Comparing the performance of fault prediction models which report multiple performance measures: recomputing the confusion matrix

David Bowes
Science and Technology Research Institute
University of Hertfordshire
College Lane
Hatfield, AL10 9AB
United Kingdom
d.h.bowes@herts.ac.uk

Tracy Hall
Department of Information Systems and Computing
Brunel University
Uxbridge
Middlesex, UB8 3PH
United Kingdom
tracy.hall@brunel.ac.uk

David Gray
Science and Technology Research Institute
University of Hertfordshire
College Lane
Hatfield, AL10 9AB
United Kingdom
d.gray@herts.ac.uk

ABSTRACT

There are many hundreds of fault prediction models published in the literature. The predictive performance of these models is often reported using a variety of different measures. Most performance measures are not directly comparable. This lack of comparability means that it is often difficult to evaluate the performance of one model against another. Our aim is to present an approach that allows other researchers and practitioners to transform many performance measures back into a confusion matrix. Once performance is expressed in a confusion matrix alternative preferred performance measures can then be derived. Our approach has enabled us to compare the performance of 601 models published in 42 studies. We demonstrate the application of our approach on several case studies, and discuss the advantages and implications of doing this.

1. INTRODUCTION

Imagine the following simplified scenario:

You are a practitioner thinking about starting to use fault prediction models. You hope that such models will help you to identify the most fault prone parts of your system. You then plan to target your test effort on those parts of the system. You think that doing this may reduce the faults delivered to your users and reduce the cost of your system. You are not an expert in fault prediction models yourself, but you have seen many such models published in the literature. You identify several published models that have been developed in a similar software development context to your own. You decide to evaluate the performance of these models with a view to trying out the top three models in your project. However when you look at the model performance figures they are reported using a variety of different performance measures. Several studies report Precision[1] and Recall. Some report Error Rate. Some report pd and pf. Others report Popt. A few report Area Under the Curve of the Receiver Operator Curve. One provides a confusion matrix. It is beyond your expertise to identify a comparative point of reference amongst these different measures. You struggle to understand how the overall performance of a model compares to the others. And so you decide that fault prediction models are too complicated to use and abandon the idea.

This type of scenario may partially explain why the uptake of fault prediction models is low in industry. This low uptake is important as finding and fixing faults in code costs the software industry many millions of dollars every year. Predicting effectively where faults are in code occupies many researchers and practitioners. Our previous work [14] showed that 208 software fault prediction studies were published between January 2000 and December 2010. These 208 studies contained many hundreds of fault prediction models.

Despite this significant research effort it remains difficult or inconvenient to compare the performance of these models. The difficulty in comparing predictive performance means that identifying which fault prediction models perform best in a given context is complex. This complexity in comparing the performance of models is not only a likely barrier to practitioners using fault prediction models, but also makes it difficult for researchers to meta-analyse fault prediction studies [8]. This lack of opportunity to meta-analyse limits the ability of the fault prediction community to mature, as we are not building an evidence base that is as useful as it should be.

One of the difficulties when comparing the performance of fault prediction models stems from the many performance measurement schemes devised, used and reported by studies. Many of the schemes used by studies highlight different aspects of predictive performance. For example, Menzies et al. [25] use pd and pf to highlight standard predictive performance, while Mende and Koschke [23] use Popt to assess effort-awareness. The different performance measurement schemes used mean that directly comparing the

[1]Definitions of particular measures are given in Section Two.

performance reported by individual studies is difficult and potentially misleading. Such comparisons cannot compare like with like as there is no adequate point of comparison.

It is perfectly legitimate for studies to report different performance measures. Studies may be interested in reporting prediction models with particular qualities. Some studies may be interested in reporting models which reduce the amount of effort wasted on code predicted as faulty which turns out not to be faulty. In these cases, measures based on the number of false positives will be of most interest. Other studies may be developing models focused on identifying the maximum number of faults in the system. In which case measures related to the number of true positives are likely to be the performance focus. The qualities needed in a fault prediction model depend on, for example, application domain. Models used in the safety critical domain are likely to need different predictive qualities to those in other domains. However developers and potential users of models may want to compare performance in terms of a particular predictive quality. This requires a conversion of performance figures from those reported to those reflecting the predictive quality of interest. The ability to convert predictive measures in this way allows the predictive performance of a wide range of models to be compared.

We previously found [14] that Precision and Recall were the most commonly reported predictive performance measures used with binary[2] fault prediction models (e.g. [1, 6, 11, 20]). However, many studies provide only limited predictive performance data, often only reporting performance using their preferred performance measures. This preferred data often represents the performance of specific models in the most positive light. An issue also highlighted by Zeller et al. [34]. This preferred measurement data may be unusual and rarely reported in other studies. For example, only a few studies report the use of Error Rate [19, 29, 33] or Popt [23]. Without additional performance data that is more commonly reported by studies, it is difficult to satisfactorily compare the predictive performance of such models. A common point of comparison is needed.

The confusion matrix is usually at the centre of measuring the predictive performance of models (the confusion matrix is discussed in detail in Section Two). Most other predictive performance measures are calculated from the confusion matrix. The confusion matrix is a powerful point of comparative reference. All models reporting binary results can have their predictive performance expressed via a confusion matrix [26]. This means that it is a relatively universal comparative basis. It is also a simple and understandable way to show predictive performance. More sophisticated measures of predictive performance can be calculated from a confusion matrix. The confusion matrix provides measurement flexibility as specific measures may be derived from the confusion matrix which evaluate particular model qualities. The importance of the confusion matrix is discussed in detail by Pizzi et al. [28].

In this paper we present a process by which we transform a variety of reported predictive performance measures back to a confusion matrix [5]. These measures cover most of those reported by the 208 fault prediction studies we previously reviewed [14]. We illustrate this process by constructing the confusion matrix for a number of published models. From these confusion matrices we compute a range of alternative performance measures. We finally evaluate the use of our transformation process.

In Section Two we describe the measurement of predictive performance by discussing in detail the basis of the confusion matrix and related compound measures of performance. In Section Three we present our method of transforming a variety of performance measures to the confusion matrix and explain how alternative measures can then be derived from this matrix. Section Four provides the results of worked examples from the literature in which we transform the reported performance measures back to the confusion matrix. Section Five identifies the threats to the validity of the study. Section Six discusses the implications of transforming performance measures. We conclude and summarise in Section Seven.

2. MEASURING PREDICTIVE PERFORMANCE

This section is based on several previous studies which provide an excellent overview of measuring the predictive performance of fault models (e.g. [26], [16] and [22]).

2.1 The Confusion Matrix

The measurement of predictive performance is often based on the analysis of data in a confusion matrix (see [26]). Pizzi et al. [28] discuss the confusion matrix in more detail. This matrix reports how the model classified the different fault categories compared to their actual classification (i.e. predicted versus observed). This is represented by four pieces of data:

- True Positive (TP): An item is predicted as faulty and it is faulty
- False Positive (FP): An item is predicted as faulty and it is not faulty
- True Negative (TN): An item is predicted as not faulty and it is not faulty
- False Negative (FN): An item is predicted as not faulty and it is faulty

Table 1 shows the structure of a confusion matrix.

Table 1: Confusion matrix

	observed true	observed false
predicted true	TP	FP
predicted false	FN	TN

[2]Binary models are those predicting that code units (e.g. modules or classes) are either fault prone (fp) or not fault prone (nfp). Binary models do not predict the number of faults in code units. In this paper we restrict ourselves to considering only binary models that are based on machine learning techniques.

Table 2: Confusion matrix with example summed instances

	observed true	observed false
predicted true	33	2
predicted false	17	98

In a confusion matrix, it is normal for the sum of the instances of each possibility to be reported, see Table 2.

Few studies report complete confusion matrices for their experiments. Studies that do include [27], [36] and [18]. Most studies prefer to report a sub-set of the compound performance measures shown in Table 3.

2.2 Compound Measures

Many performance measures are related to components of the confusion matrix. Table 3 shows how some commonly used performance measures are calculated relative to the confusion matrix.

Table 3 shows that Accuracy is the proportion of units correctly classified. Table 3 also shows that Recall (otherwise known as the true positive rate, probability of detection (pd) or Sensitivity) describes the proportion of faulty code units (usually files, modules or packages) correctly predicted as such. Precision describes how reliable a prediction is in terms of what proportion of code predicted as faulty actually was faulty. Both Recall and Precision are important when test sets are imbalanced (see the following sub-section), but there is a trade-off between these two measures (see [16] for a more detailed analysis of this trade-off). An additional composite measure is the false positive rate (pf) which describes the proportion of erroneously predicted faulty units. The optimal classifier would achieve a pd of 1, Precision of 1, a pf of 0 and an f-measure of 1. The performance measure balance combines pd and pf. A high Balance value (near 1) is achieved with a high pd and low pf. Balance can also be adjusted to factor in the cost of false alarms which typically do not result in fault fixes. Matthews Correlation Coefficient (MCC) is a measure rarely used in software fault prediction [3]. MCC is more commonly used in medical research and bioinformatics e.g. [3, 30]. It is a Chi Square based performance measure on which all four quadrants of the confusion matrix are included in the calculation. MCC results are the equivalent of reporting R^2 in regression modelling and results range from -1 to 1 (with 0 indicating random results). Popt defined by Mende and Koschke [23] is a an effort aware performance measure which ranges between 0 and 1 with 1 being desirable.

The Receiver Operator Curve (ROC) is an important measure of predictive performance. When the combinations of Recall and pf for a series of experiments are plotted they produce a ROC. It is usual to report the area under the curve (AUC) as varying between 0 and 1, with 1 being the ideal value. Because the AUC is a result of a series of experiments where the meta-parameters are varied, it is not possible to compute the confusion matrix from AUC and visa versa[3].

[3]Although AUC is a valuable measure of performance, it is beyond the scope of our work as it is not possible to construct a confusion matrix from AUC data.

Previous studies have critiqued the use of these various measures of performance. For example, Zhang and Zhang [35], Menzies et al. [24] and Gray et al. [13] discuss the use of precision. However such a critique is beyond the scope of the work reported here.

2.3 Imbalanced Data

Substantially imbalanced data sets are commonly used in fault prediction studies (i.e. there are usually many more non-faulty units than faulty units in the data sets used in fault prediction) [7], [35]. An extreme example of this is seen in the NASA data set PC2, which has only 0.4% of data points belonging to the faulty class (23 out of 5589 data points). This distribution of faulty and non-faulty units has important implications in fault prediction. Imbalanced data can strongly influence the suitability of predictive performance measures. Measures which favour the majority class (such as Accuracy and Error Rate) are not sufficient by themselves [15]. More appropriate measures for imbalanced data sets include: Precision, Recall, f-measure, MCC and G-mean [15]. Consequently data imbalance is an important consideration in our method of recomputing the confusion matrix[4].

3. OUR METHOD OF RECOMPUTING THE CONFUSION MATRIX

To compare the results of one study with the results of another we recompute the confusion matrix for each study and then calculate the preferred compound measures from this. Zhang and Zhang [35] did something similar to this by recomputing Precision for Menzies et al.'s [25] study which originally reported pd and pf. Our approach is motivated by Zhang and Zhang's [35] work. We now describe the process by which transformation from a variety of compound measures to the confusion matrix can be achieved.

3.1 Creating a Frequency-Based Confusion Matrix

The precise method needed to recompute the confusion matrix varies slightly depending upon the original measures reported. In most cases the first thing that needs to be done is that we produce a frequency-based confusion matrix. These confusion matrices are different from instance based confusion matrices (an example of which was shown in Table 2). Table 4 shows the frequencies (or proportions for each confusion matrix quadrant) based on the instances in Table 2. These frequencies are derived by dividing the instances in each quadrant by the total number of instances in the matrix. This shows the relative proportion each quadrant represents of the whole confusion matrix. From now on we will append $_f$ to TP, TN, FP and FN to distinguish frequency values from instance based values.

3.2 Calculating Faulty and Non-Faulty Data Distributions

[4]Data imbalance also has serious implications for the training of prediction models. Discussion of this is beyond the scope of this work (instead see [13], [35], [31], [4] and [17]).

Table 3: Compound Performance Measures

Measures	Defined As
Accuracy / Correct Classification Rate (CCR)	$\frac{TP+TN}{TP+TN+FP+FN}$
Error Rate	$\frac{FP+FN}{TP+TN+FP+FN}$
Recall / True Positive Rate / Sensitivity / Probability of Detection (pd)	$\frac{TP}{TP+FN}$
True Negative Rate / Specificity	$\frac{TN}{TN+FP}$
False Positive Rate / Type I Error Rate / Probability of False Alarm (pf)	$\frac{FP}{TN+FP}$
False Negative Rate / Type II Error Rate	$\frac{FN}{FN+TP}$
Precision	$\frac{TP}{TP+FP}$
F-Measure / F-Score	$\frac{2 \times Recall \times Precision}{Recall + Precision}$
Balance	$1 - \frac{\sqrt{(0-pf)^2 + (1-pd)^2}}{\sqrt{2}}$
G-mean	$\sqrt{Recall \times Precision}$
Matthews Correlation Coefficient (MCC)	$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

Constructing a frequency based confusion matrix is possible when the class distribution (i.e. the proportion of faulty versus non-faulty units) is reported[5]. To do this we use d as the frequency of the faulty units, where:

$$d = \frac{TN + FN}{TN + TP + FP + FN} \quad (1)$$

Applying (1) to the example instances reported in Table 2 would result in:

$$n = 33 + 17 + 2 + 98, d = \frac{33 + 17}{n} = 0.3333$$

This shows that given the confusion matrix shown in Table 2, 33% of the units in the data set on which the model was applied, were faulty.

Table 4: Frequency Confusion Matrix

	observed true	observed false
predicted true	0.2200	0.0133
predicted false	0.1133	0.6533

$$TN_f + TP_f + FP_f + FN_f = 1$$
$$d = 0.2200 + 0.1133 = 0.3333$$

3.3 Transforming Specific Compound Measures

A wide variety of compound measures are reported by studies. Our approach is successful when a particular sub-set of these measures is reported by studies. Table 5 shows the pre-requisite combinations of performance measures that must be available.

Each of these combinations of measures requires a specific method by which to recompute the confusion matrix. Formulae for the most common measures reported are now described.

1. Transforming $Precision$**,** $Recall$ **and** pf

We first need to know the frequency of the true class d.

$$1 = TP_f + TN_f + FP_f + FN_f \quad (2)$$

$$d = TP_f + FN_f \quad (3)$$

It then becomes possible to calculate TP_f, FP_f, TN_f and FN_f as follows:
Given pf and d

$$TN_f = (1 - d)(1 - pf) \quad (4)$$

$$FP_f = (1 - d)pf \quad (5)$$

Given $Recall(r)$ and d

$$TP_f = d \times r \quad (6)$$

$$FN_f = d(1 - r) \quad (7)$$

[5]When this class distribution is not provided it is often possible to calculate the proportion of faulty units in a data set.

Given $FNR(TypeII(t2))$, pf and d we already have (2), (4) and (5)

$$FN_f = t2 \times d \quad (8)$$

$$TP_f = 1 - FN_f - TN_f - FP_f \quad (9)$$

Given $Precision(p)$, $Recall(r)$ and d we already have (2), (6) and (7)

$$FP_f = \frac{TP_f(1 - p)}{p} = \frac{d(1 - p)r}{p} \quad (10)$$

$$TN_f = 1 - FP_f - FN_f - TP_f \quad (11)$$

2. Transforming $ErrorRate(er)$**,** $TypeII(t2)$ **and** pf

$$d = \frac{er - pf + pf \times er}{t2} \quad (12)$$

which can then be used with (4),(5),(8) and (9)

3. Transforming $Precision(p)$**,** $Recall(r)$ **and** $Accuracy(a)$

$$d = \frac{p(1 - a)}{p - 2pr + r} \quad (13)$$

which can then be used with (6),(7),(10) and (11)

4. Transforming $Accuracy(a)$**,** pf **and** $FNR(TypeII(t2))$

$$er = 1 - a \quad (14)$$

$$d = \frac{er - pf}{t2 - pf} \quad (15)$$

which can be used with (8) to give FN_f and (5) to give FP_f.

$$TP_f = d(1 - t2) \quad (16)$$

which can be used with (11) to give TN_f.

We have automated these conversations by developing a tool[6]. This tool allows individual performance measurement data to be input and will automatically recompute the confusion matrix by iterating over the the equations until no extra performance measures can be derived.

4. CONSTRUCTING THE CONFUSION MATRIX FOR SOME EXAMPLE STUDIES

We have transformed the predictive performance data produced by 601 models reported in 42 published studies. A list of these studies is provided in the Appendix. Space restrictions make it is impossible to report the detail for all these transformations. Consequently in this section we present transformations for four examples. We chose these four examples to illustrate recomputing the confusion matrix from a range of different original measures.

4.1 Case Studies

Table 6 illustrates the original performance measurement data reported by our four case study papers. Table 6 shows that a wide range of different measurement data is reported

[6]This tool is available at: https://bugcatcher.stca.herts.ac.uk/JConfusion/

Table 5: Pre-Requisite Combinations of Performance Measures for Re-Computing the Confusion Matrix.

Fault Frequency	Type I	Type II	Precision	Recall	Accuracy	pf	Error Rate	Specificity
			✔	✔		✔		
				✔	✔			✔
			✔	✔	✔			
✔			✔	✔				
✔				✔				✔
✔				✔	✔			
✔				✔		✔		
		✔			✔	✔		
		✔				✔	✔	
	✔	✔					✔	

NB this is not an exhaustive list. For example, it is possible to calculate d by dividing the number of defective instances by the total number of instances.

Table 6: Reported Performance Measurement Data

Study	pd	pf	Error Rate	Type I	Type II	Precision	Recall	Accuracy	Total Instances	Faulty Instances
[19]		0.3134	0.3127		0.2826					
[6]						0.682	0.621	0.641		
[21]	0.471	0.0834						0.8515		
[29]			0.1615	0.1304	0.2830				520	106

by these four papers. Given this range it is difficult to evaluate how the performance of these models compares against each other.

We have recomputed the confusion matrix for these four case studies (shown in Table 7). Based on this confusion matrix data, we have computed the f-measure and MCC data for each case study (also shown in Table 7). It is now possible to comparatively evaluate the predictive performance of these case studies using this common set of data[7].

5. THREATS TO VALIDITY

There are internal and external validity issues that need to be considered when using our approach to recomputing the confusion matrix.

5.1 Internal Validity

The impact of cross-validation. Performance data reported are usually based on some form of cross-validation. This means that the numbers reported are usually average figures across a specific number of folds and / or experiments. This averaging process may introduce some minor inaccuracies into our calculations[8]. The study by Elish and Elish [12] provide a clearer understanding of the variation in performance values across a series of experiments. The results of rounding errors and variations in performance values helps to explain the negative TN_f value computed in Table 11.

Divide by zero problems. Several of our formulas are based on divisions. Where some figures are very similar we encounter divide by zero problems. This division problem is exacerbated by rounding of very small numbers. These small numbers may be very different, but when rounded become the same number. Such numbers suffer from divide by zero issues.

Data uncertainty. Identifying the balance of faulty and non-faulty units is an important part of our recomputing method. However in a few studies there is inconsistency in the class distribution figures. For example, although an author may have cited a particular class distribution, when we calculate the distribution figure inherent within the results reported (i.e. via the calculation of d), the distribution is different to that stated by the authors. Similarly in some papers where the same data set has been used the distribution varies between experiments. This inconsistency casts some uncertainty over the results in such cases. We suspect that this distribution inconsistency is partly the result of a particular machine learner dealing with the data that it is processing differently to other learners, and partly the result of studies not reporting the data pre-processing that they have applied.

5.2 External Validity

Model tuning. Some models may have been developed to maximise a particular quality (e.g. to reduce false positives). Such models are likely to perform best when their performance is expressed using measures that are sympa-

[7]The aim of this paper is to provide an approach by which others may perform comparative analysis of fault prediction models. A comparative analysis is complex and requires many factors to be taken into account, e.g. the aims of the predictive model. It is beyond the scope of this paper to provide a full comparative analysis of studies against each other.

[8]An examination of the code for LibSVM shows that the performance measure is an average value of the performance measure for each fold.

Table 7: Computed Performance Measurement Data

Study	TP_f	TN_f	FP_f	FN_f	f-measure	MCC
[19]	0.0163	0.6710	0.3063	0.0064	0.0944	0.1288
[6]	0.3335	0.3075	0.1555	0.2035	0.6501	0.2845
[21]	0.0575	0.7940	0.0732	0.0646	0.4549	0.3755
[29]	0.1422	0.6963	0.1000	0.0615	0.6377	0.5381

thetic to the qualities for which the model has been built. Interpreting the performance of such models via alternative performance measures should be treated with caution.

6. DISCUSSION

The process of translating the performance measures reported by studies to the confusion matrix reveals a variety of performance issues with studies that we now discuss.

6.1 Erroneous Results

In some cases our translation to the confusion matrix demonstrated that the original results reported by some studies could not have been possible. For example we found an error in [32]. This error was revealed as our transformations would not work correctly. As a result of this we emailed the authors to clarify the problem. The authors confirmed that a typographical error had crept into their final draft. False Alarms were reported instead of False Positives. It is easy for such errors to creep into published work, especially in an area as complex as fault prediction. Without very careful interpretation such errors can easily be missed and be misleading.

6.2 Definitions of Measures

While performing our transformations we have had difficulty in making sense of the figures reported in some studies. The reason for this was that a number of studies have used non-standard definitions for some well-known performance measures (e.g. [36] does not use a standard definition of Precision and [27] does not use the standard definitions of Sensitivity and Specificity (in both cases, the issues were confirmed by emailing the authors)). Although the definitions used were given in the paper, it is difficult for a reader to pick-up on the nuances of measurement definitions (usually provided via formulae). Consequent mis-understanding could have serious implications for subsequent model users.

6.3 Reporting Performance Based on Predicting Non-Faulty Units

Some papers have reported performance measures based on predicting the majority (non-faulty class) rather than the minority (faulty) class. In some of these cases it is also not made clear that the predictive performance is on the majority class. These issues can be very misleading when trying to evaluate predictive performance. For example, Elish and Elish [12] report a very influential fault prediction study using Support Vector Machines (SVM). Their study has been cited more than 60 times and is considered a pivotal paper in the use of SVMs. Table 8 shows the very good Accuracy, Precision and Recall performances reported by Elish and Elish for SVM using datasets cm1, pc1, kc1 and kc3 (taken from [12]).

Our process to recompute the confusion matrix would not work on these figures when we assumed that the values for Precision and Recall were based on the non-faulty class. Tables 9 and 10 show our workings for this recomputation. Our workings suggest that Elish and Elish have reported the performance of their SVM models based on predicting non-faulty units rather than faulty units. Since the vast majority of units in data sets are non-faulty (ranging between 84.6% and 93.7% in their case), predicting the majority class is very easy and so high performance figures are to be expected. Such models are not useful. Our findings are complementary to those of several other authors who report problems reproducing the high predictive performances reported by Elish and Elish when using their SVM settings. For example [2] reports that most papers report a far lower Recall value. [9] and [10] used the same SVM settings. [9] reported Specificity and Sensitivity values and [10] reported Precision and Recall for both the faulty and non-faulty classes which are similar to our recomputed values. Despite emailing Elish and Elish several times we were unable to get a reply confirming the basis of their performances.

Table 10: SVM Confusion Matrix of the Majority Class

Dataset	TP_f	FN_f	FP_f	TN_f
cm1	0.9037	0.0000	0.0931	0.0032
pc1	0.9261	0.0049	0.0641	0.0049
kc1	0.8412	0.0047	0.1490	0.0051
kc3	0.9330	0.0039	0.0633	-0.0002*

* demonstrates that rounding errors occur.

Using our technique it is possible to calculate the Precision and Recall of the faulty units in Elish and Elish's study. Table 11 shows the results of this calculation. Table 11 suggests that the performance of the SVMs in the Elish and Elish study is much less positive. Table 11 shows that f-measure ranges from 0.0 to 0.12. This is compared to their original maximum f-measure of 0.96.

Table 11: Performance Measures for the Faulty Class using the Values from Table 10

Dataset	Accuracy	Precision	Recall	f-measure
cm1	0.9069	1.0000	0.0332	0.0643
pc1	0.9310	0.5000	0.0710	0.1244
kc1	0.8463	0.5204	0.0331	0.0622
kc3	0.9328	-0.0541	-0.0032	-0.0060

Table 8: Accuracy, Precision and Recall [12]

Dataset	Accuracy	Precision	Recall
cm1	0.9069	0.9066	1.0000
pc1	0.9310	0.9353	0.9947
kc1	0.8459	0.8495	0.9940
kc3	0.9328	0.9365	0.9958

Table 9: Frequency of the Class Identified as "True" and the Frequency of the Faulty class

Dataset	computed d	1 - computed d	Reported fault frequency
cm1	0.9037	0.0963	0.097
pc1	0.9311	0.0689	0.069
kc1	0.8462	0.1538	0.154
kc3	0.9370	0.0630	0.063

7. CONCLUSION

The predictive performance of published fault prediction models is expressed using a variety of different performance measures. This makes it difficult to compare the performance of published prediction models. We have presented an approach that enables the recomputation of the confusion matrix for studies originally reporting a variety of performance measures. From the confusion matrix a range of other performance measures can be calculated. Expressing the performance of fault prediction models using a consistent set of measures allows comparative analysis. Our approach has several advantages, including that it:

- allows comparative analysis of a set of fault prediction models in terms of a preferred predictive quality.
- makes meta-analysis possible across the many fault prediction studies published.
- enables the validation of the performance figures reported in published studies.

The advantages of our approach have benefits for fault prediction researchers, practitioners and reviewers. Researchers can use our approach to evaluate predictive performance across sets of models and perform meta-analysis of these models. An evidence base of fault prediction can be built by researchers that will enable more informed future model building research. Practitioners can express model performance to reflect the qualities that they are interested in, for example practitioners wanting a model that values finding as many faults as possible might might predominately focus on Recall. Practitioners are then in a more informed position to select a model that is appropriate for their development context. Reviewers of fault prediction studies can use our process as a relatively easy way to check that no errors have crept into fault prediction studies. Without our 'ready reckoner' checking performance figures in studies submitted for review is difficult. Model builders could themselves use our process as a 'ready reckoner' to check their own figures are correct. Model builders and reviewers doing this checking could improve the quality of the fault prediction work that is published.

Overall the approach that we present could significantly improve the quality of fault prediction studies and enable meta-analysis across studies. Achieving this is very important as it will help this research area to mature and grow. Such maturation could ultimately expand the industrial uptake of fault prediction modelling.

8. REFERENCES

[1] E. Arisholm, L. C. Briand, and M. Fuglerud. Data mining techniques for building fault-proneness models in telecom java software. In *Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on*, pages 215 –224, nov. 2007.

[2] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[3] P. Baldi, S. Brunak, Y. Chauvin, C. Andersen, and H. Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, 2000.

[4] G. Batista, R. Prati, and M. Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29, 2004.

[5] D. Bowes and D. Gray. Recomputing the confusion matrix for prediction studies reporting categorical output. Technical Report 509, University of Hertfordshire, 2011.

[6] C. Catal, B. Diri, and B. Ozumut. An artificial immune system approach for fault prediction in object-oriented software. In *Dependability of Computer Systems, 2007. DepCoS-RELCOMEX '07. 2nd International Conference on*, pages 238 –245, june 2007.

[7] N. V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *SIGKDD Explorations*, 6(1):1–6, 2004.

[8] D. S. Cruzes and T. Dybå. Research synthesis in software engineering: A tertiary study. *Inf. Softw. Technol.*, 53:440–455, May 2011.

[9] A. B. de Carvalho, A. Pozo, S. Vergilio, and A. Lenz. Predicting fault proneness of classes trough a multiobjective particle swarm optimization algorithm.

In *Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on*, volume 2, pages 387–394, 2008.

[10] A. B. de Carvalho, A. Pozo, and S. R. Vergilio. A symbolic fault-prediction model based on multiobjective particle swarm optimization. *Journal of Systems and Software*, 83(5):868–882, 2010.

[11] G. Denaro and M. Pezzè. An empirical evaluation of fault-proneness models. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 241–251, New York, NY, USA, 2002. ACM.

[12] K. O. Elish and M. O. Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649 – 660, 2008.

[13] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. Further thoughts on precision. In *Evaluation and Assessment in Software Engineering (EASE)*, 2011.

[14] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic review of fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, PP(99):1, 2011.

[15] H. He and E. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, pages 1263–1284, 2008.

[16] Y. Jiang, B. Cukic, and Y. Ma. Techniques for evaluating fault prediction models. *Empirical Software Engineering*, 13(5):561–595, 2008.

[17] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. Matsumoto. The effects of over and under sampling on fault-prone module detection. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 196 –204, sept. 2007.

[18] A. Kaur, P. S. Sandhu, and A. S. Bra. Early software fault prediction using real time defect data. In *Machine Vision, 2009. ICMV '09. Second International Conference on*, pages 242–245. accept, 2009.

[19] T. Khoshgoftaar and N. Seliya. Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Software Engineering*, 9(3):229–257, 2004.

[20] A. Koru and H. Liu. Building effective defect-prediction models in practice. *Software, IEEE*, 22(6):23 – 29, nov.-dec. 2005.

[21] O. Kutlubay, B. Turhan, and A. Bener. A two-step model for defect density estimation. In *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on*, pages 322 –332, aug. 2007.

[22] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *Software Engineering, IEEE Transactions on*, 34(4):485 –496, july-aug. 2008.

[23] T. Mende and R. Koschke. Effort-aware defect prediction models. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pages 107–116, 2010.

[24] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald. Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'". *Software Engineering, IEEE Transactions on*, 33(9):637 –640, sept. 2007.

[25] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *Software Engineering, IEEE Transactions on*, 33(1):2 –13, jan. 2007.

[26] T. Ostrand and E. Weyuker. How to measure success of fault prediction models. In *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*, pages 25–30. ACM, 2007.

[27] G. Pai and J. Dugan. Empirical analysis of software fault content and fault proneness using bayesian methods. *Software Engineering, IEEE Transactions on*, 33(10):675 –686, oct. 2007.

[28] N. Pizzi, A. Summers, and W. Pedrycz. Software quality prediction using median-adjusted class labels. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2405 –2409, 2002.

[29] N. Seliya, T. Khoshgoftaar, and S. Zhong. Analyzing software quality with limited fault-proneness defect data. In *High-Assurance Systems Engineering, 2005. HASE 2005. Ninth IEEE International Symposium on*, pages 89 –98, oct. 2005.

[30] Y. Sun, C. Castellano, M. Robinson, R. Adams, A. Rust, and N. Davey. Using pre & post-processing methods to improve binding site predictions. *Pattern Recognition*, 42(9):1949–1958, 2009.

[31] B. Turhan, G. Kocak, and A. Bener. Data mining source code for locating software bugs: A case study in telecommunication industry. *Expert Systems with Applications*, 36(6):9986–9990, 2009.

[32] T. Wang and W.-h. Li. Naive bayes software defect prediction model. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1–4. accept, 2010.

[33] L. Yi, T. M. Khoshgoftaar, and N. Seliya. Evolutionary optimization of software quality modeling with multiple repositories. *Software Engineering, IEEE Transactions on*, 36(6):852–864, 2010.

[34] A. Zeller, T. Zimmermann, and C. Bird. Failure is a four-letter word: a parody in empirical research. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pages 5:1–5:7, New York, NY, USA, 2011. ACM.

[35] H. Zhang and X. Zhang. Comments on "data mining static code attributes to learn defect predictors". *Software Engineering, IEEE Transactions on*, 33(9):635 –637, sept. 2007.

[36] Y. Zhou and H. Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *Software Engineering, IEEE Transactions on*, 32(10):771 –789, oct. 2006.

APPENDIX

List of Papers from which we have Recomputed Confusion Matrices.

E. Arisholm, L. C. Briand, and M. Fuglerud. Data mining techniques for building fault-proneness models in telecom java software. In *ISSRE '07. The 18th IEEE Intern Symp on*, pages 215 –224, 2007.

E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu. Putting it all together: Using socio-technical networks to predict failures. In *20th International Symposium on Software Reliability Engineering*, pages 109–119. IEEE, 2009.

L. Briand, W. Melo, and J. Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *Software Engineering, IEEE Transactions on*, 28(7):706 – 720, 2002.

B. Caglayan, A. Bener, and S. Koch. Merits of using repository metrics in defect prediction for open source projects. In *FLOSS '09. ICSE Workshop on*, pages 31–36, 2009.

G. Calikli, A. Tosun, A. Bener, and M. Celik. The effect of granularity level on software defect prediction. In *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*, pages 531 –536, 2009.

C. Catal, B. Diri, and B. Ozumut. An artificial immune system approach for fault prediction in object-oriented software. In *Dependability of Computer Systems, 2007. DepCoS-RELCOMEX '07. 2nd International Conference on*, pages 238 –245, 2007.

C. Cruz and A. Erika. Exploratory study of a uml metric for fault prediction. In *Proceedings of the 32nd ACM/IEEE Intern Conf on Software Engineering*, pages 361–364. 2010.

A. B. de Carvalho, A. Pozo, S. Vergilio, and A. Lenz. Predicting fault proneness of classes trough a multiobjective particle swarm optimization algorithm. In *Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on*, volume 2, pages 387–394, 2008.

A. B. de Carvalho, A. Pozo, and S. R. Vergilio. A symbolic fault-prediction model based on multiobjective particle swarm optimization. *J of Sys & Soft*, 83(5):868–882, 2010.

G. Denaro and M. Pezzè. An empirical evaluation of fault-proneness models. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 241–251, NY, USA, 2002. ACM.

L. Guo, Y. Ma, B. Cukic, and H. Singh. Robust prediction of fault-proneness by random forests. In *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, pages 417 – 428, 2004.

T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *Software Engineering, IEEE Transactions on*, 31(10):897 – 910, 2005.

Z. Hongyu. An investigation of the relationships between lines of code and defects. In *Software Maintenance, 2009. IEEE Intern Conf on*, pages 274–283, 2009.

Y. Jiang, B. Cukic, and Y. Ma. Techniques for evaluating fault prediction models. *Empirical Software Engineering*, 13(5):561–595, 2008.

S. Kanmani, V. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai. Object-oriented software fault prediction using neural networks. *Information and Software Technology*, 49(5):483–492, 2007.

A. Kaur and R. Malhotra. Application of random forest in predicting fault-prone classes. In *Advanced Computer Theory and Engineering, 2008, Internl Conf on*, 37–43, 2008.

A. Kaur, P. S. Sandhu, and A. S. Bra. Early software fault prediction using real time defect data. In *Machine Vision, 2009. Intern Conf on*, pages 242–245.

T. Khoshgoftaar and N. Seliya. Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Software Engineering*, 9(3):229–257, 2004.

T. Khoshgoftaar, X. Yuan, E. Allen, W. Jones, and J. Hudepohl. Uncertain classification of fault-prone software modules. *Empirical Software Engineering*, 7(4):297–318, 2002.

A. Koru and H. Liu. Building effective defect-prediction models in practice. *Software, IEEE*, 22(6):23 – 29, 2005.

O. Kutlubay, B. Turhan, and A. Bener. A two-step model for defect density estimation. In *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on*, pages 322 –332, 2007.

Y. Ma, L. Guo, and B. Cukic. *Advances in Machine Learning Applications in Software Engineering*, chapter A statistical framework for the prediction of fault-proneness, pages 237–265. IGI Global, 2006.

T. Mende and R. Koschke. Effort-aware defect prediction models. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pages 107–116, 2010.

T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *Software Engineering, IEEE Transactions on*, 33(1):2 –13, 2007.

O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno. Spam filter based approach for finding fault-prone software modules. In *Mining Software Repositories, 2007. ICSE '07. International Workshop on*, page 4, 2007.

O. Mizuno and T. Kikuno. Training on errors experiment to detect fault-prone software modules by spam filter. In *Procs European Software Engineering Conf and the ACM SIGSOFT symp on The foundations of software engineering*, ESEC-FSE '07, pages 405–414, 2007. ACM.

R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th Intern Conf on*, pages 181–190.

N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 309–318.

A. Nugroho, M. R. V. Chaudron, and E. Arisholm. Assessing uml design metrics for predicting fault-prone classes in a java system. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 21–30.

G. Pai and J. Dugan. Empirical analysis of software fault content and fault proneness using bayesian methods. *Software Engineering, IEEE Trans on*, 33(10):675–686, 2007.

A. Schröter, T. Zimmermann, and A. Zeller. Predicting component failures at design time. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 18–27. ACM, 2006.

N. Seliya, T. Khoshgoftaar, and S. Zhong. Analyzing software quality with limited fault-proneness defect data. In *High-Assurance Systems Engineering, 2005. IEEE Internl Symp on*, pages 89 –98, 2005.

S. Shivaji, E. J. Whitehead, R. Akella, and K. Sunghun. Reducing features to improve bug prediction. In *Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on*, pages 600–604.

Y. Singh, A. Kaur, and R. Malhotra. Predicting software fault proneness model using neural network. *Product-Focused Software Process Improvement*, 5089:204–214, 2008.

A. Tosun and A. Bener. Reducing false alarms in software defect prediction by decision threshold optimization. In *Empirical Software Engineering and Measurement, ESEM 2009. International Symposium on*, pages 477–480.

B. Turhan and A. Bener. A multivariate analysis of static code attributes for defect prediction. In *Quality Software, 2007. Intern Conf on*, pages 231 –237, 2007.

O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer, and R. Haesen. Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and Software*, 81(5):823–839, 2008.

R. Vivanco, Y. Kamei, A. Monden, K. Matsumoto, and D. Jin. Using search-based metric selection and oversampling to predict fault prone modules. In *Electrical and Computer Engineering, 2010, Canadian Conf on*, pages 1–6.

L. Yi, T. M. Khoshgoftaar, and N. Seliya. Evolutionary optimization of software quality modeling with multiple repositories. *Soft Engin, IEEE Trans on*, 36(6):852–864, 2010.

Y. Zhou and H. Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *Software Engineering, IEEE Trans on*, 32(10):771–789, 2006.

T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Predictor Models in Software Engineering, 2007. PROMISE'07*, page 9, 2007.