International Journal of Engineering Science Invention
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org ||Volume 5 Issue 9|| September 2016 || PP. 57-61

Practical Guidelines to Improve Defect Prediction Model – A Review

B. Dhanalaxmi [1], Dr. G. Apparao Naidu[2], Dr.K.Anuradha[3]
[1]*Associate Professor, Institute of Aeronautical Engineering, IT Dept, Hyderabad, TS, India*
[2]*Professor, J.B. Institute of Engineering & Technology, CSE Dept, Hyderabad, TS, India*
[3]*Professor & HOD, GRIET, CSE Dept, Hyderabad, TS, India.*

Abstract: *Defect prediction models are used to pinpoint risky software modules and understand past pitfalls that lead to defective modules. The predictions and insights that are derived from defect prediction models may not be accurate and reliable if researchers do not consider the impact of experimental components (e.g., datasets, metrics, and classifiers) of defect prediction modeling. Therefore, a lack of awareness and practical guidelines from previous research can lead to invalid predictions and unreliable insights. Through case studies of systems that span both proprietary and open-source domains, find that (1) noise in defect datasets; (2) parameter settings of classification techniques; and (3) model validation techniques have a large impact on the predictions and insights of defect prediction models, suggesting that researchers should carefully select experimental components in order to produce more accurate and reliable defect prediction models.*

Keywords: *Software Quality Assurance, experimental components, performance estimates, Support Vector Machines*

I. Introduction

Defect models, which identify defect-prone software modules using a variety of software metrics, serve two main purposes. First, defect models can be used to predict modules that are likely to be defect-prone. Software Quality Assurance (SQA) teams can use defect models in a prediction setting to effectively allocate their limited resources to the modules that are most likely to be defective. Second, defect models can be used to understand the impact that various software metrics have on the defect-proneness of a module.

The insights derived from defect models can help software teams to avoid past pitfalls that lead to defective modules The predictions and insights that are derived from defect prediction models may not be accurate and reliable if researchers do not consider the impact those experimental components (e.g., datasets, metrics, and classifiers) of defect prediction modeling. Indeed, there exists a plethora of research that raise concerns about the impact of experimental components on defect prediction modeling. For example, Sheppard et al. Found that the reported performance of a defect prediction model shares a strong relationship with the group of researchers who construct the models. Their observations suggest that many published defect prediction studies are biased, and calls their validity into question. To assess the impact of experimental components on defect prediction modeling, the association between the reported performance of a defect model and the used experimental components (i.e., datasets, metrics, and classifiers) are considered. Experimental components (i.e., metrics) are used to construct defect prediction models share a stronger relationship with the reported performance than research group does, suggesting that experimental components of defect prediction modeling may impact the conclusions of defect prediction studies.

This paper, investigate the impact of (1) noise in defect datasets and (2) parameter settings of classification techniques have on the predictions and insights of defect prediction models. In addition, defect prediction models may produce an unrealistic estimation of model performance when inaccurate and unreliable model validation techniques are applied, which could lead to incorrect model selection in practice and unstable conclusions of defect prediction studies. Thus, the impact of (3) model validation techniques have on the accuracy and reliability of performance estimates that are produced by defect prediction models. .

II. Typical Software Defect Prediction Model

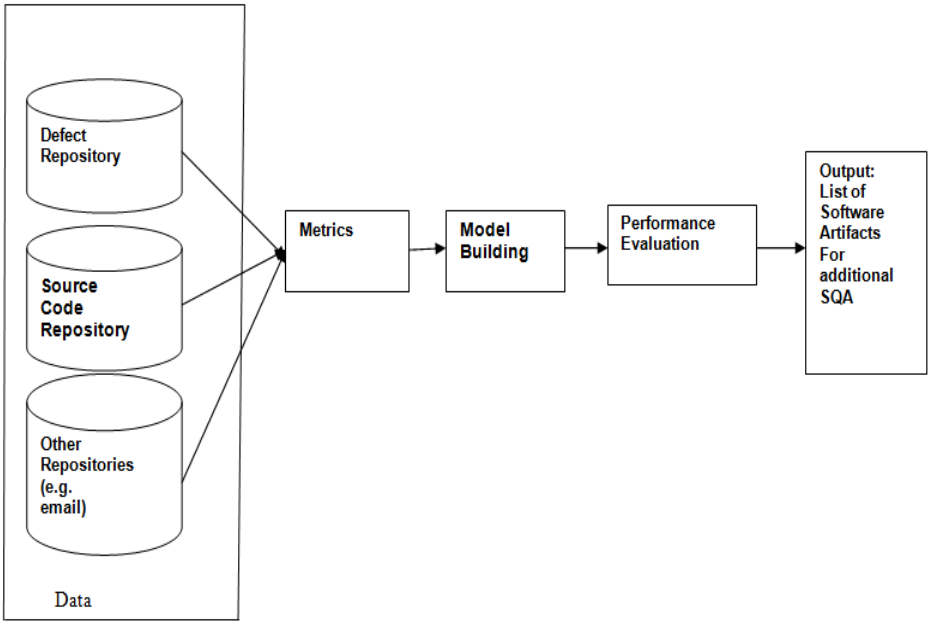

Fig.1. Overview of Software Defect Prediction model

The main goal of most software defect prediction studies is (1) to predict where defects might appear in the future and (2) to quantify the relative importance of various factors (i.e., independent variables) used to build a model. Ideally, these predictions will correctly classify software artifacts (e.g., subsystems and files) as being defect-prone or not and in some cases may also predict the number of defects, defect density (the number of defects / SLOC) and/or the likelihood of a software artifact including a defect. Figure 1 shows an overview of the software defect prediction process. First, data is collected from software repositories, which archive historical data related to the development of the software project, such as source code and defect repositories. Then, various metrics that are used as independent variables (e.g., SLOC) and a dependent variable (e.g., the number of defects)are extracted to build a model. The relationship between the independent variables and dependent variable is modeled using statistical techniques and machine learning techniques.
Finally, the performance of the built model is measured using several criteria such as precision, recall and AUC (Area Under the Curve) of ROC (the Receiver Operating Characteristic).Each of the four aforementioned steps are explained next.

A. Data

In order to build software defect prediction models, a number of metrics that make up the independent and dependent variables are needed. Large software projects often store their development history and other information, such as communication, in software repositories. Although the main reason for using these repositories is to keep track of and record development history, researchers and practitioners realize that this repository data can be used to extract software metrics. For example, prior work used the data stored in the source control repository to count the number of changes made to a file and the complexity of changes, and used this data to predict files that are likely to have future defects. Software defect prediction work generally leverages various types of data from different repositories, such as (1) source code repositories, which store and record the source code and development history of a project, (2) defect repositories, which track the bug/defect reports or feature requests filed for a project and their resolution progress and (3) mailing list repositories, which track the communication and discussions between development teams. Other repositories can also be leveraged for defect prediction.

B. Metrics

When used in software defect prediction research, metrics are considered to be independent variables, which mean that they are used to perform the prediction (i.e., the predictors). Also, metrics can represent the dependent variables, which means they are the metrics being predicted (i.e., these can be pre- or post-release defects). Previous defect prediction studies used a wide variety of independent variables (e.g., process, organizational or code metrics to perform their predictions. Moreover, several different metrics were used to represent the dependent variable as well. For example, previous work predicted different types of defects (e.g., pre-release, post release], or both).

C. Model Building

Various techniques, such as linear discriminant analysis , decision trees , Naive Bayes , Support Vector Machines and random forest, are used to build defect prediction models. Each of the aforementioned techniques have their own benefits (e.g., they provide models that are robust to noisy data and/or provide more explainable models).Generally speaking, most defect prediction studies divide the data into two sets: a training set and a test set. The training set is used to train the prediction model, whereas the testing set is used to evaluate the performance of the prediction model.

D. Performance Evaluation

Once a prediction model is built, its performance needs to be evaluated. Performance is generally measured in two ways: predictive power and explanative power.

Predictive Power: Predictive power measures the accuracy of the model in predicting the software artifacts that have defects. Measures such as precision, recall, f-measure and AUCROC, which plots the false positive rate on the x-axis and true positive rate on the y-axis over all possible classification thresholds, are commonly-used in defect prediction studies.

Explanative Power: In addition to measuring the predictive power, explanatory power is also used in defect prediction studies. Explanative power measures how well the variability in the data is explained by the model. Often the R^2 or deviance explained measures are used to quantify the explanative power. Explanative power is particularly useful since it enables us to measure the variability explained by each independent variable in the model, providing us with a ranking as to which independent variable is most "useful".

III. Three Major Experimental Components In Defect Prediction Modeling:

This paper addresses three major experimental components in defect prediction modeling:

1. Overlooking noise generated by issue report mislabeling and investigates the impact that realistic noise generated by issue report mislabeling has on the predictions and insights of defect prediction models.
2. Overlooking the optimal parameter settings of classification techniques and investigate the impact that parameter settings of classification techniques have on the accuracy and reliability of the performance of defect prediction models when automated parameter optimization is applied.
3. Overlooking the most accurate and reliable model validation techniques and investigate the impact that model validation techniques have on the accuracy and reliability of the performance of defect prediction models.

1) Overlooking noise generated by issue report mislabeling

Motivation

The accuracy and reliability of a prediction model depends on the quality of the data from which it was trained. Therefore, defect prediction models may be inaccurate and unreliable if they are trained using noisy data. Recent research shows that noise that is generated by issue report mislabeling, i.e., issue reports that describe defects but were not classified as such (or vice versa), may impact the performance of defect models . Yet, while issue report mislabeling is likely influenced by characteristics of the issue itself e.g., novice developers may be more likely to mislabel an issue than an experienced developer - the prior work randomly generates mislabeled issues.

Approach.

Investigate whether mislabeled issue reports can be accurately explained using characteristics of the issue reports themselves, and what is the impact that a realistic amount of noise has on the predictions and insights derived from defect models. Using the manually curated dataset of mislabeled issue reports provided by Herzig et al. generate three types of defect datasets: (1) realistic noisy datasets that contain mislabeled issue reports as classified manually by Herzig et al., (2) random noisy datasets that contain the same proportion of mislabeled issue reports as contained in the realistic noisy dataset, however the mislabeled issue reports are selected at random, and (3) clean datasets that contain no mislabeled issues.

Results

The results find that (1) issue report mislabeling is not random; (2) precision is rarely impacted by mislabeled issue reports; (3) however, models trained on noisy data typically achieve 56%-68% of the recall of models trained on clean data; and (4) only the metrics in top influence rank of defect models are robust to the noise introduced by mislabeling.

2) Overlooking the parameters of classification Techniques

Motivation.

Defect prediction models are classifiers that are trained to identify defect-prone software modules. Such classifiers have configurable parameters that control their characteristics (e.g., the number of trees in a random forest classifier). Recent studies show that these classifiers may underperform due to the use of suboptimal default parameter settings However, it is impractical to assess all of the possible settings in the parameter spaces.

Approach

Literature analysis reveals that 26 of the 30 most commonly-used classification techniques (87%) require at least one parameter setting. Since such parameter settings may impact the performance of defect prediction models, the settings should be carefully selected. then investigate the improvement and the reliability of the performance of defect prediction models when Caret an off the-shelf automated parameter optimization technique is applied. Caret evaluates candidate parameter settings and suggests the optimized setting that achieves the highest performance.

Results

The results find that (1) Caret improves the AUC performance of defect prediction models by up to 40 percentage points; and (2) Caret-optimized classifiers are at least as reliable as classifiers that are trained using the default settings. Our results lead us to conclude that parameter settings can indeed have a large impact on the performance of defect prediction models, suggesting that researchers should experiment with the parameters of the classification techniques.

3) Overlooking the most accurate and reliable model validation techniques

Motivation: Prediction models may provide an unrealistically optimistic estimation of model performance when (re)applied to the same sample with which that they were trained. To address this problem, Model Validation Techniques (MVTs) (e.g., k-fold cross-validation) are used to estimate how well a model will perform on unseen data. Recent research has raised concerns about the accuracy (i.e.,how much do the performance estimates differ from the ground truth?) and reliability (i.e., how much do performance estimates vary when the experiment is repeated?) of model validation techniques when applied to defect prediction models [9]. An optimal MVT would not overestimate or underestimate the ground truth performance. Moreover, the performance estimates should not vary broadly when the experiment is repeated. However, little is known about how accurate and reliable the performance estimates of MVTs tend to be.

Approach

Investigate the accuracy and the reliability of performance estimates when 10 MVTs (i.e., hold-out validation, k-fold validation and bootstrap validation) are applied. We measure in terms of 5 threshold-dependent and -independent performance measures (e.g., precision, recall, AUC) and evaluate using different types of classification techniques.

Results

Find that (1) the advanced bootstrap validation is the most accurate and the most reliable model validation technique; and (2) the holdout family is the least accurate and most reliable model validation technique in terms of both threshold-dependent and threshold-independent performance measures.

IV. Conclusion

In this paper, the impact of experimental components has on the predictions and insights of defect prediction models. Through case studies of systems that span both proprietary and open-source domains, demonstrate that the experimental components (e.g., metric family) that are used to construct defect prediction models share a stronger relationship with the reported performance than research group does. Noise generated by issue report mislabeling has an impact on the predictions and insights of defect prediction models. Parameter settings of classification techniques have an impact on the accuracy and reliability of the performance of defect prediction models. Model validation techniques have an impact on the accuracy and reliability of the performance estimates that are produced by defect prediction models.

V. Future Work

The results indicates that (1) noise in defect datasets;(2) parameter settings of classification techniques; and (3)model validation techniques have a large impact on the predictions and insights of defect prediction models. The researchers should

1. Experiment with a broader selection of datasets and metrics in order to maximize external validity.
2. Carefully mitigate co linearity issues prior to analysis in order to maximize internal and construct validity.
3. Examine the choice of metrics when building defect prediction models in order not to produce under-performing models.
4. Clean mislabeled issue reports in order to improve the ability to identify defective modules.
5. Interpret or make decisions based on the top influence metrics of defect prediction models when they are trained on noisy data.
6. Apply automated parameter optimization in order to improve the performance and reliability of defect prediction models.
7. Avoid using the holdout validation and instead opt to use the advanced bootstrap model validation technique in order to produce more accurate and reliable performance estimates.

References

[1]. C. Bird, A. Bachmann, E. Aune, J. Du_y, A. Bernstein,V. Filkov, and P. Devanbu. Fair and Balanced? Bias in Bug-Fix Datasets. In Proceedings of the joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering(ESEC/FSE), pages 121{130, 2009.

[2]. B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In Proceedings of the International Conference on Software Engineering (ICSE), pages 789{800, 2015.

[3]. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Coun-sell. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. IEEE Transactions on Software Engineering, 38(6):1276{1304, Nov. 2012.

[4]. K. Herzig, S. Just, and A. Zeller. It's not a Bug, it'sa Feature: How Misclassification Impacts Bug Prediction. In Proceedings of the International Conference on Software Engineering (ICSE), pages 392{401, 2013.

[5]. S. Kim, H. Zhang, R. Wu, and L. Gong. Dealing with noise in defect prediction. In Proceedings of the International Conference on Software Engineering (ICSE),pages 481{490, 2011.

[6]. S. Lessmann, B. Baesens, C. Mues, and S. Pietsch.Benchmarking Classi_cation Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering, 34(4):485{496, 2008.

[7]. T. Menzies and M. Shepperd. Special issue on repeatable results in software engineering prediction. Empir-ical Software Engineering, 17(1-2):1{17, 2012.

[8]. I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. IEEE Transactions on Software Engineering, 31(5):380{391, 2005.

[9]. S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, 1994.

[10]. W. W. Cohen, "Fast Effective Rule Induction," in Proceedings of the International Conference on Machine Learning, 1995, pp. 115–123.

[11]. T. Cover and P. Hart, "Nearest neighbor pattern classification," IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21–27, 1967.

[12]. M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," Empirical Software Engineering, vol. 17, no. 4-5, pp. 531–577, 2012.

[13]. J. Demsar, "Statistical comparisons of classifiers over ˇ multiple data sets," The Journal of Machine Learning Research, vol. 7, pp. 1–30, 2006.

[14]. T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," Machine learning, vol. 40, no. 2, pp. 139–157, 2000.

[15]. L. Erlikh, "Leveraging legacy system dollars for ebusiness," IT professional, vol. 2, no. 3, pp. 17–23, 2000.

[16]. C. Fraley and A. E. Raftery, "Bayesian regularization for normal mixture estimation and model-based clustering," Journal of Classification, vol. 24, no. 2, pp. 155–181, 2007.

[17]. Y. Freund and R. E. Schapire, "Experiments with a New Boosting Algorithm," in Proceedings of the International Conference on Machine Learning, 1996, pp. 148–156.

[18]. B. R. Gaines and P. Compton, "Induction of ripple-down rules applied to modeling large databases," Journal of Intelligent Information Systems, vol. 5, no. 3, pp. 211– 228, 1995.

[19]. B. Goel and Y. Singh, "Empirical investigation of metrics for fault prediction on object-oriented software," in Computer and Information Science. Springer, 2008, pp. 255–265.

[20]. T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," IEEE Transactions on Software Engineering, vol. 31, no. 10, pp. 897–910, 2005.

[21]. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," IEEE Transactions on Software Engineering, vol. 38, no. 6, pp. 1276–1304, 2012.

[22]. G. Hamerly and C. Elkan, "Learning the k in k-means," in Advances in Neural Information Processing Systems, 2004, pp. 281–288.

[23]. K. Hammouda and F. Karray, "A comparative study of data clustering techniques," Fakhreddine Karray University of Waterloo, Ontario, Canada, 2000.

[24]. A. E. Hassan, "Predicting faults using the complexity of code changes," in Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society, 2009, pp. 78–88.

[25]. T. K. Ho, "The random subspace method for constructing decision forests," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 8, pp. 832–844, 1998.

[26]. E. Jelihovschi, J. C. Faria, and I. B. Allaman, "Scottknott: A package for performing the scott- knott clustering algorithm in r," Trends in Applied and Computational Mathematics, vol. 15, no. 1, pp. 003–017, 2014.

[27]. T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013. IEEE, 2013, pp. 279–289.

[28]. K. Kaur, K. Minhas, N. Mehan, and N. Kakkar, "Static and dynamic complexity analysis of software metrics," World Academy of Science, Engineering and Technology, vol. 56, p. 2009, 2009.

[29]. H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan, "Prioritizing the devices to test your app on: a case study of android game apps," in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014, pp. 610–620.