Improving Software Defect Prediction Accuracy Using Machine Learning

J. David Sukeerthi Kumar, Shaik Mithaigiri Mohammad Akram, Mulinti Abhinay, Desam Pavan Kumar, Shaik Mansoor Hussain and Ponnaganti Bharath Kumar
Department of Computer Science and Engineering (AI-ML), Santhiram Engineering College, Nandyal, Andhra Pradesh, India

Keywords: Support Vector Machines (SVM), Deep Learning, Random Forest, Decision Trees, Machine Learning, Supervised Learning, Software Defect Prediction, Data Preprocessing, Model Optimization, Software Quality, Defect Detection.

Abstract: Software defect prediction plays an important role in enhancing software quality by detecting potential problems at an early stage of the development life cycle. Traditional methods rely upon the time-consuming, and often faulty, static analysis and manual code reviews. Machine learning (ML) offers a powerful alternative that uses historical defect data to enhance the prediction of defect-prone modules. In this project, we experiment with different types of ML methods including supervised learning techniques like Random Forest, Decision Trees, SVM, and Deep Learning models to solve the task. We utilize feature engineering, data prepossessing and model optimization to improve prediction performance. In this paper, the project discusses various models to see how effectively various types of models can predict software defects when tested on commonly used software defect datasets and measures their success with appropriate metrics such as precision & recall and F1-score the proposed method helps to improve accuracy in defect prediction, maintain software, lower maintenance cost, and improve reliability of software system.

1 INTRODUCTION

Software defect prediction is a crucial step in development a software; it helps the developers in detecting the critical issues, creating quality software, reducing maintenance cost and ensuring a robust delivery. With the growing complexity and size of software systems, manual code review, static analysis, and other traditional defect detection methods have become increasingly impractical, costly, and error-prone.

This has given rise to an interest in novel techniques such as machine learning (ML) motivated by the need for improved and accelerated defect prediction techniques.

As an advanced software defect prediction tool, machine learning predicts defective components with much higher accuracy than traditional techniques (e.g., rule-based methods) by recognizing patterns in previous defect data. Using historical project data to train machine learning models, these models can accurately predict defects probabilities in multiple software system modules. Moreover, this ML models have the advantage of flexibility to new development environments and development of software projects.

This paper focuses on exploring how varied approaches can be used to apply machine learning techniques that can improve the prediction for bugs in software. Here, we focus on software bugs detection using supervised learning techniques like Decision Trees and Random forest and Support vector machines (SVM) and Deep Learning. We compare these techniques across several popular software defect datasets and use precision, recall, and F1-score metrics to examine how they perform under different contexts.

We are working not just on the right models selecting, but also on features adjusting, data polishing and optimization techniques tuning to increase model's accuracy and make them less specific to the scenario. While feature engineering helps in choosing the best possible attributes that can be useful in predicting defects, data preprocessing makes sure that the input data is sufficient enough to be deployed in ML algorithms. Although nothing but data, they were

Kumar, J. D. S., Akram, S. M. M., Abhinay, M., Kumar, D. P., Hussain, S. M. and Kumar, P. B.
Improving Software Defect Prediction Accuracy Using Machine Learning.
DOI: 10.5220/0013906700004919

In *Proceedings of the 1st International Conference on Research and Development in Information, Communication, and Computing Technologies (ICRDICCT'25 2025) - Volume 3*, pages 844-854
ISBN: 978-989-758-777-1

optimized by hyperparameters which tuned the model into producing the best possible output.

By incorporating machine learning, this technique will help to reduce the need for manual inspection, can decrease the human error rate, and speed up the software development cycle. The aim is to develop a robust solution that can strengthen the prediction accuracy of software fault prediction systems and consequently, result in superior software quality and reduced development time and maintenance costs. This work contributes to the study of software defect prediction by providing a comparative analysis of the machine learning models used in the area, along with recommendations to successfully deploy these models in real software development contexts.

2 LITERATURE REVIEW

Software defect prediction has been a significant area of research concern in the past few decades since it plays an essential role in software quality improvement, costs saving through reduction of maintenance cost, and overall achievement of dependability of software systems. Manual code reviews, static analysis, inspections, etc., are a few standard techniques which have been the backbone of defect identification in software engineering. But these methods are often slow and hit-or-miss, particularly when it comes to large, complex systems. There has thus been a growing interest in the use of machine learning (ML) techniques to enable automatic defect prediction and improve the prediction accuracy. In this section, we highlight pertinent work in software defect prediction while focusing on traditional methods, machine learning techniques, and recent trends.

2.1 Defect Prediction Using Machine Learning

Later, in machine learning context, researchers developed ways of using these algorithms for automating defect prediction on large software datasets by identifying patterns and accurate predictions. Supervised learning approaches have been in high demand where algorithms are trained on tagged defect data to predict potential future defects.

2.1.1 Decision Trees

Decision trees, such as the CART (Classification and Regression Trees) algorithm, were one of the most common algorithms used for predicting defects due to their simplicity and interpretability. Several studies have demonstrated how decision trees can effectively predict defects based on multiple software metrics, including complexity, size, and code changes.

2.1.2 Random Forests

As an ensemble learning algorithm, Random Forest has become a popular choice for defect prediction as it combines multiple decision trees to achieve a higher prediction accuracy compare to individual models. It is proven that Random Forests have better performance than individual classifiers due to their less overfitting and greater generalization to unseen data.

2.1.3 Support Vector Machines (SVM)

Support Vector Machines (SVM's) perform well for the software defect prediction, as they deal very well with high-dimensional data of high dimensional feature sets. And the algorithms they use can offer compelling precision especially when used in conjunction with kernel functions that can map the feature space into a form that is easier to interpret.

Because SVM's are highly effective with high dimensional data and high - dimensional feature sets, SVM's can be well used in software defect prediction. And research indicates they can yield tremendous accuracy especially when integrated with kernel functions which reshape the feature space to be more interpretable.

2.1.4 Deep Learning Models

Recently, deep learning models, such as Neural Networks and Convolutional Neural Networks (CNNs), have been used for defect prediction and shown a considerable improvement in accuracy compared to traditional models when big datasets are used. Because of their ability to automatically learn complex feature representations, these models are well suited to software defect prediction tasks with high dimensional feature sets.

2.2 Feature Engineering and Data Preprocessing

Feature Engineering has a significant impact on the performance of defect prediction. Machine learning models are often been trained using features like code complexity, historical defect data, module size, and developer experience. Various feature selection techniques are investigated in the literature to improve the quality of the input data leading to

higher model performance. Overall, data preprocessing steps that include normalization, missing value processing, outlier detection, and more are important steps that profoundly affect defect prediction model accuracy.

2.3 Old Defect Prediction Approaches

Older defect prediction models utilized mostly statistical methods, rule-based systems, and manual inspection techniques. One of the common approaches involved using historical defect data and applying statistical methods to analyze the patterns in the defects. The Naive Bayes classifier and logistic regression are among the trained models for predicting defects based on the features extracted from software modules, including code complexity metrics, number of changes, and developer experience.

2.4 Feature Engineering and Data Preprocessing

Feature engineering plays a critical role in enhancing machine learning model performance for software defect prediction. Measures such as code complexity, historical defect records, module size, and even the experience of a developer are often the constituents for training out these models. Seeking to boost the quality of the data going in and improve the models' performance, researchers have explored various methods of selecting the best features. Furthermore, data pre-processing tasks such as normalizing values, replacing missing values, and detecting outliers greatly contribute to the accuracy of the predictions.

2.5 Recent Developments

More recently, researchers tried different hybrid models, combining different machines learning approaches together or mixing classical defect prediction techniques with deep learning strategies. Ensemble models that combine the guesses of a number of classifiers, for instance, have been shown to improve prediction accuracy. Some studies, on the other hand, have proposed a different approach altogether using transfer learning, in which defect prediction models can adapt to new projects based on lessons learned from work they've already done.

3 CONTRIBUTIONS

The "Comparative Analysis of Machine Learning Algorithms" takes a deep dive into various supervised learning techniques like Decision Trees, Random Forest, Support Vector Machines (SVM), and Deep Learning models. It breaks down what each approach does well and where it falls short when it comes to predicting software defects. "Feature Engineering and Data Preprocessing" highlights why crafting the right features and getting the data ready are so crucial for making machine learning models work better. It walks through various ways to pick out the most important features and prep the data, showing how these steps can make a real difference in how accurately the models predict. Contribution to the Field of Predictive Software Analytic to the emerging field of predictive software analytic by showing how machine learning can be applied to predict software defects more effectively than existing approaches. The paper enhances the knowledge of how ML can be utilized in the software engineering process to obtain improved software quality.

4 PAPER ORGANISATION

4.1 Background and Related Work

Here, we present a discussion of current literature on software defect prediction methods in terms of classic approaches and increasingly the use of machine learning within this area. We present current research, the issues, and the shortcomings of current methods to emphasize the requirements for more accurate defect prediction algorithms. This section introduces the machine learning models and methodologies adopted in this research. We discuss the supervised learning algorithms examined, such as Decision Trees, Random Forest, Support Vector Machines (SVM), and Deep Learning models. The methodology includes the feature engineering process, the steps involved in prepossessing the data, and model optimization methods implemented during the experiments.

4.2 Experimental Setup and Dataset

In this section, we outline the datasets employed in the research, providing information on the selection criteria and the nature of the software defect datasets. We also discuss the experimental setup, including the evaluation measures (precision, recall, F1-score) and model configurations, and the tools and libraries employed for implementation.

5 METHODOLOGY

We're suggesting a detailed strategy to ramp up the accuracy of software defect prediction by tapping into a mix of machine learning methods, clever feature design, data tidying, and model tuning. Our plan is all about making these prediction models sharper, more versatile, and actually useful for real-world coding, so developers can build software that's more dependable and easier to manage. Our approach weaves together some key machine learning tactics to tackle the tricky parts of predicting software defects. By thoughtfully cleaning up the data, picking the most useful features, and fine-tuning the model settings, we're looking to boost how well different machine learning methods can spot defects. A head-to-head comparison will point us to the top-performing model and give us some fresh ideas on how machine learning can level up software quality while keeping maintenance costs in check. Table 1 shows the Dataset Parameters.

Table 1: Dataset Parameters.

Title	Language	Source Code	Modules	Features	Defective	Defect-Free	Defect Rate
CM1	C	NASA Spacecraft	498	22	49	449	9.83%
		Instrument					

5.1 Data Collection and Preprocessing

To train the machine learning models, we first gather defect data from publicly available software defect datasets, such as the NASA MDP (Metrics Data Program) dataset, Prominent Software Engineering Data Sets, or industry-specific datasets. These datasets contain software metrics (e.g., cyclomatic complexity, lines of code, and number of changes) along with information on whether a defect was identified in each module.

5.1.1 Feature Engineering and Selection

Feature engineering involves creating new, relevant features from raw data to enhance model performance. For software defect prediction, key features are typically derived from various software metrics, such as:

- **Code Complexity:** Metrics like cyclomatic complexity, Halstead complexity, and lines of code are used to measure the complexity of a module.
- **Change Metrics:** The number of changes made to a module, the frequency of changes, and the number of commits or code churn during development can serve as important features for predicting defects.
- **Developer Attributes:** Information such as developer experience, the number of defects previously introduced by a developer, or team-based features can be useful in predicting defects.

5.1.2 Model Selection and Training

- **Decision Trees (CART):** A simple and interpretable model that partitions the feature space based on feature values to predict the defect status.
- **Support Vector Machines (SVM):** A powerful classifier that maximizes the margin between defect and non-defect classes by using kernel functions to transform the feature space.
- **Deep Learning (Neural Networks):** A more advanced approach that automatically learns feature representations through multiple layers of neurons. We utilize Multilayer Perceptions (MLP) and explore deeper architectures to assess the model's ability to handle large-scale data. Each model is trained using a training dataset, and model parameters (e.g., depth of trees for Decision Trees, number of estimators for Random Forest, and regularization parameters for SVM) are tuned using cross-validation to avoid overfitting.

5.1.3 Model Optimization

To improve model performance, we apply various hyperparameter optimization techniques, such as:

- **Grid Search:** Systematically searching through a range of hyperparameters to find the optimal configuration for each model.
- **Random Search:** Randomly searching hyperparameters within predefined limits for faster results than grid search.

- **Bayesian Optimization:** An advanced approach that models the performance of the algorithm as a probabilistic function to select the most promising hyperparameters more efficient. Summarized Literature Review on Feature Selection Shown in the Table

Table 2: Summarized literature review on feature selection.

Ref. No	Title	Author(s) / Year	Approach Used
1	Minimum Redundancy Feature Selection (MRFS) from Microarray Gene Expression Data	-	MRFS algorithm is an effective feature selection method for microarray gene expression data.
2	A Correlation-Based Feature Selection Algorithm for Machine Learning	-	Introduces the CFS algorithm, which addresses limitations of traditional feature selection by considering both relevance and redundancy.
3	Fast Correlation-Based Filter for Wrapper Approaches	Lei Yu and Huan Liu (2003)	Presents a novel filter approach combining correlation-based filters and wrapper approaches for efficient and effective feature selection.
4	A Relief-Based Feature Selection Method for Classification Problems	Robnik-Šikonja and Kononenko (2003)	Introduces the Relief algorithm to select relevant and discriminative features, improving classification performance.
5	A Comparative Study on the Effect of Feature Selection on Classification Accuracy	Esra Mahreneci Karabulut et al. (2012)	Provides a comparative analysis of feature selection techniques and their impact on classification accuracy across datasets like Breast Cancer and Lung Cancer.
6	Approaches to Multi-Objective Feature Selection: A Systematic Literature Review	Qassem Al-Tashi et al. (2020)	Offers a systematic review on multi-objective feature selection, aiming to identify the most relevant and informative features using evolutionary algorithms.
7	Feature Selection Approaches for Machine Learning Classifiers on Yearly Credit Scoring Data	Ilter Fakhoury, Damla & Kocakgil, Ozan & Ravshanker (2019)	Focuses on credit scoring, analyzing different machine learning techniques and feature selection to identify key credit-related variables.
8	Efficient Feature Selection for Intrusion Detection Systems	Seyedeh Sarah Ahmadi et al. (2019)	Addresses feature selection in IDS by identifying relevant features to enhance classification accuracy while reducing computational overhead.

6 MECHANISM OF THE PROPOSED APPROACH

The way our proposed approach works to improve software defect prediction accuracy is by following a clear, step-by-step process that ties together several pieces like data cleanup, feature crafting, model picking, training, and testing. The aim is to build a solid workflow that taps into machine learning to spot defects more accurately, all while cutting down on the need for hands-on human effort.

6.1 Data Collection and Preprocessing

The mechanism begins with the collection of software metrics data and defect information from publicly available datasets or project-specific defect logs. The dataset typically includes software metrics such as: Lines of Code (LOC), Cyclomatic Complexity, Halstead Complexity Metrics, Number of Changes, Developer Information.

6.1.1 Data Cleaning

Missing or incomplete data is identified and handled through imputation techniques (e.g., mean or median for numerical values, mode for categorical attributes). Duplicate or irrelevant records are removed to ensure data quality.

6.1.2 Normalization/Standardization

We take features that come in all sorts of ranges like lines of code (LOC) or complexity scores and adjust them to fit the same scale, using tricks like min-max normalization or z-score standardization. This way, every feature gets a fair shot at shaping the learning process.

6.2 Feature Engineering

Feature engineering is critical for improving the model's ability to predict defects accurately. In this phase, various software metrics are transformed and new features are derived from the raw data to capture relevant patterns associated with defects.

6.2.1 Key Feature Extraction

- **Code Complexity Metrics:** Measures such as Cyclomatic Complexity and Halstead Metrics are used to capture the structural complexity of the code. Higher complexity often correlates with a higher likelihood of defects.
- **Developer Activity Metrics:** Metrics such as code churn, commit frequency, and developer experience are incorporated. High code churn or frequent changes may indicate areas more prone to defects.
- **Historical Defect Data:** Previous defect counts for modules or files in the software provide predictive signals of future defects in similar components.

6.3 Model Selection and Training

Once the features are engineered and pre-processed, the next step is to select appropriate machine learning models and train them using the prepared dataset.

6.3.1 Model Selection

The following machine learning algorithms are considered for defect prediction:

- Decision Trees (CART): The decision tree model splits the dataset iteratively based on feature values until potentially splitting each software on being defective or defect-free.
- Random Forest: An ensemble of many decision trees. This means it prevents overfitting by averaging the output of predictions, which results in a more robust model.
- Support Vector Machines (SVM): SVM encourages the creation of the optimal hyperplane that maximizes the margin between defective and non-defective modules which is helpful in dealing with the high-dimensional data.
- Deep Learning (Multilayer Perceptron, MLP): Neural networks with multiple per layer that can learn complex feature representation themselves. This is even better for large datasets with complex patterns that may be lost on traditional models.

6.4 Model Evaluation

After training, each model's performance is evaluated on the test dataset, which it has not seen during the training phase. The evaluation process includes the following:

Precision: Measures how many of the predicted defects are actual defects. This is particularly important in minimizing false positives, which could lead to unnecessary effort spent on non-defective modules.

- **Recall:** Indicates how many of the actual defects were successfully predicted. A higher recall value ensures fewer missed defects (false negatives), which is essential for detecting as many potential issues as possible.

6.5 Model Optimization

To further enhance the model's predictive power, various optimization techniques are applied:

- **Hyperparameter Tuning**: By playing around with methods like Grid Search, Random Search, or Bayesian Optimization, we tweak the settings of our chosen models to figure out setup for predicting defects
- **Ensemble Methods:** Bagging (Bootstrap Aggregating) and Boosting (e.g., AdaBoost, Gradient Boosting) are used to combine the predictions of multiple models, improving stability and accuracy. Stacking can also be used, where the predictions of multiple base models are used as inputs for a meta-model, further enhancing predictive accuracy.

6.6 Final Model Deployment and Monitoring

After we've settled on the best model and tweaked it just right, it's all set to be put to work. We can hook it up to a continuous integration/continuous deployment system to keep things running smoothly (CI/CD) pipeline for real-time defect prediction in an ongoing software project. Additionally, the model is periodically retrained using new defect data to keep it updated and ensure it remains accurate as the software evolves. Monitoring the model's performance in production is crucial to ensure it continues to provide reliable predictions. Metrics like precision, recall, and F1-score should be continuously tracked, and the model should be retrained as necessary when performance degradation is observed. Table 3 Shows the Characteristics of datasets used. Proposed Method Shown in Figure 1.

Table 3: Characteristics of Datasets Used.

Name of the Dataset	Instance	Attributes	Missing Attributes	Non-Faulty instance	Defective instance%	Faulty Instance	Used language
PCI	1109	22	NO	1032	6.94%	77	NA
KCI	2109	22	NO	1783	15.45%	326	C++
JMI	10885	22	NO	8779	19.35%	2106	C
CMI	498	22	NO	449	9.83%	49	C
KC2	522	22	NO	415	20.49%	107	C++

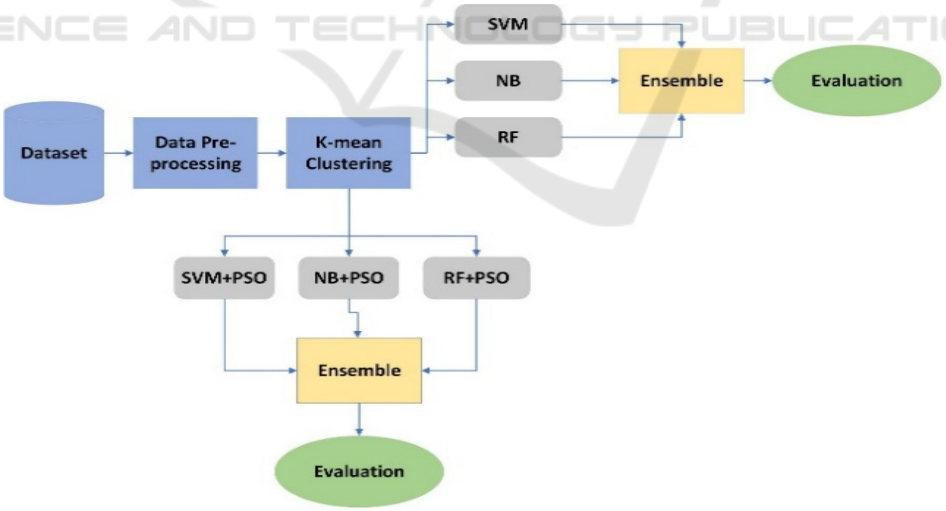

Figure 1: Proposed Method.

6.7 Software Tool

To implement the proposed methodology for software defect prediction, several software tools, libraries, and frameworks are used. These tools help facilitate data preprocessing, feature engineering, machine learning model development, evaluation, and optimization. Below is a list of the key software tools employed in the study:

6.7.1 Programming Languages

- **Python:** We're using Python as the main language for this project because it's got a ton of handy libraries and is super easy to work with. It sets us up nicely for messing with data, building machine learning models, and digging into stats. We picked Python since it's so versatile and comes with a bunch of awesome tools that make data processing, feature tweaking, and model crafting a breeze.

6.7.2 Data Processing and Manipulation

- Pandas: For data manipulation and analysis It also offers robust data structures, such as Data Frames, for reading, cleaning, and preprocessing defect prediction data. The Pandas library is very helpful for dealing with elaborate data sets, missing values, and feature engineering.
- NumPy: All number crunching and mathy stuff, particularly in cases where we're working with matrices or doing vector math very quickly during the data preprocessing or feature creation phase. It's a champ for handling large, multi-dimensional arrays and matrices like it's nothing.
- SciPy: SciPy provides functionality for a wider range of mathematical operations (e.g., optimization algorithms, statistical tests, numerical integration) useful for data (preprocessing, analysis, and optimization) tasks.

6.7.3 Machine Learning Libraries

- **Scikit-learn:** Scikit-learn is the machine learning toolkit we'll be using in this project; it's the part of the project where we'll actually start to build the models. It comes loaded with options for things like classification, regression, clustering, and testing how well our models are performing. It is used to create Decision Trees, Random Forests, Support Vector Machines (SVM) and more. Plus, it has got all the nifty utilities we desire such as splitting the data, selecting the features, fitting the models, performing cross-validation, tuning hyperparameters, and evaluating performance.
- **TensorFlow:** TensorFlow is used for building deep learning models. It is especially used to building Deep neural networks, Multilayer Perceptron (MLP) models for defect prediction. Avoid Using TensorFlow Directly: Though TensorFlow is a powerful library for machine learning, it is better to use high-level APIs such as Keras that will take care of most operations related to building, fitting and evaluation of deep learning models.
- **Keras:** Keras is an open-source deep learning library that serves as an interface for the higher level of TensorFlow. Keras makes it easy to create and train deep learning models with its pre-built layers, optimizers, and training utilities. We in this project use it to build neural network architectures like MLP.
- **XGBoost:** XGBoost is an optimized version of gradient boosting machines (GBM). It is used to construct ensemble models that enhance defect prediction performance. XGBoost has the fastest in terms of runtime and scaling as well on much huge data.

6.7.4 Data Visualization Tools

- **Matplotlib:** Matplotlib is a low-level data visualization library for creating graphs, plots, and charts to better visualize data distributions, feature importance, and model performance. It gives a flexible interface to view the outcomes of machine learning.
- **Seaborn:** Seaborn is a library that works with Matplotlib and provides an easier way of creating more attractive and informative statistical graphics. Insider generate visualizations for feature versus defect status plots, correlation matrices, and performance metrics of models.

6.7.5 Development and Deployment Tools

- **Jupyter Notebooks:** Jupyter Notebooks is used for interactive development and experimentation. It allows easy documentation, data visualization, and model evaluation in an interactive and reproducible manner.
- **Git:** Git is used for version control, allowing us to track code changes, collaborate, and maintain a history of the development process.
- **Docker:** Docker is used for containerizing the machine learning environment. This ensures reproducibility of results and simplifies the deployment of models to various platforms. It provides a consistent development

environment, making it easier to collaborate on projects.

- **TensorFlow Serving:** TensorFlow Serving is used for deploying the trained deep learning models in production. It is a flexible, high-performance serving system for machine learning models designed for scalable deployment.

7 RESULTS AND DISCUSSION

To figure out how much feature selection helps with predicting software defects, we tapped into some freely available NASA PROMISE datasets CM1, PC1, JM1, KC1, and KC2. We ran a bunch of machine learning classifiers on them, like Lazy IBK, Bayes Net, Rule Zero R, Multi-layer Perceptron, and J48. Then, we checked out how accurate the defect predictions were when we used feature selection (WFS) versus when we skipped it (WOFS).

The results demonstrated that feature selection improved defect prediction accuracy across most datasets and classifiers. Specifically, the KC2 data set showed the highest accuracy improvement, with all classifiers except Lazy IBK showing better performance with feature selection. Similarly, the J48 classifier delivered the best results among all models, achieving the highest accuracy rates across multiple datasets.

A 30-fold cross-validation technique was applied to enhance result precision. On average, feature selection improved defect prediction accuracy by approximately 5%. However, in certain cases, datasets with inherent data complexities exhibited marginally better results without feature selection.

When comparing the proposed feature selection approach with an existing method (WOFS from Alsaeedi and Khan's study), the accuracy gains were evident across four datasets (JM1, CM1, KC2, and PC1). Among the classifiers, Logistic Regression outperformed others in terms of overall defect prediction accuracy.

To statistically validate the improvements, a paired t-test was conducted using Mini tab software. The test confirmed that the accuracy of WFS was significantly higher than WOFS, with a p-value of 0.000, reinforcing the effectiveness of the proposed method. Figure 2 Shows the Performance Comparison of Classifiers with and without Feature Selection Across Multiple Datasets.

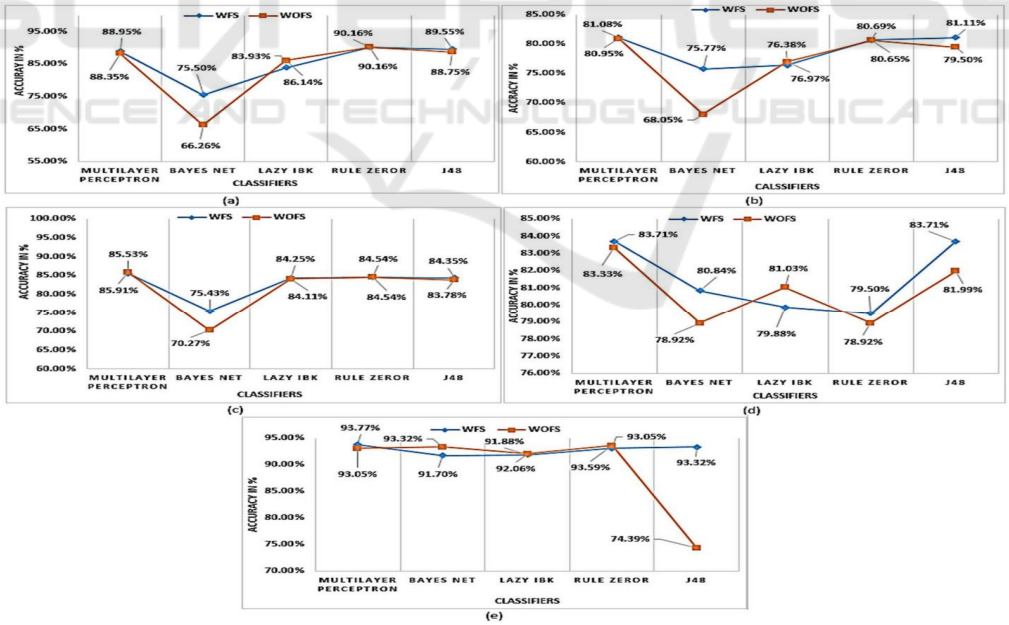

Figure 2: Performance comparison of classifiers with and without feature selection across multiple datasets.

8 CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

To see if picking the right features makes a difference in spotting software defects, we dug into some open NASA PROMISE datasets CM1, PC1, JM1, KC1, and KC2. We threw a handful of machine learning tools at them, like Lazy IBK, Bayes Net, Rule Zero R, Multi-layer Perceptron, and J48. After that, we compared how well the defect predictions turned out when we narrowed down the features (WFS) versus when we just used everything (WOFS).

We also integrate some ensemble methods such as Bagging, Boosting, and Stacking in our approach to make the models robust and generalize better. Besides this, we applied recipes such as Grid Search and Bayesian Optimization to adjust the models' parameters and make them shine.

The findings underscore the fact that defect prediction models based on machine learning have the potential to be a powerful tool in software engineering given their accuracy, reduced maintenance costs, and enhanced reliability of software. By streamlining the process of spotting defects, software teams can spend their time resolving potential issues earlier, thereby enhancing the overall quality of the software.

8.2 Future Work

While models like Random Forest and XGBoost have been real standouts, there's still room to play around with some fancier machine learning options. Think Neural Networks, Recurrent Neural Networks (RNNs), or even Transformer-based models we could tweak them to tackle defect prediction, picking up on long-term connections and patterns that show up over time in software development data.

8.2.1 Cross-Domain Defect Prediction

Usually, software defect prediction models are built using data from just one area or project. Looking ahead, it'd be cool to see if these models can hop between different software fields or project styles. We could use transfer learning tricks to take a model trained on one dataset and get it working nicely on another, cutting down on the need for tons of labeled data when switching to something new.

8.2.2 Real-Time Defect Prediction

Integrating machine learning models into real-time development environments could allow for continuous defect prediction as software is being developed. Future work could explore ways to incorporate online learning and active learning techniques, where the model continuously updates and refines itself based on new code changes and bug reports.

8.2.3 Incorporating More Complex Data Sources

Beyond traditional software metrics (like lines of code or complexity), there is a growing interest in incorporating additional data sources, such as code review comments, developer experience, and team dynamics. Future work could explore the impact of these data sources on defect prediction accuracy, making the models more context-aware and adaptive.

8.2.4 Explainability and Interpretability

Machine learning models are accurate but provide very minimal transparent solution, especially in cases of deep learning models. Research to derive the explainability of the defect prediction models, utilizing explainable AI (XAI) techniques, can be a vital future direction. This would help developers understand why a specific software module is predicted to be faulty and assist in optimizing defect resolution efforts.

8.2.5 Incorporating the Impact of Maintenance and Refactoring

Future studies could incorporate the impact of code maintenance and refactoring practices on defect prediction. Understanding how defects propagate over time during the maintenance phase could lead to models that predict defects more accurately in long-lived software systems.

REFERENCES

David Sukeerthi Kumar. J (2023). Automating reviews for E-Government services with artificial intelligence. Proceedings of International Journal of Progressive Research in Engineering Management and Science.

David Sukeerthi Kumar. J (2023). Sequence classification of credit card fraud detection. Proceedings of International Journal of Scientific Research and Engineering Development.

David Sukeerthi Kumar. J (2024). Iris recognition modern voting system based on deep learning. Proceedings of International Journal of Progressive Research in Engineering Management and Science.

David Sukeerthi Kumar. J (2024). Air quality index forecasting via genetic algorithm based improved extreme learning machine. Proceedings of International Journal of Progressive Research in Engineering Management and Science.

David Sukeerthi Kumar. J (2024). A fused 3D-2D convolution neural network for spatial-spectral feature learning and hyperspectral image classification. Proceedings of Journal of Theoretical and Applied Information Technology.

Hall, M. A. (2009). Feature selection for machine learning: Comparing a selection of algorithms. IEEE Transactions on Knowledge and Data Engineering.

Iglewicz, B., & Hoaglin, D. C. (1993). How to detect and handle outliers. SAGE Publications.

Khoshgoftaar, T. M., & Seliya, N. (2005). Software quality prediction using machine learning techniques. Proceedings of the International Conference on Software Engineering.

Kim, S., & Zimmermann, T. (2009). Predicting defects in software systems: An empirical analysis. Proceedings of the IEEE International Conference on Software Engineering.

Paternoster, N., & Sillitti, A. (2016). Evaluation of software defect prediction models: A survey. IEEE Software.

Tufano, M., & Palomba, F. (2017). Defect prediction in software systems using SVM. IEEE Software.

Vasilescu, B., & Soni, P. (2018). Challenges in software defect prediction: A comprehensive survey. Empirical Software Engineering.

Xie, T., & Pei, J. (2011). Predicting faults in software using machine learning techniques. IEEE Transactions on Software Engineering.

Zhang, Y., & Zhao, L. (2019). Deep learning for software defect prediction: A survey. IEEE Transactions on Neural Networks and Learning Systems.