Poster: Bridging Effort-Aware Prediction and Strong Classification - a Just-in-Time Software Defect Prediction Study

Yuchen Guo
Xi'an Jiaotong University
Xian, China
wispcat@stu.xjtu.edu.cn

Martin Shepperd*
Brunel University London
London, UK
Martin.Shepperd@brunel.ac.uk

Ning Li
Northwestern Polytechnical University
Xian, China
lining@nwpu.edu.cn

ABSTRACT

Context: Most research into software defect prediction ignores the differing amount of effort entailed in searching for defects between software components. The result is sub-optimal solutions in terms of allocating testing resources. Recently effort-aware (EA) defect prediction has sought to redress this deficiency. However, there is a gap between previous classification research and EA prediction.
Objective: We seek to transfer strong defect classification capability to efficient effort-aware software defect prediction.
Method: We study the relationship between classification performance and the cost-effectiveness curve experimentally (using six open-source software data sets).
Results: We observe extremely skewed distributions of change size which contributes to the lack of relationship between classification performance and the ability to find efficient test orderings for defect detection. Trimming allows all effort-aware approaches bridging high classification capability to efficient effort-aware performance.
Conclusion: Effort distributions dominate effort-aware models. Trimming is a practical method to handle this problem.

CCS CONCEPTS

• **Software and its engineering** → *Risk management*; *Maintaining software*;

KEYWORDS

Software, defect prediction, effort-aware, just-in-time

ACM Reference format:
Yuchen Guo, Martin Shepperd, and Ning Li. 2018. Poster: Bridging Effort-Aware Prediction and Strong Classification - a Just-in-Time Software Defect Prediction Study. In *Proceedings of 40th International Conference on Software Engineering Companion, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18 Companion),* 2 pages.
https://doi.org/10.1145/3183440.3194992

*Corresponding author.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05…$15.00
https://doi.org/10.1145/3183440.3194992

1 INTRODUCTION

A major cost in software engineering is testing. Though software defect prediction aims to make testing a more focused activity, it assumes that there are equal costs for detecting and repairing any defect, which is far from true in the real world. Cost-effectiveness was highlighted by Arisholm et al. [2] who proposed the cost-effectiveness curve as a new predictive performance measure. They also found that classifier performance is not clearly related to generating good effort-aware (EA) defect prediction. Subsequently, several effort-aware defect prediction models have been proposed [4, 5, 7], however, the connection between classification ability (defective or not) and effort-aware prediction remains unclear.

In this paper, we aim to bridge the gap between high classification capability and effort-aware performance by revisiting six effort-aware models in the context of just-in-time (JiT) defect prediction. The main contributions of this paper are that we: (i) show there is great variability in repair effort which explains why many effort-aware prediction models are unstable when confronted with extreme values of effort; (ii) find a linear relationship between effort-aware performance and classification capability when we trim effort outliers.

2 EFFORT-AWARE DEFECT PREDICTION

We summarize the six different effort-aware prediction approaches to date in Table 1. The basic idea of effort-aware prediction is to find more defects with less effort. To achieve this goal, effort-aware models prioritize software components according to their estimated relative defect risk.

Table 1: Summary of EA Prediction Approaches

No	Model	Learn	Predict	Relative Risk	Reference
1	R_{ad}	defect	$p(x)$	$p(x) \cdot (1 - \frac{E(x)}{E_{max}})$	[5]
2	R_{dd}	defect	$Y(x)$	$\frac{Y(x)}{E(x)}$	[3, 5]
3	R_{ee}	defect	$p(x)$	$\frac{p(x)}{E(x)}$	[6]
4	EALR	$\frac{defect}{effort}$	$R_{LR}(x)$	$R_{LR}(x)$	[4]
5	R_{non-EA}	defect	$p(x)$	$p(x)$	[2, 7]
6	R_{unsup}	-	$\frac{1}{metric}$	$\frac{1}{metric}$	[7]

Note that **(1)** x denotes a software change, $p(x)$ is its probability to be defective. **(2)** $Y(x)$ is the binary prediction whether change x is defective or not. **(3)** $E(x)$ is the effort required to inspect x, and E_{max} is the max value of E(x). **(4)** R_{LR} is the predicted risk of EALR.

Relative risk is the trade-off between the risk of being defective and the effort to inspect the defect. Sorting predictions by relative risk gives the cost-effective order each model predicts. The cost-effectiveness curve proposed by Arisholm et al. [2] simulates inspection on defects following such cost-effective order, then assesses its actual cost-effectiveness.

3 EXPERIMENTAL DETAILS

Our experiment is based on public available scripts and data sets including six open source projects shared by Yang et al. [7] and Kamei et al. [4]. They studied effort-aware models in the context of JiT prediction that focuses on determining whether a software *change* is likely to be risky.

Differing from their work, our paper explores the relationship between classification capability and EA performance. For classification, we use Area under the Curve (AUC) which sums up the potential classification capability.For EA performance, there are two popular performance measures calculated from the cost-effectiveness (CE) curve: (i) P_{opt} that computes the area under the CE curve (normalized version); (ii) *ACC* computes recall of defective components when using 20% of the entire effort required.

Cross-validation is applied to evaluate. To setup EA models i.e. R_{ad}, R_{dd}, R_{ee} and R_{non-EA}, 11 supervised machine learning methods are chosen to build base classifiers: IBk, C4.5, LMT, Random Forest, Naive Bayes, JRip, Ridor, SMO, RBFNet and Logistic Regression and Simple Logistic. For details of algorithms see [7].

4 EXPERIMENTAL RESULTS

4.1 Extreme Distribution of Effort Data

The first finding is that the distribution of effort (measured by churn) is extreme (skewness ≈ 20). For 75% of changes the churn is less than 52 lines but a few large changes can be as exceed 6000 lines plus. Alali et al. [1] found similar characteristics for typical software changes.

Since effort part defines the CE curve, it is crucial to the EA performance. Such extreme distributions will significantly impact the EA performance, so we trim the outliers. Outliers were identified according to Alali et al. [1]'s change size categories. Less than 15% data were trimmed by the cut-line $Q_3 + 1.5 \cdot IQR$.

4.2 A Path and Evidence

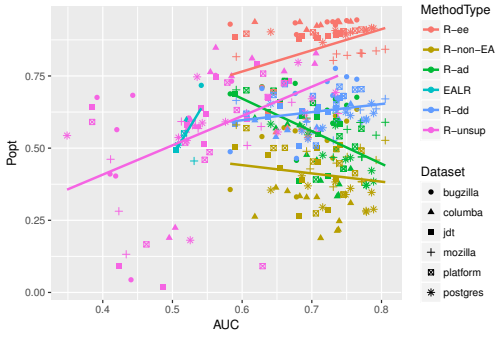

Figure 1: AUC vs P_{opt} including effort outliers

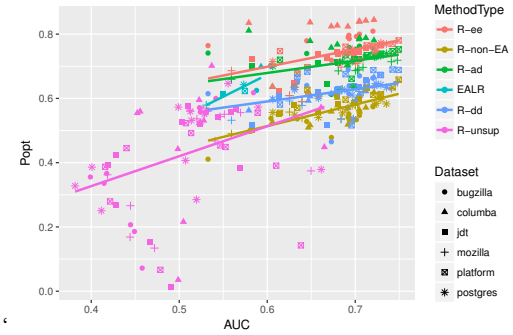

Figure 2: AUC vs P_{opt} excluding effort outliers

Figs. 1 and 2 show the overall picture of effort-aware approaches as scatter-plots before and after trimmed. Observe the lack of positive correlation between AUC and P_{opt} on Fig. 1. This supports the findings of other researchers who also reported unclear or even negative relationships [2, 7]. There are also some interesting datapoints in the bottom right corner: specifically models with strong classification capability that perform badly when considering effort. In contrast, there are some high values ($P_{opt} \approx 0.87$) when $AUC \approx 0.6$, which is unexpected since the classification capability is not much better than random.

Second, there is a trend in Fig. 2 that higher AUC could relate to better EA performance evidenced by the moderate positive correlation (the percentage bend correlation r=0.575). This suggests that we harnessing stronger classifiers to enable better EA performance. The same trends are found for *ACC*. Such contrasts are because all EA models are dominated by the effort distributions.

5 CONCLUSIONS

We believe our results[1] are important to both researchers and software engineering practitioners. For *researchers*, we consider why models with poor classification performance can perform well in effort-aware prediction and *vice versa*. The results demonstrate that all six EA models are vulnerable to skewed effort distributions. For *software engineering practitioners* we offer a guideline to bridge the gap between classification and effort-aware prediction.

REFERENCES

[1] A. Alali, H. Kagdi, and J. Maletic. 2008. What's a typical commit? a characterization of open source software repositories. In *16th IEEE Intl. Conf. on Program Comprehension*. 182–191.
[2] E. Arisholm, L. Briand, and E. Johannessen. 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *J. of Systems & Software* 83, 1 (2010), 2–17.
[3] Y. Kamei, S. Matsumoto, and et al. 2010. Revisiting common bug prediction findings using effort-aware models. In *IEEE Intl. Conf. on Softw. Maint.* 1–10.
[4] Y. Kamei, E. Shihab, and et al. 2013. A large-scale empirical study of just-in-time quality assurance. *IEEE Trans. on Softw. Eng.* 39, 6 (2013), 757–773.
[5] T. Mende and R. Koschke. 2010. Effort-aware defect prediction models. In *14th European Conf. on Software Maintenance and Re-engineering*. 107–116.
[6] Y. Yang, Y. Zhou, and et al. 2015. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. *IEEE Trans. on Softw. Eng.* 41, 4 (2015), 331–357.
[7] Y. Yang, Y. Zhou, and et al. 2016. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In *24th ACM SIGSOFT International Symp. on Foundations of Softw. Eng.* 157–168.

[1] See https://github.com/yuchen1990/EAposter for data, scripts and further analysis.