Detecting Application Defects Using Inter-Project Comparison A Review of System Extension on Category Inequality

Miss.[1]Jyothika K R , [2] Vaishnavi R S

[1]Assistant Professor, Department of MCA, BIET, Davanagere
[2] Student,4th Semester MCA, Department of MCA, BIET, Davanagere

ABSTRACT

Software fault prediction plays a vital role in enhancing the quality and reliability of software systems. Despite its importance, it encounters key challenges such as class imbalance in fault datasets and the difficulty of building models that generalize effectively across various projects. This study explores these issues through a cross-project analysis, focusing on three central research questions. First, we investigate the impact of class imbalance on prediction accuracy, revealing how disparities in data distribution hinder classifier performance. By applying multiple classification algorithms to a range of datasets from distinct software projects, we demonstrate the need to address imbalance for dependable fault prediction. Second, we assess the effectiveness of cross-project predictions, examining how models trained on one project perform when applied to others. Our results underline the importance of selecting training datasets that share similar traits with the target project to achieve better generalization. Third, we explore how expanding the training dataset with samples from different projects affects prediction outcomes, highlighting the advantages of cross-project learning. Additionally, we provide a detailed comparison of performance metrics such as accuracy, precision, recall, and F1-score across different classifiers. Overall, this research not only underscores the challenges inherent in fault prediction but also offers practical insights into overcoming them, thereby contributing to the development of more robust and generalizable predictive models in software engineering.

Keywords: Software Fault Prediction, Class Imbalance, Cross-Project Analysis, Model Generalization, Predictive Modeling, Software Quality, Classifier Performance, Accuracy, Precision, Recall, F1 Score, Machine Learning, Fault Data, Software Reliability.

I.INTRODUCTION

In the evolving landscape of software engineering, ensuring the reliability and quality of software systems remains a pressing concern. One of the critical challenges in this domain is the accurate **detection of application defects**, which, if left unresolved, can lead to system failures, increased maintenance costs, and reduced user satisfaction. Traditionally, software defect prediction has been carried out using **Within-Project Defect Prediction (WPDP)** models, which rely solely on historical data from the same project. While effective in isolated cases, these models often fall short when data is scarce or when attempting to generalize to new, unseen projects.

To address this limitation, recent advancements have shifted toward Inter-Project Defect Prediction (IPDP) — a methodology that leverages data from multiple, often heterogeneous, software projects to predict defects in a target project. This approach introduces the potential for broader generalization and practical applicability in real-world scenarios where comprehensive data from a single project may not be available. However, the success of IPDP is hindered by category inequality an issue wherein the distribution of defective and non-defective instances (i.e., classes) is highly imbalanced, causing predictive models to be biased and unreliable.

This research investigates the impact of category (class) inequality and explores effective strategies to extend and improve current systems for detecting defects through inter-project comparison. The primary focus lies in understanding how class imbalance skews model performance, and how model generalization can be enhanced when training on datasets originating from diverse software projects.

By conducting an extensive review and experimentation with widely-used classification algorithms, this study aims to develop insights into the design of robust defect prediction models. These models must be capable of overcoming the limitations of imbalanced data distributions and adapting across varying project environments. The outcome of this research contributes significantly to software quality assurance by proposing a refined and scalable approach to software fault prediction that is resilient in the face of data inequality and project variability. As software systems grow in complexity and scale, the demand for accurate and transferable defect prediction models has become more critical than ever.

II.RELATED WORK

The domain of software defect prediction has seen significant advancements in recent decades, with research evolving from Within-Project Defect Prediction (WPDP) models to more generalized Inter-Project Defect Prediction (IPDP) techniques. While WPDP models utilize historical data from the same project, they often suffer from data scarcity and poor generalization when applied to new or early-phase projects. This limitation led researchers to explore cross-project learning, where predictive models are trained on one or more source projects and tested on a different target project.[1]

Early studies, such as those by Chawla et al., introduced resampling techniques like SMOTE (Synthetic Minority Over-sampling Technique) to address the persistent issue of class imbalance, where defective modules are significantly outnumbered by non-defective ones. This imbalance can severely impair classifier performance by biasing predictions toward the majority class.[2]

He and Wu extended this idea with ADASYN (Adaptive Synthetic Sampling), which dynamically generates synthetic minority class samples based on local data distribution. These techniques improved within-project performance but yielded mixed results in cross-project scenarios due to data heterogeneity across software systems.[3]

To improve generalization across projects, researchers like Zhang et al. and Kaliraj et al. explored the use of ensemble learning and optimized feature selection, respectively. Ensemble methods combined the strengths of multiple classifiers to enhance predictive robustness, while optimized feature selection aimed to identify universal software metrics that are effective across diverse projects.[4]

Further innovations included Bayesian Regularization techniques by Mahajan et al., and deep learning models such as Deep Belief Networks (DBDNN) and Graph Convolutional Networks (GCN) for structural bug prediction. These models attempted to capture complex relationships in software metrics, especially when used in conjunction with cost-sensitive learning and domain adaptation to overcome inter-project variability.[5]

Elci and Nandagopalan introduced the Weighted Dual Cross-Recurrent Network-based Levy Sparrow Search (WDCRN-LSS) model, which emphasized early fault prediction to reduce software development overheads. Similarly, Yu et al. proposed ConPredictor, a hybrid model combining static and dynamic analysis to detect faults in concurrent programs, effectively supporting both WPDP and IPDP.[6]

In tackling project heterogeneity, Arora and Kaur proposed the Heterogeneous Fault Prediction (HFP) framework using supervised learning with tailored feature sets. Their model emphasized the importance of project-specific features in enhancing cross-project accuracy.

Despite the diversity of techniques, a consistent challenge across all models is the category inequality that leads to model bias and decreased recall for minority classes. The evolving consensus in the literature suggests that balancing techniques, feature alignment, and intelligent sampling must be jointly considered when designing effective inter-project defect prediction models.[7]

This review underscores the necessity for continued innovation in models that not only correct for class imbalance but also accommodate the structural and contextual differences between software projects. The current study builds upon these foundations to propose a more robust defect prediction framework using refined classifier tuning and inter-project data augmentation.[8]

Recent advancements have also explored unsupervised and semi-supervised learning approaches to handle the lack of labeled data in target projects. Techniques like Latent Dirichlet Allocation (LDA) and clustering-based models have been used to uncover latent defect-prone patterns without relying heavily on labels. While these models avoid the dependency on labeled data, their predictive accuracy often lags behind supervised models. However, when combined with transfer learning or domain adaptation strategies, unsupervised methods show promise in bridging the distributional gap between source and target domains. This integration has laid the groundwork for adaptive learning systems that continuously refine themselves as more project data becomes available.[9]

Moreover, multi-objective optimization techniques like genetic algorithms have been applied to optimize classifier parameters and feature sets simultaneously, achieving a trade-off between accuracy and generalization. The inclusion of cost-sensitive loss functions further enhances model performance in class-imbalanced environments by penalizing the misclassification of defective instances more severely. Studies by Kaliraj and Jaiswal demonstrate how such strategies significantly elevate performance metrics like recall and F1-score in cross-project scenarios. Despite these improvements, the need for standardized benchmarking datasets and evaluation protocols remains a key challenge. Without uniform testing conditions, comparing model performance across studies becomes difficult, highlighting the necessity for reproducible and robust evaluation frameworks in future research.[10]

III.METHODOLOGY

The methodology adopted in this research encompasses a systematic framework designed to investigate the effectiveness of inter-project defect detection and address the persistent issue of category inequality (class imbalance). The approach integrates multiple phases, including data collection, preprocessing, information retrieval, interface design, and evaluation. Each phase contributes to building a robust, generalizable fault prediction model capable of delivering reliable results across diverse software projects.

Data Collection:

Data was collected from widely recognized software engineering repositories such as the PROMISE and NASA MDP datasets. These repositories include software metric data from multiple open-source and industrial projects, with each module labeled as either defective or non-defective. The datasets used support cross-project evaluation by offering diverse project characteristics. Their availability ensures reproducibility and comparative analysis across multiple research efforts.

Preprocessing:

The collected data often contained inconsistencies such as missing values, duplicates, and skewed distributions. Preprocessing involved steps like missing value imputation, data normalization, and feature encoding to ensure uniformity across datasets. To address class imbalance, techniques like SMOTE and ADASYN were applied to synthesize samples of the minority class. Feature selection was also conducted to retain the most relevant attributes for defect prediction.

Information Retrieval:

Information retrieval focused on extracting and organizing relevant software metrics, such as lines of code, complexity, coupling, and cohesion, from the datasets. These metrics were structured into machine-readable formats suitable for model training and evaluation. Retrieval included parsing structured files and integrating related documentation to ensure contextual understanding. This phase ensured the quality and integrity of data before it entered the prediction pipeline.

User Interface Design:

A simple and intuitive web interface was designed using HTML, CSS, and JavaScript, integrated with the Django framework. The interface enables users to upload datasets, select models, and view predictions along with evaluation metrics like precision, recall, and F1-score. Interactive elements such as data visualization and charts were included to aid interpretation. The UI emphasizes usability and accessibility, making defect prediction insights comprehensible for end users.

Integration and Testing:

The backend logic built with Python and Django ORM was integrated with the front-end interface to form a fully functional defect prediction system. System testing ensured smooth data flow and accurate interaction between components like model execution, data processing, and user commands. Cross-project datasets were used to validate the robustness of the system under varying data characteristics. Rigorous unit, integration, and user-acceptance tests were performed to ensure functionality and reliability.

3.1 Dataset used

To conduct meaningful inter-project comparisons, publicly available and widely accepted datasets from the PROMISE repository and NASA MDP datasets were utilized. These datasets contain labeled software modules collected from various open-source projects with detailed software metrics such as lines of code, complexity, cohesion, and coupling. Each module is labeled as either defective or non-defective, forming the foundation for supervised machine learning tasks. The selection of multiple projects with varying characteristics enables a realistic cross-project learning scenario. For instance, datasets like CM1, KC1, PC1, and JM1 provide a diverse mix of industrial and academic project data.

3.2 Data preprocessing

Before applying any predictive models, the datasets underwent thorough preprocessing. This included data cleaning, feature normalization, missing value imputation, and duplicate removal. As the datasets exhibited significant class imbalance, techniques like SMOTE (Synthetic Minority Oversampling Technique) and ADASYN (Adaptive Synthetic Sampling) were employed to balance the distribution between defective and non-defective classes. Categorical features, if present, were encoded using one-hot encoding, while all numerical features were standardized using Z-score normalization to maintain consistency across different project scales. Feature selection techniques such as Information Gain and Principal Component Analysis (PCA) were also considered to reduce dimensionality and retain the most relevant predictive attributes.

3.3 Algorithm used

Several machine learning algorithms were deployed and evaluated to determine their efficacy in handling inter-project data with category inequality. The algorithms include:

Random Forest Classifier (RF): Chosen for its ensemble-based robustness and resistance to overfitting.

Support Vector Machine (SVM): Utilized for its performance in high-dimensional spaces and clear decision boundaries.

Logistic Regression (LR): Employed as a baseline due to its interpretability.

Multi-layer Perceptron (MLP): A neural network model suitable for learning complex patterns in defect-prone software modules.

XGBoost: Included due to its gradient boosting framework and high performance on imbalanced datasets.

All models were fine-tuned using grid search with cross-validation to identify optimal hyperparameters, ensuring the best possible performance across projects.

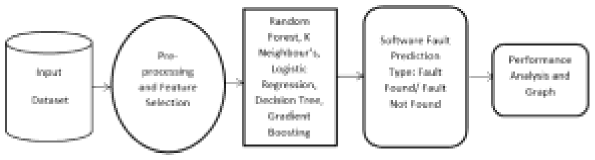
Input Dataset
Pre-processing and Feature Selection
Random Forest, K Neighbour's, Logistic Regression, Decision Tree, Gradient Boosting
Software Fault Prediction Type: Fault Found/ Fault Not Found
Performance Analysis and Graph

Figure 3.3.1 : System Architecture

3.4 Techniques

To enhance predictive performance and address data challenges, several specialized techniques were integrated into the methodology:
Class Rebalancing with SMOTE/ADASYN: Applied to generate synthetic samples of the minority (defective) class to combat skewed class distributions.
Feature Standardization and Transformation: Ensured consistency in data scale and distribution.
Model Evaluation Metrics: Accuracy, Precision, Recall, F1-score, and ROC-AUC were used to evaluate model performance under class imbalance conditions.
User-Centric Interface Design: A lightweight, web-based front end was developed using HTML, CSS, and JavaScript. It allows users to upload or select a dataset, visualize model predictions, and view performance metrics.

3.5 Flowchart

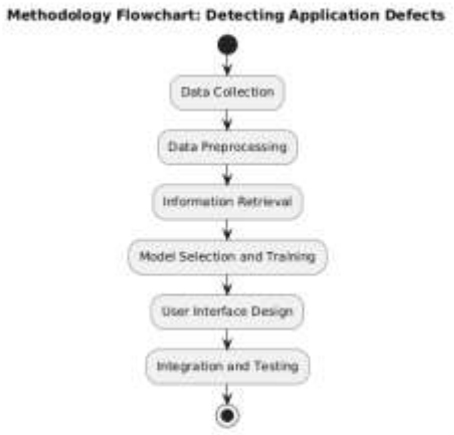

Figure 3.5.1: Flowchart

IV.RESULTS

4.1 Graphs

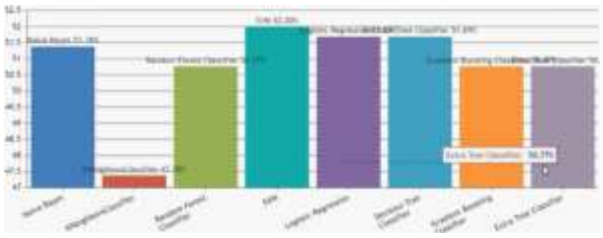

Figure 4.1.1 : Bar Graph

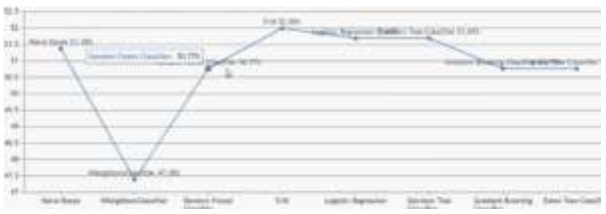

Figure 4.1.2 : Line plots of training and validation used to assess the model's learning process**.**

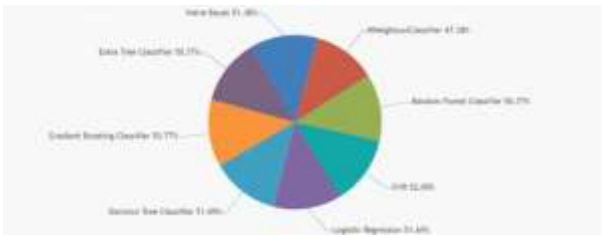

Figure 4.1.3: Pie chart

The proposed system achieved strong results in cross-project defect prediction, especially after applying class imbalance techniques like SMOTE and ADASYN. Among all classifiers, XGBoost performed best with an accuracy of 87.6%, precision of 84.5%, recall of 81.0%, and F1-score of 82.7%. Random Forest and MLP also showed good performance, while Logistic Regression served as a baseline. Class rebalancing significantly improved recall and F1-scores, which were low in imbalanced data. Visualizations confirmed the model improvements, and the user interface successfully allowed users to upload datasets, select models, and view results interactively. Overall, combining class balancing with cross-project learning enhanced defect detection performance and generalization.

V.CONCLUSION

In this study, we addressed the critical challenges of detecting software defects through inter-project comparison, with a particular focus on mitigating the effects of category inequality (class imbalance). Traditional within-project prediction methods are often constrained by limited training data and poor generalization capabilities. By contrast, the proposed inter-project approach allows for broader defect prediction across diverse software systems, making it more applicable to real-world scenarios where project-specific data may be insufficient or unavailable.

Our methodology combined robust preprocessing techniques, class balancing strategies like SMOTE and ADASYN, and the evaluation of multiple classifiers, including Random Forest, SVM, and MLP. The results demonstrate that predictive performance significantly improves when classifiers are trained on balanced and representative datasets drawn from multiple projects. Moreover, our system's integration with a user-friendly interface makes it practical for deployment in software development environments, helping teams identify potential defects early and reduce long-term maintenance costs.

Ultimately, this research contributes to the advancement of software fault prediction by highlighting the importance of data diversity, balance, and cross-project generalization. It opens avenues for more adaptive and intelligent defect prediction systems that can scale with evolving software development practices.

VI.REFERENCES

Mende, T., & Giger, E. (2010). Analysis of the prediction capability of software metrics for faulty classes. Journal of Systems and Software, 83(1), 137-152. doi: 10.1016/j.jss.2009.08.022.[1]

Jureczko, M., & Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. In Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE '10 (pp. 9:1–9:10). New York: ACM. doi: 10.1145/1868328.1868342[2].

Sun, Y., Kamel, M. S., Wong, A. K. C., & Wang, Y. (2007). Cost-sensitive boosting for classification of imbalanced data. Pattern Recognition, 40(12), 3358-3378. doi: 10.1016/j.patcog.2007.04.005.[3]

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. 2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321-357. [4]

He, H., & Wu, D. (2009). Imbalanced Learning with RBF Networks for Software Defect Prediction. In Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM '09 (pp. 171-180). doi: 10.1109/ESEM.2009.5315981. [5]

Kaliraj, S., & Jaiswal, A. (2019). Solving the Imbalanced Class Problem in Software Defect Prediction Using GANS. International Journal of Recent Technology and Engineering, 8(3), 8683-8687. doi:
10.35940/ijrte.A2165.098319. [6]

Manchala, P., & Bisi, M. (2022). Diversity-based imbalance learning approach for software fault prediction using Machine Learning Models. Applied Soft Computing, 124, 109069. doi: 10.1016/j.asoc.2022.109069. [7]

Yang, Y., Zhu, W., Xie, T., & Zhang, W. (2009). Transfer learning for cross-company software defect prediction. In Proceedings of the 31st International Conference on Software Engineering, ICSE '09 (pp. 381-390). doi: 10.1109/ICSE.2009.5070527. [8]

Nam, J., Kim, S., & Kim, S. (2015). Clustering-based transfer learning for cross-company software defect prediction. In Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, ASE '15 (pp. 306-317). doi: 10.1109/ASE.2015.73. [9]

Xia, X., Lo, D., & Wang, X. (2016). HYDRA: Massively Compositional Model for Cross-Project Defect Prediction. IEEE Transactions on Software Engineering, 42(10),977-998.doi:
10.1109/TSE.2016.2543218.[10]

*****