3	0.5	0.586	0.669	0.748	0.818	0.876	0.923	0.958	0.982	0.996	1
0.4	0.5	0.58	0.659	0.735	0.804	0.865	0.915	0.953	0.98	0.995	1
0.5	0.5	0.578	0.656	0.731	0.8	0.861	0.912	0.951	0.979	0.995	1

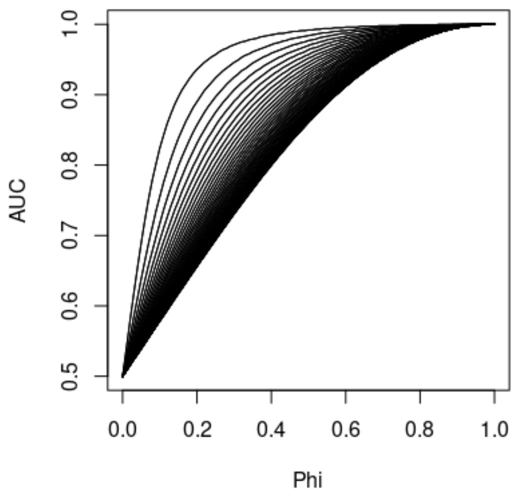

Fig. 5. *AUC* vs ϕ, for various values of ρ.

0.7, which is usually interpreted as an indication of quite good performance. The closer ρ to zero, the more divergent the indications by *AUC* and ϕ are.

In conclusion, it seems that for very unbalanced datasets, *AUC* can be misleading, showing good (or even optimum) performance when the considered classifier's performance is actually poor.

3.3 An Example

We here present an example, based on real-life datasets from the collection by Jureczko and Madeyski [8]. The example shows how *AUC*-based performance evaluations can be misleading.

Figure 6 shows the ROC curves of two defect-proneness models, which we obtained by using Binary Logistic Regression (BLR). Specifically, one model is for the `tomcat` project, and was obtained using LOC (Lines Of Code) as the independent variable. The other model was obtained for the `xalan 2.6` project using CBO (Coupling Between Objects) as the independent variable.

Both models have *AUC*=0.79. Therefore, if performance evaluation is based on *AUC* only, one should conclude that the two models have equivalent performance, and a fairly good one, according to Table 3.

However, `tomcat` and `xalan 2.6` have quite different prevalence values: ρ=0.09 for `tomcat` and ρ=0.46 for `xalan 2.6`. It can be computed that the iso-ϕ curve that has *AUC*=0.79 when ρ=0.46 is the one for which ϕ=0.38. Accordingly, we can conclude that the performance obtained for `xalan 2.6` is actually fairly good. Instead, the iso-ϕ curve that has *AUC*=0.79 when ρ=0.09 has ϕ slightly less than 0.24. Hence, the performance obtained for `tomcat` (which has ρ=0.09) cannot be regarded as good, according to ϕ.

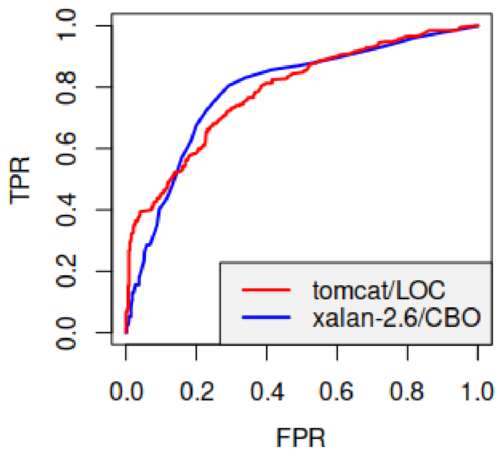

Fig. 6. ROC curves of the model based on LOC for `tomcat` (ρ=0.09) and the model based on CBO for `xalan 2.6` (ρ=0.46).

4 DISCUSSION OF RESULTS AND RECOMMENDATIONS

4.1 Practical Consequences

The example in Section 3.3 shows a performance evaluation that can lead to misleading conclusions. This can be the case in practice, as we show via the following example.

Uchigaki et al. proposed a faultiness prediction technique that they introduced as "an ensemble of simple regression models" to improve the performance of cross-project prediction [20]. They used NASA datasets [17, 19] to test the proposed technique. Each project dataset was used to build a faultiness prediction model, which was then used to estimate the faultiness of all of the other projects. The prevalence ρ of the datasets is in the [0.005, 0.323] range. Table 5, which is taken from [20], shows the results of the study, which are evaluated via *AUC*. For instance, when using project `MC2` to estimate the faultiness of project `KC1`, the value of *AUC* obtained is 0.734.

A closer look at Table 5 shows that the datasets for which the best performance is achieved (`PC5`, `PC2` and `MC1`) are also the ones having smallest values of ρ. Instead, no prediction concerning `MC2`, the project with the highest ρ, achieves *AUC*$\geq$0.7, that is, no prediction concerning `MC2` is acceptable, according to Table 3. This fact is highly suspicious: it may be the consequence of the "inflation" of *AUC* when ρ is very small, as observed in Table 4 and Figure 5.

The constant ϕ corresponding to the $\langle\rho, AUC\rangle$ combinations of Table 5 are presented in Table 6. Just a few predictions, all concerning `PC5`, achieve $\phi \geq 0.3$, which is considered the acceptability threshold.

Therefore, Table 6 with its low values of ϕ seems to confirm that the results reported in Table 5 and their interpretations according to Table 3 are overoptimistic.

Table 5. Results table from [20]

		Fit data											
		CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
	CM1 (ρ=0.095)		0.745	0.747	0.727	0.752	0.746	0.757	0.751	0.748	0.75	0.708	0.746
	JM1 (ρ=0.194)	0.655		0.645	0.661	0.658	0.654	0.665	0.658	0.648	0.649	0.664	0.643
	KC1 (ρ=0.154)	0.726	0.746		0.733	0.75	0.734	0.748	0.754	0.728	0.73	0.74	0.741
	KC3 (ρ=0.100)	0.77	0.778	0.775		0.78	0.773	0.776	0.781	0.77	0.782	0.781	0.768
	MC1 (ρ=0.015)	0.774	0.798	0.792	0.833		0.768	0.798	0.813	0.752	0.803	0.822	0.743
Test data	MC2 (ρ=0.323)	0.634	0.631	0.632	0.616	0.628		0.631	0.626	0.638	0.634	0.609	0.648
	MW1 (ρ=0.077)	0.749	0.76	0.758	0.742	0.777	0.726		0.777	0.717	0.757	0.754	0.696
	PC1 (ρ=0.072)	0.658	0.679	0.675	0.684	0.703	0.658	0.684		0.657	0.715	0.696	0.659
	PC2 (ρ=0.005)	0.845	0.856	0.854	0.837	0.859	0.847	0.851	0.853		0.856	0.848	0.849
	PC3 (ρ=0.106)	0.706	0.733	0.725	0.736	0.752	0.706	0.743	0.75	0.686		0.745	0.678
	PC4 (ρ=0.132)	0.63	0.66	0.645	0.715	0.672	0.631	0.645	0.687	0.626	0.706		0.635
	PC5 (ρ=0.033)	0.853	0.921	0.925	0.898	0.921	0.913	0.922	0.936	0.871	0.898	0.916	

Table 6. Derived values of ϕ for [20]

		Fit data											
		CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
	CM1		0.195	0.197	0.179	0.203	0.197	0.207	0.201	0.199	0.201	0.162	0.197
	JM1	0.158		0.148	0.164	0.162	0.156	0.168	0.162	0.150	0.152	0.168	0.145
	KC1	0.217	0.238		0.225	0.244	0.227	0.242	0.248	0.219	0.221	0.232	0.234
	KC3	0.225	0.234	0.230		0.236	0.229	0.230	0.236	0.225	0.238	0.236	0.223
	MC1	0.096	0.107	0.104	0.128		0.093	0.107	0.115	0.085	0.110	0.121	0.082
Test data	MC2	0.160	0.156	0.158	0.139	0.152		0.156	0.150	0.166	0.160	0.131	0.178
	MW1	0.182	0.191	0.189	0.176	0.207	0.162		0.207	0.154	0.189	0.186	0.139
	PC1	0.105	0.121	0.118	0.125	0.139	0.105	0.125		0.105	0.148	0.134	0.107
	PC2	0.079	0.084	0.083	0.076	0.086	0.080	0.082	0.083		0.084	0.081	0.081
	PC3	0.168	0.193	0.186	0.195	0.211	0.168	0.203	0.209	0.150		0.205	0.143
	PC4	0.113	0.141	0.127	0.193	0.152	0.113	0.127	0.166	0.109	0.184		0.117
	PC5	0.205	0.305	0.313	0.262	0.305	0.289	0.305	0.340	0.225	0.262	0.293	

Uchigaki et al. [20] concluded that the technique they proposed 1) achieves better results than conventional Multivariate Logistic Regression models, and 2) never gets worse than random performance, while conventional techniques do. Our analysis does not disprove these conclusions; in fact, it can be used to confirm the conclusions by Uchigaki et al. However, the performance of the proposed technique, although better than conventional techniques', appears rather low. In cases like this, evaluating predictive models by means of additional performance metrics could be useful, to appreciate the real value of the results.

4.2 Recommendations

The observations in the previous sections suggest that, when evaluating binary classifiers, considering *AUC* alone may provide an incomplete picture, especially if there is a noticeable difference between the numbers of positives and negatives in the test dataset. However, this does not mean that *AUC* must be considered completely unreliable and misleading. In fact, there is a good concordance between *AUC* and ϕ in balanced test datasets, as apparent in Table 4 and Figure 5. Instead, *AUC* and ϕ disagree when ρ is very small or very large, and ϕ is small: this fact is also apparent in Table 4 and Figure 5.

Therefore, the results suggest that *AUC* can be used and should give reliable results when evaluating a binary classifier if ρ is not far from 0.5. As ρ gets closer to either 0 or 1, the computation of *AUC* should be accompanied by the computation of ϕ in order to understand whether the *AUC* is reliable or not.

4.3 Limitations

The relationship between *AUC* and ϕ described in Section 3 holds for ROC curves having constant ϕ. This gives us an idea of how ρ affects *AUC*, as shown also by the examples in Sections 3.3 and 4.1.

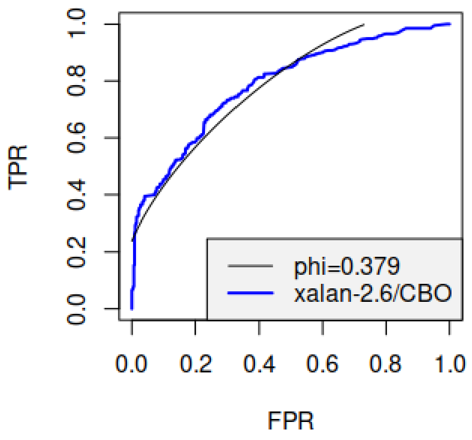

Fig. 7. ROC curve of the model based on CBO for `xalan 2.6` and ROC curve having ϕ=0.379.

However, we must consider that in general ROC curves are made of points having different values of ϕ. For instance, Figure 7 shows the ROC curve of the model based on CBO for project `xalan 2.6` and the iso-ϕ curve with the same *AUC*. Although the two lines are close to each other, it is easy to see that the binary classifiers corresponding to some points of the ROC curve (namely those corresponding to the points above the thin black curve) have $\phi > 0.379$, while the binary classifiers corresponding to other points (those below the black thin curve) have $\phi < 0.379$.

Studying the relationship between *AUC* and ϕ in the general case requires an empirical study that is among our future objectives.

5 RELATED WORK

Several scientific papers have investigated the properties of performance metrics used for binary classifiers in software engineering applications. The ones that are most related to our study involve the comparison of performance metrics. For instance, [1, 9, 22, 24] compare ϕ and the F-score, which is the harmonic mean of Precision and Recall. Our study carries out a similar research, involving *AUC* and ϕ.

It is interesting to note that some results of the mentioned research highlight characteristics of performance metrics, or, more precisely, of relationships among performance metrics, that are similar to those reported in this paper, but of opposite sign. For instance, Lavazza and Morasca showed that the F-measure and ϕ provide coherent and well correlated results when prevalence ρ is very small, while F-measure and ϕ can provide quite discordant indications when ρ is not very small (including when ρ=0.5) [9]. Hence, the results described in this paper, along with those described in previous work by other researchers, suggest that performance metrics should be chosen carefully, considering the prevalence of the dataset to which the classifier to be evaluated was applied.

AUC has been the subject of criticism outside the software engineering arena. For instance, Hand [5] shows that *AUC* is a weighted minimum misclassification cost function. However, the weights (i.e., the unit costs per misclassification) used are variable, and essentially depend on the binary classifier used. This makes *AUC* an ill-defined performance metric when it comes to interpreting it as being related to some kind of cost. As a result, Hand suggests the use of other performance metrics, with a more direct correspondence with the objectives of the users of binary classifiers (software development organizations and software engineering researchers, in our case).

An empirical comparison between *AUC* and ϕ was carried out by Halimu et al. [4]. They used 54 unbalanced datasets and 23 balanced datasets to train binary classifiers built with four different algorithms and validated them using cross-validation. The aim of the research was to investigate which of the two performance metrics *AUC* and ϕ is better, by using as indicators the degree of consistency of ranking between the two performance metrics and the degree of discrimination that they provide: in these respects, Halimu et al. found that *AUC* is better than ϕ.

There are a few important differences between the evaluation carried out by Halimu et al. and ours. They perform an empirical evaluation, while our evaluation is based on analytical grounds, and descends directly from the definitions of the considered performance metrics. In addition, Halimu et al. address only the behavior of ϕ and *AUC* in comparing two (or more) given models: they do not provide any conclusion concerning the reliability of ϕ or *AUC*; i.e., they do not guarantee that if any of the two metrics states that a model's performance is good (respectively, not good) then the model has actually good (respectively, not good) accuracy. Instead, we provide also an evaluation of how reliable ϕ and *AUC* are in assessing the performance of a single given model. Finally, we can note that our study confirms, on theoretically solid ground, that ϕ and *AUC* provide concordant indications when comparing the performances of two given models applied to the same dataset: this property is guaranteed by the fact that all ϕ vs. *AUC* functions for a given ρ are monotonic, as shown in Figure 5.

6 CONCLUSIONS

In this paper, we have investigated under what conditions and to what extent *AUC*, which is one of the leading performance metrics used in software engineering to assess binary classifiers [15], can be considered reliable. We therefore compared it to the results that can be obtained by using ϕ, another performance metric that has been shown to be reliable under many circumstances and recommended by several researchers [1, 9, 23].

Our analysis studies the impact of prevalence on the relationship between *AUC* and ϕ. This relationship is quite strong for balanced datasets; however, when a dataset is very unbalanced, *AUC* has a tendency to provide overoptimistic evaluations.

As a result, we also provide some guidelines that can be useful when using *AUC* to assess classifiers. Also, they can be used to evaluate the reliability of results that are published in the scientific literature.

Concerning future work, there are two activities that we intend to carry out to extend the work presented here. The first one is intended to overcome the limitations described in Section 4.3: we will study the relationship between *AUC* and ϕ in the general case, via suitable empirical studies. The second one is motivated by the critical aspects of *AUC* that have been highlighted by several authors [10, 11]. Specifically, it has been observed that *AUC* includes unrealistic decision thresholds, as it accounts for the entire ROC curve, i.e., for all thresholds, including those very close to zero (when all modules are classified as positive) and one (when all modules are classified negative). To overcome this limitation, Morasca and Lavazza proposed to limit the computation of the area under the curve to the "Region of Interest" where the points of the ROC curve represent classifiers having acceptable performance [14]. For instance, a possible way to define the Region of Interest is by considering only the region where both *TPR* and *FPR* are better than the values achieved (on average) via random classification. The work reported here, as well as the extension to the general case, can be repeated for indicators based on the area under the curve, but limited to the Region of Interest.

We also plan to extend the work to study the relationship between *AUC* and misclassification cost, i.e., the cost due to false positives and false negatives.

ACKNOWLEDGMENTS

This work has been supported by the "Fondo di Ricerca d'Ateneo" funded by the Università degli Studi dell'Insubria.

REFERENCES

[1] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21, 1 (2020), 1–13.

[2] Jacob Cohen. 1988. *Statistical power analysis for the behavioral sciences Lawrence Earlbaum Associates.* Routledge, New York, NY, USA. 20–26 pages.

[3] Tom Fawcett. 2006. An Introduction to ROC Analysis. *Pattern Recogn. Lett.* 27, 8 (June 2006), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

[4] Chongomweru Halimu, Asem Kasem, and S. H. Shah Newaz. 2019. Empirical Comparison of Area under ROC curve (AUC) and Mathew Correlation Coefficient (MCC) for Evaluating Machine Learning Algorithms on Imbalanced Datasets for Binary Classification. In *Proceedings of the 3rd International Conference on Machine Learning and Soft Computing, ICMLSC 2019, Da Lat, Vietnam, January 25-28, 2019.* ACM, 1–6. https://doi.org/10.1145/3310986.3311023

[5] David J. Hand. 2009. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning* 77, 1 (2009), 103–123. https://doi.org/10.1007/s10994-009-5119-5

[6] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression.* John Wiley & Sons.

[7] Seyedrebvar Hosseini, Burak Turhan, and Dimuthu Gunarathna. 2017. A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering* 45, 2 (2017), 111–147.

[8] Marian Jureczko and Lech Madeyski. 2010. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering.* 1–10.

[9] Luigi Lavazza and Sandro Morasca. 2022. Comparing ϕ and the F-measure as performance metrics for software-related classifications. *Empir. Softw. Eng.* 27, 7 (2022), 185. https://doi.org/10.1007/s10664-022-10199-2

[10] Jorge M Lobo, Alberto Jiménez-Valverde, and Raimundo Real. 2008. AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography* 17, 2 (2008), 145–151.

[11] Susan Mallett, Steve Halligan, Matthew Thompson, Gary S Collins, and Douglas G Altman. 2012. Interpreting diagnostic accuracy studies for patient care. *Bmj* 345 (2012).

[12] Faseeha Matloob, Taher M Ghazal, Nasser Taleb, Shabib Aftab, Munir Ahmad, Muhammad Adnan Khan, Sagheer Abbas, and Tariq Rahim Soomro. 2021. Software defect prediction using ensemble learning: A systematic literature review. *IEEE Access* (2021).

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

[14] Sandro Morasca and Luigi Lavazza. 2020. On the assessment of software defect prediction models via ROC curves. *Empir. Softw. Eng.* 25, 5 (2020), 3977–4019. https://doi.org/10.1007/s10664-020-09861-4

[15] Rebecca Moussa and Federica Sarro. 2022. On the Use of Evaluation Measures for Defect Prediction Studies. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA).* Association for Computing Machinery (ACM).

[16] Jalaj Pachouly, Swati Ahirrao, Ketan Kotecha, Ganeshsree Selvachandran, and Ajith Abraham. 2022. A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. *Engineering Applications of Artificial Intelligence* 111 (2022), 104773.

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

[18] Qinbao Song, Yuchen Guo, and Martin Shepperd. 2018. A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering* 45, 12 (2018), 1253–1269.

[19] Chakkrit Tantithamthavorn. 2016. NASA Defect Dataset.

[20] Satoshi Uchigaki, Shinji Uchida, Koji Toda, and Akito Monden. 2012. An ensemble approach of simple regression models to cross-project fault prediction. In *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE, 476–481.

[21] Cody Watson, Nathan Cooper, David Nader Palacio, Kevin Moran, and Denys Poshyvanyk. 2022. A Systematic Literature Review on the Use of Deep Learning in Software Engineering Research. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–58.

[22] Jingxiu Yao and Martin Shepperd. 2020. Assessing software defection prediction performance: Why using the Matthews correlation coefficient matters. *Proceedings of the Evaluation and Assessment in Software Engineering* (2020), 120–129.

[23] Jingxiu Yao and Martin J. Shepperd. 2021. The impact of using biased performance metrics on software defect prediction research. *Inf. Softw. Technol.* 139 (2021), 106664. https://doi.org/10.1016/j.infsof.2021.106664

[24] Qiuming Zhu. 2020. On the performance of Matthews correlation coefficient (MCC) for imbalanced dataset. *Pattern Recognition Letters* 136 (2020), 71–80.