SOME MACHINE LEARNING ASSISTED SOFTWARE BUG PREDICTION TECHNIQUES

The thesis submitted in partial fulfilment

for the Award of Degree

DOCTOR OF PHILOSOPHY

by

RAKESH KUMAR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY

(BANARAS HINDU UNIVERSITY)

VARANASI -221005

INDIA

Roll No.: 18071011 **Year: 2023**

Chapter 8

Conclusion and Future Work

"A conclusion is simply the place where you got tired of thinking."- Dan Chaon

In this thesis, we explored and put forward four distinct approaches for predicting bugs in software. We also delved into the various challenges that come with SBP and explained how our methods tackle these challenges. We reexamined existing solutions to these challenges and provided an analysis. This chapter presents the overall conclusion for the thesis in Section 8.1, where we examine the outcomes of the thesis objectives. Additionally, we present potential areas for future research based on the findings made in this thesis in Section 8.2.

8.1 Conclusions

In today's world, software plays an integral role across various domains, and the pursuit of high-quality software is a constant endeavor. Nonetheless, uncontrollable factors during the software development process inevitably result in bugs, potentially leading to adverse consequences. Detecting and rectifying these bugs before software delivery is crucial for enhancing software reliability. To achieve this, researchers have introduced SBP techniques that leverage statistical and machine learning methods to identify modules at high risk of bugs by analyzing historical development data from

software repositories. These identified high-risk modules are then recommended for priority review by developers or testers, streamlining resource allocation for software testing, enhancing development efficiency, and bolstering software quality assurance.

In this thesis, our significant contribution and findings begin by proposing the most widely used classification-based supervised SBP model (WMV). We encountered that WMV can only be employed on labeled datasets. So, to overcome this limitation, we have developed a classification-based unsupervised SBP method (TCL/T-CLP). Further, we analyzed that only predicting binary class is not much informative to SQA team, then we expanded this TCL/TCLP by developing a regression-based unsupervised SBP method (MTB/MTBP), which can predict bug count value. Additionally, we found that most of the SBP models are developed on OOP datasets, So we extended this and worked on the FP datasets. NO FP bug datasets are available, so we created four Haskell datasets and developed a classification-based unsupervised SBP method (UMV) for predicting bugs in each function of Haskell, which is an FP language. The conclusion of each contribution is presented in the following paragraphs.

WMV: Classification-based supervised SBP- Developing a dependable and versatile Software Bug Prediction (SBP) technique for newly created software projects using traditional machine learning (ML) algorithms alone has proven to be challenging. To tackle this issue, we've introduced a reward-based majority voting ensemble approach, WMV. Our experimental findings demonstrate that WMV surpasses both the Single Majority Voting (SMV) and Base Classifiers (BCs), offering higher performance. In comparison to state-of-the-art (SOTA) techniques, WMV consistently outperforms them, with an average F-measure improvement ranging from 0.05 to 0.53. This study establishes WMV as an effective method for building SBP models, and it recognizes Random Forest (RF) as the top-performing BC, while Least Squares Support Vector Machine with Polynomial Kernel (LSSVM-P) and STACKING emerge as the leading SOTA techniques in terms of mean F-measure.

TCL/TCLP: Classification-based unsupervised SBP - Developing a robust and universally applicable approach for SBP in newly developed software projects or those with limited historical data is a formidable task. In response to the limitations of existing methods, we introduce the TCL/TCLP approach, designed to create automated and high-performing SFP models for unlabeled datasets. In our empirical study across 28 software projects with varying software metric types, TCL/TCLP consistently demonstrates superior or comparable performance to state-of-the-art techniques in accuracy, F-measure, and MCC. The results from tables, box plots, and statistical analyses collectively affirm TCL/TCLP's potential for SFP on unlabeled datasets, all without the need for human intervention. Our experiments also validate TCL/TCLP's resilience to class imbalance issues.

MTB/MTBP: Regression-based unsupervised SBP - The MTB/MTBP approach demonstrates performance that is either superior or on par with standard machine learning models, excelling particularly in terms of average MAE, MRE, and Pred(l)_Error across a range of datasets. The effect size analysis reveals only negligible deviation from baseline models, while the p-value analysis firmly establishes MTBP's significant outperformance of most baseline models. When scrutinizing performance through boxplots across 22 datasets, MTBP consistently equals or outperforms the majority of supervised regression models, ranking favorably with respect to MAE, MRE, and Pred(l)_Error. Notably, MTBP's performance is heavily influenced by the number of selected software metrics, with the ability to accurately predict up to 7 bugs when employing a maximum of 7 metrics. In summary, MTB/MTBP emerges as a promising technique, exhibiting its effectiveness relative to existing machine learning methods.

UMV: Classification-based unsupervised SBP for FP - This research focuses on developing SBP in the functional paradigm, particularly within Haskell packages where no bug datasets exist. Four functional paradigm datasets are created, each containing software source code metrics for Haskell functions, aimed at classifying functions into bug-prone or non-bug-prone categories using statistics

and machine learning models. A novel SDP model, UMV, is introduced, utilizing transformation techniques to reduce metric skewness and establish accurate thresholds. This method can eliminate the need for analyzing the entire source code. UMV demonstrates promising performance across Haskell datasets, outperforming threshold-based and unsupervised machine learning models, although a supervised random forest model achieves the best results. This study marks the first attempt at developing an SDP system for the functional paradigm. UMV exhibits strong potential for future applications in functional programming languages like Standard Meta Language (SML), List Processing Language (LISP), and Clean, as well as software engineering tasks such as refactoring and code reviews.

The results of the objectives outlined above validate the accomplishment of the thesis's predefined goals, with the proposed models effectively addressing the intended issues. Both the experimental and theoretical findings presented in this thesis signify a substantial contribution to the field of software engineering.

8.2 Future Work

Within this thesis, we have conducted an analysis and exploration of various concerns in four distinct SBP scenarios. We presented feasible solutions to address these issues/concerns. Although some progress has been made in the current work, some aspects still need to be expanded and improved. Furthermore, some aspects have yet to be investigated, necessitating additional exploration. Here, we outline potential avenues for future work as follows:

WMV: Classification-based supervised SBP- In the future, it would be intriguing to delve into advanced Deep-learning models to enhance SBP prediction. Exploring the inclusion of additional relevant software metrics that might contribute to SBP in software projects is another fascinating avenue of research. Additionally, investigating a more efficient weight-evaluation scheme within the traditional ensemble method presents an interesting prospect. Further application of the proposed

SBP technique in real-world software, such as social networking apps, online marketing software, project team management software, and numerous other potential software, holds promise. Integrating deep learning within the ensemble learning phase offers the potential for superior results and effective handling of class imbalance concerns. Moreover, incorporating suitable optimization techniques, vectorization methods, and broadcasting techniques could further elevate the effectiveness of SBP. Extending the WMV approach to predict bug count problems is a compelling direction for future research.

TCL/TCLP: Classification-based unsupervised SBP - The intriguing avenues for future research in unsupervised Software Bug Prediction (SBP) can be delineated as follows. Firstly, it would be captivating to investigate a software metrics threshold derivation model that operates as a fully online process, negating the need for pre-extracting software metrics and then deriving metric thresholds. Secondly, exploring alternative and effective metric threshold evaluation approaches and conducting comparative analyses would be a compelling endeavor. Thirdly, the application of our proposed models in real-time scenarios, such as predicting the spread of fake news and predicting abnormalities in the human body, which can be classified as binary—viral generated diseases or non-viral generated diseases—presents an intriguing area for exploration. Lastly, delving into alternative unsupervised learning models for SBP could yield valuable insights and advancements in this field.

MTB/MTBP: Regression-based unsupervised SBP - The research work presents several promising directions for future exploration. Firstly, the model's potential applicability in cross-project bug count prediction. Secondly, there is potential value in estimating testing effort by considering bug size and integrating testing methodologies with design metrics. Thirdly, the extension of similar experiments to other open-source software systems with multiple versions of datasets. Fourth, fine-tuning hyperparameters stands as a potential avenue for enhancing results. Fifth, the utilization of transfer learning on design metrics to predict bug

counts in successive software versions holds considerable potential. It is also imperative for the software industry to contribute datasets for research purposes to strengthen the development of more robust prediction models.

UMV: Classification-based unsupervised SBP for FP - In the future, we plan to expand our work in several ways. We aim to create additional datasets for the functional paradigm (FP) to enhance the comprehensiveness of our results. Furthermore, we intend to apply the UMV method to other functional programming languages, such as SML, LISP, and Clean, opening up new possibilities for SBP. Additionally, our proposed UMV technique can find applications in software engineering beyond SBP, including refactoring and code review challenges. We anticipate the inclusion of more software metrics to improve the system's performance, as only six software metrics were considered initially. We also plan to overcome the challenges we encountered during metric extraction, as some tools did not function optimally with certain Haskell packages. Our research, primarily focused on the functional paradigm, lays the foundation for potential exploration of its applicability in other programming paradigms, such as object-oriented, aspect-oriented, and procedural paradigms, broadening its scope and impact.

Most of the existing and proposed SBP techniques are employed only on public datasets. So, there is a need to collaborate with the software industry, which can help to share commercial or private datasets. These datasets can be used to validate these existing SBP models and evaluate the performance of SBP techniques on real software project datasets. Nowadays, Android applications are gaining increasing prominence in daily life, and their reliability has become a widespread concern. The future work will involve analyzing and researching SBP problems specific to Android software projects. Additionally, while the current research in this thesis primarily employs various machine learning methods to identify the presence of bugs within software modules without distinguishing bug types, future research endeavors will consider software bug type prediction. Additionally, SMs are not always sufficient to analyze the source code, so source code level SBP using abstract syntax tree (AST),

control flow graph (CFG), etc. can be used. Lastly, the thesis presently approaches SBP as a machine learning challenge, neglecting the interaction between SBP and the broader software design, development, testing, and maintenance processes. In forthcoming studies, the plan is to integrate the results of SBP with the software development process, facilitating a comprehensive exploration of the practical applications of machine learning-assisted SBP techniques.

We have developed the SBP/SBCV prediction models based on the software metrics values of the software source code. So, the performance of these models is directly proportional to the quality of software metrics collected. Currently, we have not considered factors like developer experience, coding standards, etc. Third-party tools are used for tasks like feature extraction, model training, etc. Errors or biases in these tools could influence results. This is a threat to the validity of the results. In this thesis, we have focused only on binary classification and regression. More advanced tasks like defect type prediction, severity prediction, etc., are interesting and comprehensive tasks. We will work on these problems in the future. The suggested SBP methods depend on software metrics as features; however, software metrics might not account for all the variables that affect software quality, including organizational, environmental, and human aspects. As a result, adding more features or data sources could enhance the techniques' functionality and generalizability. Extraction of FP metrics, like laziness, higher-order functions, lambda functions, currying, etc., is not done because we could not find any supporting software metrics extraction tool.