A

Major Project On

**A NOVEL APPROACH TO IMPROVE SOFTWARE DEFECT PREDICTION ACCURACY USING MACHINE LEARNING**

(Submitted in partial fulfillment of the requirements for the award of Degree)

**BACHELOR OF TECHNOLOGY**

In

**COMPUTER SCIENCE AND ENGINEERING**

By

N.NISHANTH REDDY (217R1A0539)

S.NAGARAJU (227R5A0504)  
P.SAINATH (217R1A0545)

Under the Guidance of

**Mr. G PAVAN KUMAR**

(Assistant Professor)



## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CMR TECHNICAL CAMPUS**

**UGC AUTONOMOUS**

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi) Recognized Under Section 2(f) & 12(B) of the UGCAct.1956,

Kandlakoya (V), Medchal Road, Hyderabad-501401.

**April, 2025.**

## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**CERTIFICATE**

This is to certify that the project entitled “**A NOVEL APPROACH TO IMPROVE SOFTWARE DEFECT PREDICTION ACCURACY USING MACHINE LEARNING**” being submitted by **N. Nishanth Reddy (217R1A0539), S. Nagaraju (227R5A0504) & P. Sainath (217R1A0545)** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University Hyderabad, during the year 2024-25.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

#### **Mr. G. Pavan Kumar Dr. Nuthanakanti Bhaskar**

#### **Assistant Professor HOD**

**INTERNAL GUIDE**

**Dr. A. Raji Reddy Signature of External Examiner**

**DIRECTOR**

**Submitted for viva voice Examination held on**

We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project, we take this opportunity to express our profound gratitude and deep regard to our guide **Mr. G. Pavan Kumar,** Associate Professor for his exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help, and guidance given by him shall carry us a long way in the journey of life on which we are about to embark.

We take this opportunity to extend our heartfelt appreciation to the Project Review Committee (PRC) Coordinators—**Dr. K. Maheswari, Dr. J. Narasimharao, Ms. K. Shilpa, and Mr. K. Ranjith Reddy**—for their unwavering support, insightful guidance, and valuable inputs, which played a crucial role in steering this project through its various stages.

Our sincere appreciation also goes to **Dr. Nuthanakanti Bhaskar**, Head, for his encouragement and continuous support in ensuring the successful completion of our project.

We are deeply grateful to **Dr. A. Raji Reddy**, Director, for his cooperation throughout the course of this project. Additionally, we extend our profound gratitude to Sri. **Ch. Gopal Reddy**, Chairman, Smt. **C. Vasantha Latha**, Secretary and Sri. **C. Abhinav Reddy**, Vice-Chairman, for fostering an excellent infrastructure and a conducive learning environment that greatly contributed to our progress.

We also acknowledge and appreciate the guidance and assistance provided by the faculty and staff of **CMR Technical Campus**, whose contributions have been invaluable in bringing this project to fruition.

Lastly, we sincerely thank our families for their unwavering support and encouragement. We also extend our gratitude to the teaching and non-teaching staff of CMR Technical Campus for their guidance and assistance. Their contributions, along with the support of everyone who helped directly or indirectly, have been invaluable in the successful completion of this project.

**N. Nishanth Reddy(217R1A0539)**

**S. Nagaraju (227R5A0504)**

**P. Sainath (217R1A0545)**

**INSTITUTE VISION:**

To Impart quality education in serene atmosphere thus strive for excellence in Technology and Research.

**INSTITUTE MISSION:**

1. To create state of art facilities for effective Teaching- Learning Process.
2. Pursue and Disseminate Knowledge based research to meet the needs of Industry & Society.
3. Infuse Professional, Ethical and Societal values among Learning Community.

**DEPARTMENT VISION:**

To provide quality education and a conducive learning environment in computer engineering that foster critical thinking, creativity, and practical problem-solving skills.

**DEPARTMENT MISSION:**

1. To educate the students in fundamental principles of computing and induce the skills needed to solve practical problems.
2. To provide State-of-the-art computing laboratory facilities to promote industry institute interaction to enhance student’s practical knowledge.
3. To inculcate self-learning abilities, team spirit, and professional ethics among the students to serve society.

In software engineering community, defect prediction is one the active domain. For the software’s success, it is essential to reduce the software engineering and data-mining gap. Software defects prediction forecasts the source code errors before the testing phase. Methods for predicting software defects, such as clustering, statistical methods, mixed algorithms, metrics based on neural networks, black box testing, white box testing and machine learning are frequently used to explore the effect area in software. The main contribution of this research is the use of feature selection for the first time to increase the accuracy of machine learning classifiers in defects pre-diction. The objective of this study is to improve the defects prediction accuracy in five data sets of NASA namely; CM1, JM1, KC2, KC1, and PC1. These NASA data sets are open to public. In this research, the feature selection technique is use with machine-learning techniques; Random Forest, Logistic Regression, Multilayer Perceptron, Bayesian Net, Rule ZeroR, J48, Lazy IBK, Support Vector, Neural Networks, and Decision Stumpo achieve high defect prediction accuracy as compared to without feature selection (WOFS). The research workbench, a machine-learning tool called WEKA (Waikato Environment for Knowledge Analysis), is used to refine da-ta, preprocess data, and apply the mentioned classifiers. To assess statistical analyses, a mini tab statistical tool is used. The results of this study reveals that accuracy of defects prediction with feature selection (WFS) is improve in contrast with the accuracy of WOFS.

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **FIGURE NAME** | **PAGE NO** |
| Figure 3.1 | Project Architecture of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 18 |
| Figure 3.2 | Dataflow diagram of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 21 |
| Figure 5.1 | GUI/Main Interface of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 41 |
| Figure 5.2 | Login page of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 42 |
| Figure 5.3 | Registration Page of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 43 |
| Figure 5.4 | Profile Page of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 44 |

|  |  |  |
| --- | --- | --- |
| Figure 5.5 | Form of Prediction of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 45 |
| Figure 5.6 | Defect status of given data of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 46 |
| Figure 5.7 | Service Provider login page of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 47 |
| Figure 5.8 | Training and Testing data of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 48 |
| Figure 5.9 | Training and Testing data accuracy of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 49 |
| Figure 5.10 | Software Defect Prediction type ratio of A Novel Approach To Improve Software Defect Prediction Accuracy Using Machine Learning | 50 |

|  |  |  |
| --- | --- | --- |
| **TABLE NO** | **TABLE NAME** | **PAGE NO** |
| Table 6.2.1 | Uploading Dataset | 53 |
| Table 6.2.2 | Classification | 53 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ABSTRACT** | |  |  | i |
| **LIST OF FIGURES** | | |  | ii |
| **LIST OF TABLES** | | | | iv |
| **1.** | **INTRODUCTION** | | | 1 |
|  | 1.1 | PROJECT PURPOSE | | 1 |
|  | 1.2 | PROJECT FEATURES | | 2 |
| **2.** | **LITERATURE SURVEY** | | | 5 |
|  | 2.1 | REVIEW OF RELATED WORK | | 10 |
|  | 2.2 | DEFINITION OF PROBLEM STATEMENT | | 12 |
|  | 2.3 | EXISTING SYSTEM | | 12 |
|  | 2.4 | PROPOSED SYSTEM | | 13 |
|  | 2.5 | OBJECTIVES | | 14 |
|  | 2.6 | HARDWARE & SOFTWARE REQUIREMENTS | | 15 |
|  |  | 2.6.1 | HARDWARE REQUIREMENTS | 15 |
|  |  | 2.6.2 | SOFTWARE REQUIREMENTS | 16 |
| **3.** | **SYSTEM ARCHITECTURE & DESIGN** | | | 17 |
|  | 3.1 | PROJECT ARCHITECTURE | | 17 |
|  | 3.2 | DESCRIPTION | | 18 |
|  | 3.3 | DATA FLOW DIAGRAM | | 20 |
| **4.** | **IMPLEMENTATION** | | | 21 |
|  | 4.1 | ALGORITHMS USED | | 21 |
|  | 4.2 | SAMPLE CODE | | 24 |
| **5.** | **RESULTS & DISCUSSION** | | | 39 |
| **6.** | **VALIDATION** | | | 49 |
|  | 6.1 | INTRODUCTION | | 49 |
|  | 6.2 | TEST CASES | | 50 |
|  |  | 6.2.1 | UPLOADING DATASET | 50 |
|  |  | 6.2.2 | CLASSIFICATION | 50 |
| **7.** | **CONCLUSION & FUTURE ASPECTS** | | | 51 |
|  | 7.1 | PROJECT CONCLUSION | | 51 |
|  | 7.2 | FUTURE ASPECTS | | 52 |
| **8.** | **BIBLIOGRAPHY** | | | 53 |
|  | 8.1 | REFERENCES | | 53 |
|  | 8.2 | GITHUB LINK | | 54 |

1. **INTRODUCTION**

**1. INTRODUCTION**

### 

In a software system, unexpected performance in response to a client’s need is known as a defect. Software testers typically notice this unusual behaviour in software. Software testers notice errors in the software testing process. The term ‘‘software fault’’ is also used to describe, ‘‘irregularities in the software development process that frequently result in software failure and fall short of user expectations’’ ensuring accurate detection of inappropriate content. A defect is a lack of imperfection caused by an error, fault, or failure in the software development process or product. According to the paradigm, ‘‘error’’ refers to human behavior that leads to inappropriate out-comes, and ‘‘defect’’ refers to a decision that leads to incorrect outcomes when trying to solve a problem.

The process of predicting software defects involves the detection of defected modules and a variety of testing requirements. It is extremely difficult in software engineering to design a good defect prediction model, which would predict malfunctioning software modules or software defects in earlier phases of the software development life cycle. Re-viewing the source code, doing beta testing, integration testing, system testing, and unit testing are all steps in the traditional process of finding software errors. Therefore, it becomes challenging to carry out these tests as software expands in size, complexity, and size of source code.

### PROJECT PURPOSE

The primary goal of this project is to increase the accuracy of software defect prediction models by leveraging machine learning techniques and feature selection. By improving defect prediction accuracy, software developers can identify potential errors early in the development lifecycle, reducing costs, improving software quality, and enhancing overall reliability. This contributes to efficient testing strategies, better resource allocation, and improved decision-making in software engineering.

### PROJECT FEATURES

This project incorporates several key features to improve the software defect prediction accuracy:

Machine Learning Techniques: Implementation Machine learning techniques play a crucial role in data classification, with various algorithms offering unique approaches for model development. Random Forest leverages an ensemble of decision trees to improve prediction accuracy, while Logistic Regression models the probability of binary outcomes using a linear approach. Multilayer Perceptron, a type of neural network, employs multiple layers to capture complex relationships in data, and Bayesian Net utilizes probabilistic graphical models to represent and reason about uncertain data. Additionally, J48, an implementation of the C4.5 algorithm, builds decision trees, and Lazy IBK applies the k-nearest neighbors algorithm to classify data based on proximity to labeled instances, each contributing distinct strengths to classification tasks.

Feature Selection: Feature selection techniques are crucial in improving the performance and accuracy of classification models by identifying and retaining the most relevant features while removing redundant or irrelevant ones. Methods like mutual information, correlation-based feature selection, and recursive feature elimination help in reducing the dimensionality of the data, thereby making models more efficient and less prone to overfitting. By focusing only on the most informative features, feature selection also speeds up the learning process and enhances model interpretability. Overall, these techniques lead to more accurate, faster, and scalable machine learning models.

NASA Dataset Analysis: The Testing and evaluation using five publicly available NASA datasets allows for a comprehensive analysis of machine learning models in real-world, high-stakes domains such as aerospace and engineering. These datasets typically contain complex, multidimensional data from various spacecraft, satellites, or other space exploration projects, offering a rich environment for model testing. By applying different classification algorithms to these datasets, researchers can assess the accuracy, robustness, and generalizability of their models in predicting anomalies or system behaviors. This type of evaluation is essential for developing reliable models that can be used in critical applications, such as predictive maintenance and fault detection in space missions.

By integrating these features, this machine learning-based system provides a powerful, scalable, and automated solution for software defect prediction, ensuring better accuracy, improved detection of potential issues, and a more reliable software development process.

1. **LITERATURE SURVEY**

* + 1. **LITERATURE SURVEY**

The most desirable study field is defect prediction via machine learning, data metrics, and other methods. Different approaches have provided various models and interpretations. There have been numerous studies published on the analysis of software fault prediction from 1990 to 2022. Size and complexity metrics for defect prediction were developed by Benton and Neil in 1999. Software size and software complexity metrics were discussed in the defect prediction process. Using software metrics, processing high-quality data, multivariate methods, and a critique of existing methods, defect prediction is carried out. According to their calculations, each thousands of lines of code(KLOC) contains about 23 flaws.

For defect prediction, Vanmali et al. used Machine learning (ML) approaches To predict the fault, they used neural networks. Similar comparisons between Neural Network and other approaches were uses, and it was conclude that Neural Network outperformed other methodologies in terms of error detection.

They also discussed the application of various ML techniques. They use the PROMISE dataset and hypothesized that the best indicators of programming are responses to classes, LOC, and the absence of good coding. Additionally, they are engaged in comparative research on ensemble approaches for software best practices.

Biçer et al. investigated the situation in which it is impractical to survey thoroughly every component of complicated frameworks. They examined numerous tactics for their stages and described the qualities of good conformity indicators. They assembled their analyses with regard to static code measures and discovered that these flaw identifiers produce results that are consistent across a wide range of applications, are cost-effective to use, and can be adjusted to the point of interest of current business conditions. They took into account realistic conditions for evaluating programming expenses and agreed that a better evaluation allowed for tenfold greater financial gains. They demonstrate how quality pointers can be found early on in the process of product improvement by utilizing reliable measurements and ML approaches.

To categorize various datasets related to liver patients, Ramana et al. [19] examined a few selected machine learning classification techniques. Using two datasets, the effectiveness of a few machine learning classification algorithms was assessed. The first dataset included records for 751 liver patients from Andhra Pradesh in India with 12 attributes. The University of California, Irvine (UCI) Machine Learning Repository provided the second dataset, which included 345 records with five attributes. With a core i7 processor and 4 GB of RAM, the WEKA data mining open-source machine learning tool or workbench was utilized for the trials. Here, the Naive Bayes classifier, K-Nearest Neighbour Algorithm, Back Propagation Neural Network Algorithm, C4.5, and Support Vector Machines were taken into consideration as machine learning classification techniques. Accuracy, Specificity, Sensitivity, and Precision were the four criteria used by these algorithms to evaluate the outcomes. K-nearest neighbor Algorithm, Backpropagation, and Support Vector Machines provide superior results with all feature set combinations when utilizing a chosen dataset.

In order to predict software defects, Gray et al. created models that took into account the quality of the datasets, which were noisy and contained missing values that might affect the outcomes. The researcher concluded that the influence of quality relies on the dataset while building a model and predicting defects, where data cleansing could be a major factor.

NetAskari and Bardsiri [21] applied artificial neural networks for defect prediction. For effective extension machine learning algorithms, evolutionary approaches were combined with SVM learning for prediction. Machine learning models using NASA Datasets were used to test the support vector approach. They concluded that, when accuracy and precision are combined, SVM learning outperformed other methods in terms of accuracy and precision.

Prasad et al. [22] talked about various ML classification techniques, including supervised (Bayesian Network, Ensemble Method/Random Forests, SVM, Decision Tree), unsupervised (j48, Random Forest, Naive Bayes classifier, k-mean clustering, Hierarchical Clustering), and semi-supervised (Low-density separation, SVM, expectation maximization, and class mass normalization) methods that are the best classification strategy to forecast the defect for high-quality software.

Chandra Yadav et al. presented the defect detection algorithm employing classification, association rule, and clustering approaches. The authors explain the Knowledge Discovery in Database (KDD) process and explain how to uncover potential outcomes by applying patterns and extracting errors or faults. The authors concluded that finding bugs or defects might be possible with little testing equipment.

In Kumar and Shukla, the early detection of defects was accomplished using the fuzzy logic information system. At the function level, the proposed model’s fuzzy inference system uses metrics data and other error data. This model has five input variables that are utilized in one input layer and one output layer to determine if the input layer data points to defects..

Federated learning has also gained attention as a privacy-preserving approach to training deep learning models for content moderation. Instead of transmitting raw video data to centralized servers, federated learning allows models to be trained locally on user devices while sharing only model updates. This decentralized approach ensures data privacy while maintaining high classification performance.

Mandal and Ami examined how software attributes depend on quality, performance, and effectiveness in defect prediction models. Several software qualities were used to predict software module malfunctioning. In defect prediction, if the right attributes are not selected, the model’s performance will suffer. Therefore, it is crucial to choose the right features to create a useful prediction model in order to enhance the efficiency and performance of defect prediction. In order to identify software with flaws or approach the right model, the researcher presented an attribute selection technique.

The research findings demonstrates that the described strategy offered a comparably effective collection of features that improved the performance, quality, and efficiency of the model. In order to improve the outcome of software defect prediction, one ML classifier was use. For further improvement of the suggested approach, researchers hope to incorporate the performance of many ML classifiers in the future.

Hammouriet al. addressed the defects prediction model. In order to forecast defects, supervised machine learning techniques such as Naive Bayes, Artificial Neural Networks, confusion matrices, and decision tree algorithms were apply to various datasets. Three debugging datasets were use in the experiment. The aspects of the experimental outcomes were recall, precision, RMSE measurements, F-measure, and accuracy. In addition, these experimental results demonstrate that machine learning is superior to other approaches, such as the POWM model and AR model, in terms of results and pre-diction model performance. Memonetal. evaluates the methods for predicting software errors and mitigating their effects on the production of high-quality software. They discusses several defect prediction mechanisms (based on pattern, graph mining ASA using Classifier) and prevention mechanisms through defect detection, defect analysis, and its importance to minimize the causes of system failure using the most recent technology. Additionally, they describes the advantages and disadvantages of certain systems for the creation of high-quality products

In a study Li et al. included 2456 experimental findings, 49 articles that satisfied the inclusion criteria which were published from January 2000 to March 2018. Matthew’s Correlation Coefficient (MCC) terms were used to calculate the comparison prediction performance over studies in a consistent manner, with confusion matrices serving as the basis. The researchers, for the effectiveness of unsupervised defect

prediction algorithms, performed a meta-analysis. Recalculating the confusion matrices from the primary experiments to get improved performance in order to compare these results in a more credible manner. Numerous frequently used outfit-learning calculations, like stacking calculations, had been widely used. In this way, the subsequent stage of word inquiry mostly entails attempting to examine the benefits and drawbacks of another ensemble learning, and then utilizing other different base classifier blends in the vote calculation.

Though there was a concerning amount of inadequate reporting, undemanding benchmark datasets, and plainly incorrect experimental outcomes.

Zhongetal. begins by highlighting the importance of time-series event prediction in various domains, such as finance, weather forecasting, and anomaly detection. It also emphasizes the limitations of existing prediction methods and the need for more effective techniques. The proposed method introduces a sequence labeling framework, which models the time-series data as a sequence of discrete events.

The framework consists of three main components: data pre-processing, feature extraction, and sequence labelling. In the data-preprocessing step, the time-series data is pre processed to handle missing values, noise, and other data quality issues. This ensures that the subsequent steps operate on clean and reliable data. The paper presents a new method for time-series event prediction based on sequence

labeling. The approach combines data preprocessing, feature extraction, and sequence labeling techniques to accurately predict events in time-series data. The experimental results highlight the effectiveness and robustness of the proposed method, suggesting its potential for applications in various domains requiring accurate event prediction in time-series data. Further research and exploration of this method could lead to advancements in time-series analysis and prediction tasks.

Marioetal. 2023 introduces a novel approach for updating REM models. By leveraging clustering and Random Forest techniques, the proposed methodology addresses the challenges of updating REM models effectively. The experimental results indicate its superiority over traditional

methods, offering promising avenues for future research and practical applications in the field of monitoring and data analysis. Author’s addresses the challenge of updating REM models effectively, as system dynamics and data distribution may change over time. The proposed methodology aims to enhance the accuracy and adaptability of REM models by incorporating new data while preserving the efficiency of the monitoring process. To achieve this goal, the authors propose a two-step approach. In the first step, the data points are clustered into distinct groups based on their similarity. In the second step, a Random Forest model is trained on each cluster to update the . The implications of this research are significant for various applications that rely on REM models. By improving the accuracy and adaptability of REM models, organizations can enhance their monitoring and decision-making processes, leading to more efficient resource allocation, anomaly detection, and predictive analytics.

### REVIEW OF RELATED WORK

This review discusses previous research and existing methodologies, highlighting their strengths and limitations.

1. Traditional Content Moderation Approaches

Early software defect prediction models relied on rule-based and statistical approaches, which were limited by their reliance on predefined thresholds and domain expertise. These methods involved analyzing static code metrics such as lines of code (LOC), cyclomatic complexity, and code churn to estimate defect-prone modules. While useful in some cases, these methods often lacked adaptability, leading to inconsistent results across different software projects and datasets. Additionally, traditional approaches struggled with large-scale datasets, as they relied heavily on manually defined rules that failed to generalize across different projects.

2. Machine Learning-Based Approaches

The introduction of machine learning in software defect prediction marked a significant improvement in accuracy and efficiency. Researchers explored various classifiers such as Support Vector Machines (SVM), Random Forest, Decision Trees, and Naïve Bayes. These models utilized static code metrics and historical defect data to classify software modules as defective or non-defective. The advantage of machine learning approaches was their ability to learn patterns from data without explicit rule definitions. However, these models heavily relied on feature engineering, where domain experts manually selected the most relevant attributes for training the classifiers. This led to challenges in scalability and adaptability, as different datasets required different feature selection strategies.

3. Feature Selection Techniques in Defect Prediction

Feature selection has emerged as a crucial technique for improving software defect prediction accuracy. Many studies have explored wrapper-based, filter-based, and embedded feature selection methods to identify the most relevant attributes for classification. Wrapper methods use classifier performance as a criterion for feature selection, while filter methods assess statistical properties of individual features. Embedded methods, such as LASSO regression and decision tree-based techniques, integrate feature selection within the learning process. Studies have shown that applying feature selection significantly enhances the performance of defect prediction models by reducing noise, improving model interpretability, and lowering computational costs.

4. Recent Advances: Attention Mechanisms & Transformer-Based Models

Hybrid and ensemble approaches have gained popularity in software defect prediction, as they combine the strengths of multiple models to improve accuracy and robustness. Techniques such as bagging, boosting, and stacking have been employed to aggregate predictions from different classifiers. Recent studies have also explored combining machine learning and deep learning techniques, where deep learning models extract high-level features, and traditional classifiers perform final predictions. This approach addresses the limitations of deep learning models by leveraging interpretable machine learning classifiers for decision-making.

5. Comparison with the Proposed Approach

While existing methods have made significant progress in software defect prediction, challenges remain in terms of accuracy, scalability, and real-time detection. The proposed approach builds upon previous work by integrating advanced feature selection techniques with state-of-the-art machine learning classifiers. By leveraging feature selection, the model eliminates redundant attributes, reducing overfitting and improving classification accuracy. Additionally, the proposed system optimizes classifier performance using hyperparameter tuning and ensemble learning strategies, ensuring better defect detection across diverse datasets.

This review highlights the evolution of content moderation techniques, emphasizing the shift from rule-based to machine learning approaches . The proposed methodology aims to address the limitations of previous research by offering a highly accurate, scalable, and efficient solution for inappropriate content detection in YouTube videos.

### DEFINITION OF PROBLEM STATEMENT

Traditional defect prediction models suffer from challenges such as high false positives, poor generalization across datasets, and computational inefficiencies. Many models do not effectively handle imbalanced datasets, leading to biased predictions. The proposed system aims to overcome these limitations by incorporating feature selection, optimizing classifier performance, and ensuring better defect detection accuracy.

### EXISTING SYSTEM

### **Traditional software defect prediction systems relied on rule-based statistical models and early machine learning classifiers, such as Support Vector Machines and Decision Trees. These systems analyzed static code metrics and defect history to predict potential software issues. However, these methods faced several limitations, including high false positive rates, difficulty in handling large datasets, and low generalization across different software projects. The reliance on handcrafted features made traditional models less adaptable to evolving software development practices, leading to inconsistent accuracy levels across different datasets.**

### Limitations of Existing System

1. Inconsistent Accuracy Across Datasets

Software defect prediction models often show varying accuracy when applied to different datasets. This inconsistency arises due to differences in software project characteristics, programming languages, and data distributions. Some models may perform well on one dataset but fail on another due to feature variations, leading to unreliable predictions.

2. High Computational Complexity

Many machine learning models, especially deep learning approaches, require extensive computational resources for training and inference. The need for large datasets, feature extraction, and hyperparameter tuning increases processing time, making real-time defect prediction challenging for large-scale applications.

3. High False Positives and False Negatives

Traditional models often misclassify software modules, leading to a high rate of false positives (defect-free code flagged as defective) and false negatives (actual defects going undetected). This reduces developer confidence in automated defect prediction systems and leads to unnecessary debugging efforts or undetected critical issues in production.

To avoid all these limitations and make the working more accurately the system needs to be implemented efficiently.

### PROPOSED SYSTEM

The proposed system enhances software defect prediction by integrating feature selection techniques with advanced machine learning classifiers. Unlike traditional models that rely solely on static code metrics, this approach leverages optimized feature selection to remove redundant attributes, ensuring better model efficiency. By employing classifiers such as Random Forest, Neural Networks, and Support Vector Machines (SVM), the system improves prediction accuracy while minimizing false positives. Additionally, ensemble learning techniques such as bagging and boosting further enhance model generalization across diverse datasets. The system is designed to be scalable, handling large datasets effectively while maintaining computational efficiency.

### Advantages of the Proposed System:

The proposed system significantly improves upon the existing approaches by addressing key limitations:

* Improved Accuracy – Feature selection enhances classification accuracy by reducing noise and irrelevant data.
* Scalability – The system is capable of handling large-scale software datasets without significant performance degradation.
* Reduced False Positives – By eliminating redundant features, the system minimizes misclassification rates.
* Automated Preprocessing – The system automates data cleaning, normalization, and transformation, reducing manual effort.
* Enhanced Generalization – The use of multiple classifiers and ensemble learning ensures robustness across different software projects.
* Optimized Computational Efficiency – The integration of feature selection techniques reduces training time and improves model efficiency.

### OBJECTIVES

The primary objective of this project is to enhance software defect prediction accuracy using advanced machine learning techniques. By leveraging feature selection and multiple classifiers, the system aims to improve defect detection efficiency and reliability. The specific objectives of this research include:

* Improving Prediction Accuracy – Enhance the precision of software defect prediction by integrating feature selection techniques to remove redundant and irrelevant data.
* Reducing False Positives and False Negatives – Minimize misclassification errors to ensure more reliable defect predictions.
* Optimizing Computational Efficiency – Develop a scalable model that can efficiently process large datasets without excessive computational costs.
* Utilizing Machine Learning Techniques – Compare different machine learning classifiers such as Random Forest, Logistic Regression, Neural Networks, and SVM to determine the best-performing model.
* Validating Performance on Real Datasets – Test the model on publicly available NASA datasets (CM1, JM1, KC2, KC1, and PC1) to assess its effectiveness in real-world scenarios.
* Providing a Generalized Solution – Ensure that the model can generalize well across different datasets rather than being dataset-dependent.
* Enhancing Software Development Processes – Assist software developers in identifying potential defects early in the development cycle, reducing debugging time and improving software quality.

### HARDWARE & SOFTWARE REQUIREMENTS

* + 1. **HARDWARE REQUIREMENTS:**

Hardware interfaces specifies the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements,

|  |  |  |
| --- | --- | --- |
| * Processor | : | Intel Core i7 |
| * Hard disk | : | 512GB. |
| * RAM | : | 4GB. |

### SOFTWARE REQUIREMENTS:

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software requirements,

* Operating system : Windows 10
* Language : Python
* Back-End : Django
* Front-end : HTML,CSS

1. **SYSTEM ARCHITECTURE & DESIGN**

**3. SYSTEM ARCHITECTURE & DESIGN**

The system architecture follows a structured pipeline starting from data preprocessing, feature selection, model training, and final defect prediction. The architecture is designed to handle large datasets efficiently while ensuring high prediction accuracy. It consists of the following components:

### PROJECT ARCHITECTURE

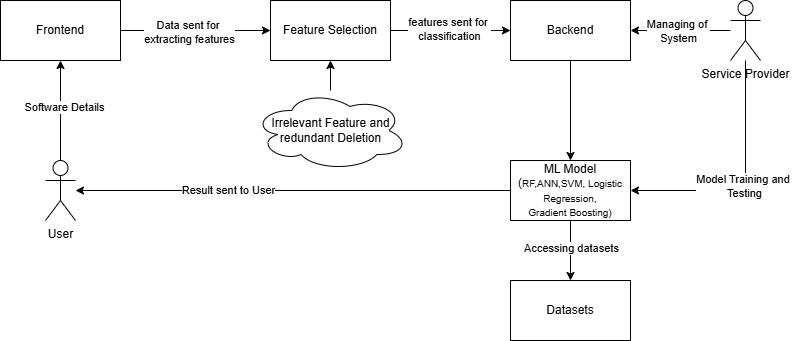


Figure 3.1: Project Architecture of A Novel approach to improve software defect prediction accuracy using machine learning

**3.2 DESCRIPTION**

* **Data Preprocessing:** Cleansing, normalization, and transformation of data for consistency. This step involves handling missing values, removing outliers, and standardizing feature distributions to ensure better model performance. Proper preprocessing enhances data quality and prevents biases in predictions.
* **Feature Selection**: Reducing dimensionality by selecting the most relevant features. Irrelevant or redundant features are removed to improve model efficiency and reduce overfitting. This process ensures that the model focuses on the most important attributes affecting defect prediction.
* **Model Training**: Training machine learning classifiers using the selected features. Various algorithms like Random Forest, SVM, and Neural Networks are trained on historical data to learn patterns. Hyperparameter tuning and cross-validation are applied to optimize the model’s predictive performance.
* **Defect Prediction:** Classifying software modules as defective or non-defective. Once trained, the model analyzes new data and predicts whether a software component contains defects. This prediction helps developers identify potential risks early in the development process.
* **Performance Evaluation:** Assessing model accuracy using statistical metrics. Metrics like accuracy, precision, recall, and F1-score are used to measure effectiveness. Comparative analysis with existing models helps validate improvements in defect prediction accuracy.

### DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a graphical representation that illustrates how data flows within a system, showcasing its processes, data stores, and external entities. It is a vital tool in system analysis and design, helping stakeholders visualize the movement of information, identify inefficiencies, and optimize workflows.

A Data Flow Diagram comprises Four primary elements:

* External Entities: Represent sources or destinations of data outside the system.
* Processes: Indicate transformations or operations performed on data.
* Data Flows: Depict the movement of data between components.
* Data Stores: Represent where data is stored within the system.

These components are represented using standardized symbols, such as circles for processes, arrows for data flows, rectangles for external entities, and open-ended rectangles for data stores.

**Benefits:**

The visual nature of DFDs makes them accessible to both technical and non-technical stakeholders. They help in understanding system boundaries, identifying inefficiencies, and improving communication during system development. Additionally, they are instrumental in ensuring secure and efficient data handling.

**Applications:**

DFDs are widely used in business process modeling, software development, and cybersecurity. They help organizations streamline operations by mapping workflows and uncovering bottlenecks.

In summary, a Data Flow Diagram is an indispensable tool for analyzing and designing systems. Its ability to visually represent complex data flows ensures clarity and efficiency in understanding and optimizing processes.

**Levels of DFD:**

DFDs are structured hierarchically:

* Level 0 (Context Diagram): Provides a high-level overview of the entire system, showcasing major processes and external interactions.
* Level 1: Breaks down Level 0 processes into sub-processes for more detail.
* Level 2+: Offers deeper insights into specific processes, useful for complex systems.

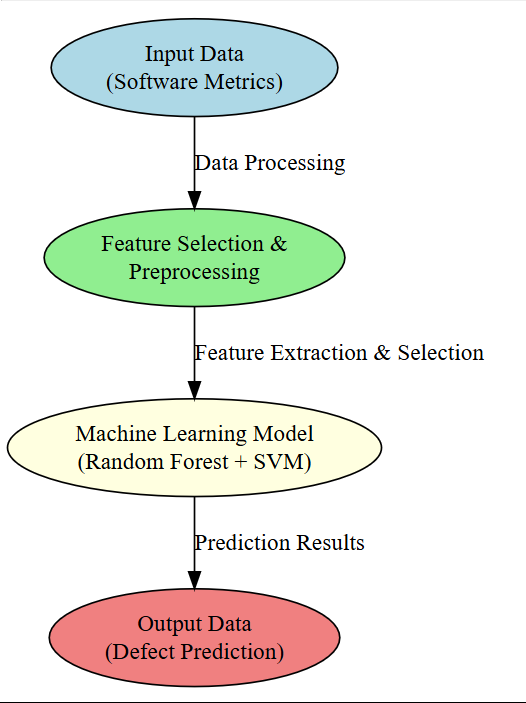


Figure 3.2: Dataflow Diagram of A Novel approach to improve software defect prediction accuracy using machine learning

1. **IMPLEMENTATION**

**4. IMPLEMENTATION**

The implementation of the Software Defect Prediction System follows a structured pipeline. The system begins with data preprocessing, where raw datasets undergo cleansing, normalization, and transformation. Feature selection is then applied to remove redundant attributes, optimizing model performance. The selected features are then used to train machine learning classifiers, ensuring accurate defect prediction. Various supervised learning models, including Random Forest, Support Vector Machine (SVM), Neural Networks, Bayesian Networks, and Logistic Regression, are implemented to classify software modules as defective or non-defective. The results are validated using accuracy, precision, recall, and F1-score to determine the best-performing model.

**4.1 ALGORITHMS USED**

1. **Random Forest (RF)**

Random Forest is an ensemble learning algorithm that constructs multiple decision trees during training and merges their outputs to improve accuracy and reduce overfitting. It is highly effective for defect prediction as it handles high-dimensional data and missing values efficiently. Each tree in the forest makes an independent decision, and the final prediction is determined by majority voting. This enhances robustness and prevents overfitting. In software defect prediction, Random Forest effectively differentiates between defective and non-defective modules by learning from past defect patterns, making it a preferred choice for handling complex software datasets.

1. **Support Vector Machine (SVM)**

Support Vector Machine (SVM) is a powerful classification algorithm that finds the optimal hyperplane that separates different classes in high-dimensional space. In defect prediction, SVM distinguishes defective and non-defective software modules by maximizing the margin between different data points. Kernel functions such as linear, polynomial, and radial basis function (RBF) improve classification performance, making SVM suitable for both small and large datasets. It is highly effective when dealing with imbalanced datasets by adjusting the decision boundary. SVM’s strong mathematical foundation and ability to handle non-linearly separable data make it a widely used technique in defect detection.

1. **Neural Networks (Multilayer Perceptron - MLP)**

Neural Networks, specifically Multilayer Perceptron (MLP), are deep learning models that simulate the human brain’s functioning. They consist of an input layer, hidden layers, and an output layer connected via weighted neurons. For defect prediction, MLP learns patterns from historical software defect data and generalizes this knowledge to classify new instances. Activation functions such as ReLU and sigmoid help in decision-making, while backpropagation optimizes weight adjustments. The model's high adaptability and pattern recognition capabilities enable it to achieve superior accuracy. MLP is particularly useful when datasets contain complex relationships between software attributes.

1. **Bayesian Networks (BN)**

Bayesian Networks (BN) are probabilistic graphical models that represent dependencies between variables using directed acyclic graphs (DAGs). Each node represents a variable, and edges define conditional dependencies. In software defect prediction, BN models the relationships between different software metrics (e.g., code complexity, number of bugs, and lines of code). It estimates the probability of a module being defective based on historical data. BN is highly interpretable and robust in handling uncertainty, making it useful when dealing with noisy and incomplete datasets. It is commonly used in defect prediction when explainability and probabilistic reasoning are required.

1. **Logistic Regression (LR)**

Logistic Regression is a simple yet powerful classification algorithm used for binary classification tasks. It predicts the probability of an instance belonging to a particular class using a sigmoid function. In defect prediction, LR analyzes software metrics and assigns probabilities to software modules being defective or non-defective. The model is easy to implement, computationally efficient, and interpretable. Feature selection enhances its performance by removing irrelevant attributes. Despite its simplicity, LR performs well on linearly separable data, making it a valuable baseline algorithm for comparing the performance of more complex machine learning models.

1. **Lazy IBk (Instance-Based k-Nearest Neighbors - IBk)**

Lazy IBk is a lazy learning algorithm, meaning it does not build a model during training but instead stores the entire dataset and makes predictions at runtime. It is based on the k-Nearest Neighbors (k-NN) algorithm, where classification is performed by comparing new instances to their closest k neighbors in the training data. In software defect prediction, Lazy IBk classifies software modules as defective or non-defective based on historical defect data. The choice of k and the distance metric (e.g., Euclidean, Manhattan) significantly impact prediction accuracy.

**4.2 SAMPLE CODE**

from django.db.models import Count

from django.db.models import Q

from django.shortcuts import render, redirect, get\_object\_or\_404

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import VotingClassifier

# Create your views here.

def login(request):

if request.method == "POST" and 'submit1' in request.POST:

username = request.POST.get('username')

password = request.POST.get('password')

try:

enter = ClientRegister\_Model.objects.get(username=username,password=password)

request.session["userid"] = enter.id

return redirect('ViewYourProfile')

except:

pass

return render(request,'RUser/login.html')

def index(request):

return render(request, 'RUser/index.html')

def Add\_DataSet\_Details(request):

return render(request, 'RUser/Add\_DataSet\_Details.html', {"excel\_data": ''})

def Register1(request):

if request.method == "POST":

username = request.POST.get('username')

email = request.POST.get('email')

password = request.POST.get('password')

phoneno = request.POST.get('phoneno')

country = request.POST.get('country')

state = request.POST.get('state')

city = request.POST.get('city')

address = request.POST.get('address')

gender = request.POST.get('gender')

obj = "Registered Successfully"

return render(request, 'RUser/Register1.html',{'object':obj})

else:

return render(request,'RUser/Register1.html')

def ViewYourProfile(request):

userid = request.session['userid']

obj = ClientRegister\_Model.objects.get(id= userid)

return render(request,'RUser/ViewYourProfile.html',{'object':obj})

def Predict\_Software\_Defect\_Type(request):

if request.method == "POST":

if request.method == "POST":

Fid=request.POST.get('Fid')

defect\_id=request.POST.get('defect\_id')

creation\_date=request.POST.get('creation\_date')

software\_name=request.POST.get('software\_name')

short\_description=request.POST.get('short\_description')

long\_description=request.POST.get('long\_description')

assignee\_name=request.POST.get('assignee\_name')

reporter\_name=request.POST.get('reporter\_name')

defect\_fix\_time=request.POST.get('defect\_fix\_time')

status\_category=request.POST.get('status\_category')

resolution\_category=request.POST.get('resolution\_category')

df = pd.read\_csv('Datasets.csv')

def apply\_response(Label):

if (Label == 0):

return 0 # normal

elif (Label == 1):

return 1 # critical

from django.db.models import Count, Avg

from django.shortcuts import render, redirect

from django.db.models import Count

from django.db.models import Q

import datetime

import xlwt

from django.http import HttpResponse

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

def serviceproviderlogin(request):

if request.method == "POST":

admin = request.POST.get('username')

password = request.POST.get('password')

if admin == "Admin" and password =="Admin":

detection\_accuracy.objects.all().delete()

return redirect('View\_Remote\_Users')

return render(request,'SProvider/serviceproviderlogin.html')

def View\_Prediction\_Of\_Software\_Defect\_Type\_Ratio(request):

detection\_ratio.objects.all().delete()

ratio = ""

kword = 'Normal'

print(kword)

obj = software\_defect\_prediction.objects.all().filter(Q(Prediction=kword))

obj1 = software\_defect\_prediction.objects.all()

count = obj.count();

count1 = obj1.count();

ratio = (count / count1) \* 100

if ratio != 0:

detection\_ratio.objects.create(names=kword, ratio=ratio)

ratio12 = ""

kword12 = 'Critical'

print(kword12)

obj12 = software\_defect\_prediction.objects.all().filter(Q(Prediction=kword12))

obj112 = software\_defect\_prediction.objects.all()

count12 = obj12.count();

count112 = obj112.count();

ratio12 = (count12 / count112) \* 100

if ratio12 != 0:

detection\_ratio.objects.create(names=kword12, ratio=ratio12)

obj = detection\_ratio.objects.all()  
def View\_Remote\_Users(request):

obj=ClientRegister\_Model.objects.all()

return render(request,'SProvider/View\_Remote\_Users.html',{'objects':obj})

def Download\_Predicted\_DataSets(request):

response = HttpResponse(content\_type='application/ms-excel')

# decide file name

filename="Predicted\_Datasets.xls"'

# creating workbook

wb = xlwt.Workbook(encoding='utf-8')

# adding sheet

ws = wb.add\_sheet("sheet1")

# Sheet header, first row

row\_num = 0

font\_style = xlwt.XFStyle()

# headers are bold

font\_style.font.bold = True

# writer = csv.writer(response)

obj = software\_defect\_prediction.objects.all()

data = obj # dummy method to fetch data.

for my\_row in data:

row\_num = row\_num + 1

ws.write(row\_num, 0, my\_row.Fid, font\_style)

ws.write(row\_num, 1, my\_row.defect\_id, font\_style)

ws.write(row\_num, 2, my\_row.creation\_date, font\_style)

ws.write(row\_num, 3, my\_row.software\_name, font\_style)

ws.write(row\_num, 4, my\_row.short\_description, font\_style)

ws.write(row\_num, 5, my\_row.long\_description, font\_style)

ws.write(row\_num, 6, my\_row.assignee\_name, font\_style)

ws.write(row\_num, 7, my\_row.reporter\_name, font\_style)

ws.write(row\_num, 8, my\_row.defect\_fix\_time, font\_style)

ws.write(row\_num, 9, my\_row.status\_category, font\_style)

ws.write(row\_num, 10, my\_row.resolution\_category, font\_style)

ws.write(row\_num, 11, my\_row.Prediction, font\_style)

wb.save(response)

return response

def train\_model(request):

detection\_accuracy.objects.all().delete()

df = pd.read\_csv('Datasets.csv', encoding='latin-1')

def apply\_response(Label):

if (Label == 0):

return 0 # normal

elif (Label == 1):

return 1 # critical

df['results'] = df['severity\_category'].apply(apply\_response)

cv = CountVectorizer()

X = df['Fid']

y = df['results']

print("Fid")

print(X)

print("Results")

print(y)

cv = CountVectorizer()

X = cv.fit\_transform(X)

models = []

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20)

X\_train.shape, X\_test.shape, y\_train.shape

print("Random Forest Classifier")

from sklearn.ensemble import RandomForestClassifier

rf\_clf = RandomForestClassifier()

rf\_clf.fit(X\_train, y\_train)

rfpredict = rf\_clf.predict(X\_test)

print("ACCURACY")

print(accuracy\_score(y\_test, rfpredict) \* 100)

print("CLASSIFICATION REPORT")

print(classification\_report(y\_test, rfpredict))

print("CONFUSION MATRIX")

print(confusion\_matrix(y\_test, rfpredict))

models.append(('RandomForestClassifier', rf\_clf))

detection\_accuracy.objects.create(names="Random Forest Classifier", ratio=accuracy\_score(y\_test, rfpredict) \* 100)

print("ANN")

from sklearn.neural\_network import MLPClassifier

mlpc = MLPClassifier().fit(X\_train, y\_train)

y\_pred = mlpc.predict(X\_test)

testscore\_mlpc = accuracy\_score(y\_test, y\_pred)

accuracy\_score(y\_test, y\_pred)

print(accuracy\_score(y\_test, y\_pred))

print(accuracy\_score(y\_test, y\_pred) \* 100)

print("CLASSIFICATION REPORT")

print(classification\_report(y\_test, y\_pred))

print("CONFUSION MATRIX")

print(confusion\_matrix(y\_test, y\_pred))

models.append(('MLPClassifier', mlpc))

detection\_accuracy.objects.create(names="ANN", ratio=accuracy\_score(y\_test, y\_pred) \* 100)

# SVM Model

print("SVM")

from sklearn import svm

lin\_clf = svm.LinearSVC()

lin\_clf.fit(X\_train, y\_train)

predict\_svm = lin\_clf.predict(X\_test)

svm\_acc = accuracy\_score(y\_test, predict\_svm) \* 100

print(svm\_acc)

print("CLASSIFICATION REPORT")

print(classification\_report(y\_test, predict\_svm))

print("CONFUSION MATRIX")

print(confusion\_matrix(y\_test, predict\_svm))

models.append(('svm', lin\_clf))

detection\_accuracy.objects.create(names="SVM", ratio=svm\_acc)

print("Logistic Regression")

from sklearn.linear\_model import LogisticRegression

reg = LogisticRegression(random\_state=0, solver='lbfgs').fit(X\_train, y\_train)

y\_pred = reg.predict(X\_test)

print("ACCURACY")

print(accuracy\_score(y\_test, y\_pred) \* 100)

print("CLASSIFICATION REPORT")

print(classification\_report(y\_test, y\_pred))

print("CONFUSION MATRIX")

print(confusion\_matrix(y\_test, y\_pred))

models.append(('logistic', reg))

print("Gradient Boosting Classifier")

from sklearn.ensemble import GradientBoostingClassifier

clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=1.0, max\_depth=1, random\_state=0).fit(

X\_train,

y\_train)

clfpredict = clf.predict(X\_test)

print("ACCURACY")

print(accuracy\_score(y\_test, clfpredict) \* 100)

print("CLASSIFICATION REPORT")

print(classification\_report(y\_test, clfpredict))

print("CONFUSION MATRIX")

print(confusion\_matrix(y\_test, clfpredict))

models.append(('GradientBoostingClassifier', clf))

detection\_accuracy.objects.create(names="Gradient Boosting Classifier",

ratio=accuracy\_score(y\_test, clfpredict) \* 100)

csv\_format = 'Results.csv'

df.to\_csv(csv\_format, index=False)

obj = detection\_accuracy.objects.all()

return render(request,'SProvider/train\_model.html', {'objs': obj})

#!/usr/bin/env python

"""Django's command-line utility for administrative tasks."""

import os

import sys

def main():

"""Run administrative tasks."""

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'a\_novel\_approach\_to\_improve\_software\_defect\_prediction.settings')

try:

from django.core.management import execute\_from\_command\_line

except ImportError as exc:

raise ImportError(

"Couldn't import Django. Are you sure it's installed and "

"available on your PYTHONPATH environment variable? Did you "

"forget to activate a virtual environment?"

) from exc

execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

main()

from django.db.models import Count, Avg

from django.shortcuts import render, redirect

from django.db.models import Count

from django.db.models import Q

import datetime

import xlwt

from django.http import HttpResponse

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

# Create your views here.

from Remote\_User.models import ClientRegister\_Model,software\_defect\_prediction,detection\_ratio,detection\_accuracy

def serviceproviderlogin(request):

if request.method == "POST":

admin = request.POST.get('username')

password = request.POST.get('password')

if admin == "Admin" and password =="Admin":

detection\_accuracy.objects.all().delete()

return redirect('View\_Remote\_Users')

return render(request,'SProvider/serviceproviderlogin.html')

def View\_Prediction\_Of\_Software\_Defect\_Type\_Ratio(request):

detection\_ratio.objects.all().delete()

ratio = ""

kword = 'Normal'

print(kword)

obj = software\_defect\_prediction.objects.all().filter(Q(Prediction=kword))

obj1 = software\_defect\_prediction.objects.all()

count = obj.count();

count1 = obj1.count();

ratio = (count / count1) \* 100

if ratio != 0:

detection\_ratio.objects.create(names=kword, ratio=ratio)

ratio12 = ""

kword12 = 'Critical'

print(kword12)

obj12 = software\_defect\_prediction.objects.all().filter(Q(Prediction=kword12))

obj112 = software\_defect\_prediction.objects.all()

count12 = obj12.count();

count112 = obj112.count();

ratio12 = (count12 / count112) \* 100

if ratio12 != 0:

detection\_ratio.objects.create(names=kword12, ratio=ratio12)

def Download\_Predicted\_DataSets(request):

response = HttpResponse(content\_type='application/ms-excel')

# decide file name

response['Content-Disposition'] = 'attachment; filename="Predicted\_Datasets.xls"'

# creating workbook

wb = xlwt.Workbook(encoding='utf-8')

# adding sheet

ws = wb.add\_sheet("sheet1")

# Sheet header, first row

row\_num = 0

font\_style = xlwt.XFStyle()

# headers are bold

font\_style.font.bold = True

# writer = csv.writer(response)

obj = software\_defect\_prediction.objects.all()

data = obj # dummy method to fetch data.

for my\_row in data:

row\_num = row\_num + 1

ws.write(row\_num, 0, my\_row.Fid, font\_style)

ws.write(row\_num, 1, my\_row.defect\_id, font\_style)

ws.write(row\_num, 2, my\_row.creation\_date, font\_style)

ws.write(row\_num, 3, my\_row.software\_name, font\_style)

ws.write(row\_num, 4, my\_row.short\_description, font\_style)

ws.write(row\_num, 5, my\_row.long\_description, font\_style)

ws.write(row\_num, 6, my\_row.assignee\_name, font\_style)

ws.write(row\_num, 7, my\_row.reporter\_name, font\_style)

ws.write(row\_num, 8, my\_row.defect\_fix\_time, font\_style)

ws.write(row\_num, 9, my\_row.status\_category, font\_style)

ws.write(row\_num, 10, my\_row.resolution\_category, font\_style)

ws.write(row\_num, 11, my\_row.Prediction, font\_style)

wb.save(response)

return response

def train\_model(request):

detection\_accuracy.objects.all().delete()

def serviceproviderlogin(request):

if request.method == "POST":

admin = request.POST.get('username')

password = request.POST.get('password')

if admin == "Admin" and password =="Admin":

detection\_accuracy.objects.all().delete()

return redirect('View\_Remote\_Users')

return render(request,'SProvider/serviceproviderlogin.html')

df = pd.read\_csv('Datasets.csv', encoding='latin-1')

def apply\_response(Label):

if (Label == 0):

return 0 # normal

elif (Label == 1):

return 1 # critical

df['results'] = df['severity\_category'].apply(apply\_response)

cv = CountVectorizer()

X = df['Fid']

y = df['results']

1. **RESULTS & DISCUSSION**

**5. RESULTS & DISCUSSION**

The following screenshots showcase the results of our project, highlighting key features and functionalities. These visual representations provide a clear overview of how the system performs under various conditions, demonstrating its effectiveness and user interface. The screenshots serve as a visual aid to support the project's technical and operational achievements.

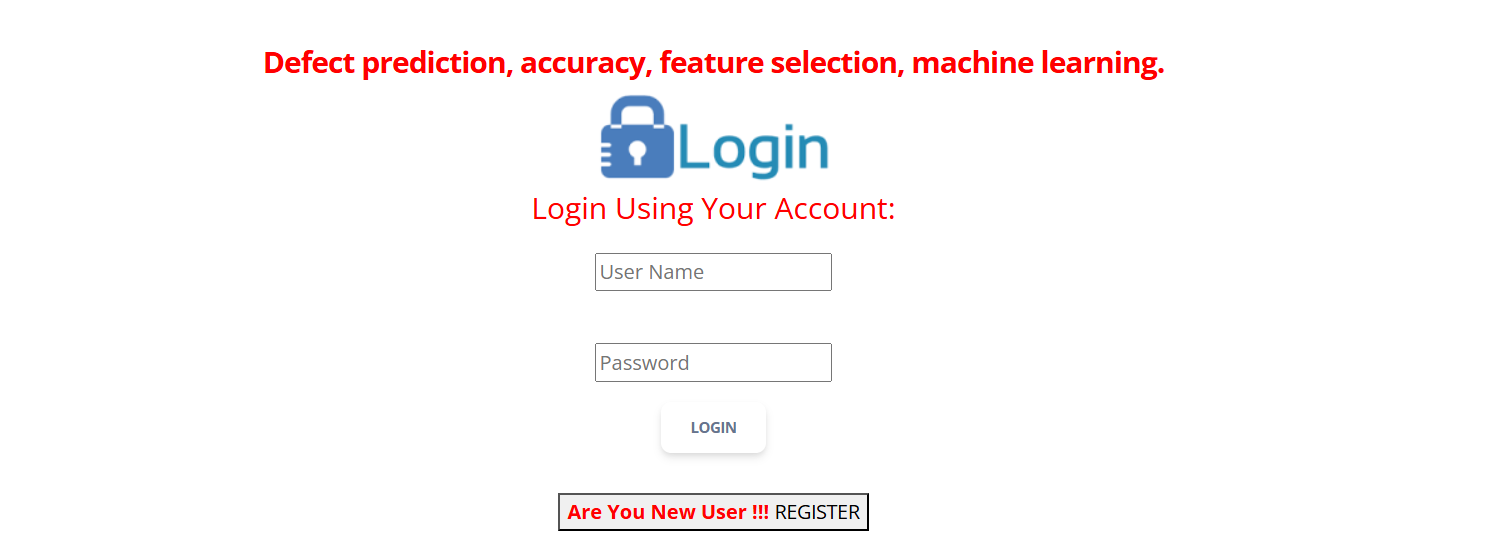
**5.1 GUI/Main Interface :**



**Figure**  **5.1 :** GUI/Main Interface of A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.2 Login page for remote users :**

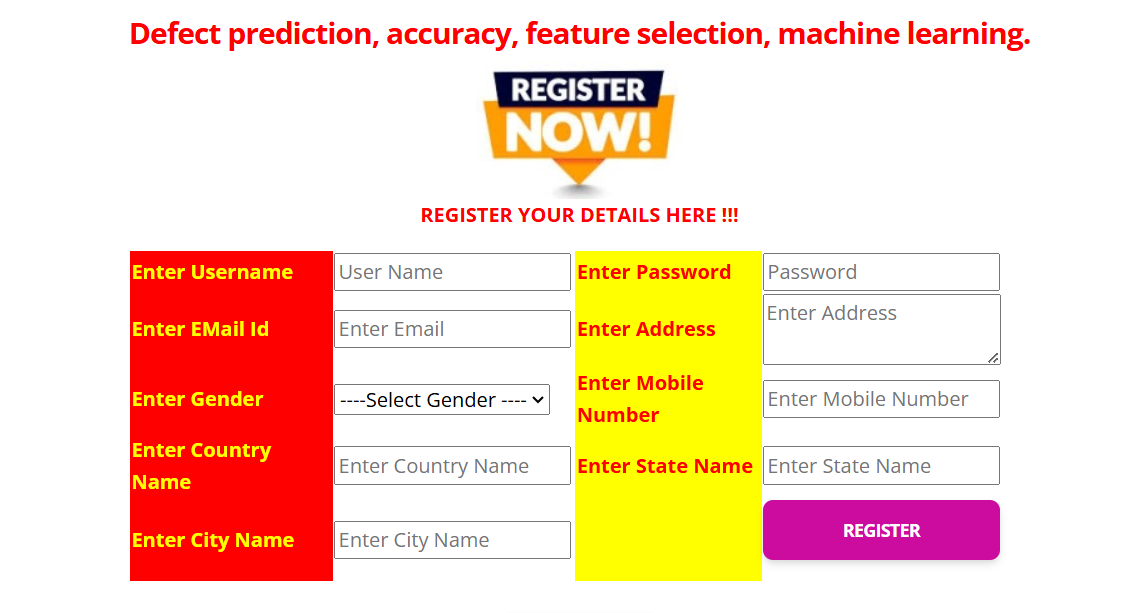
In below screen, it is used as login page for remote users



**Figure 5.2 :** Login page of A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.3 Registration page :**

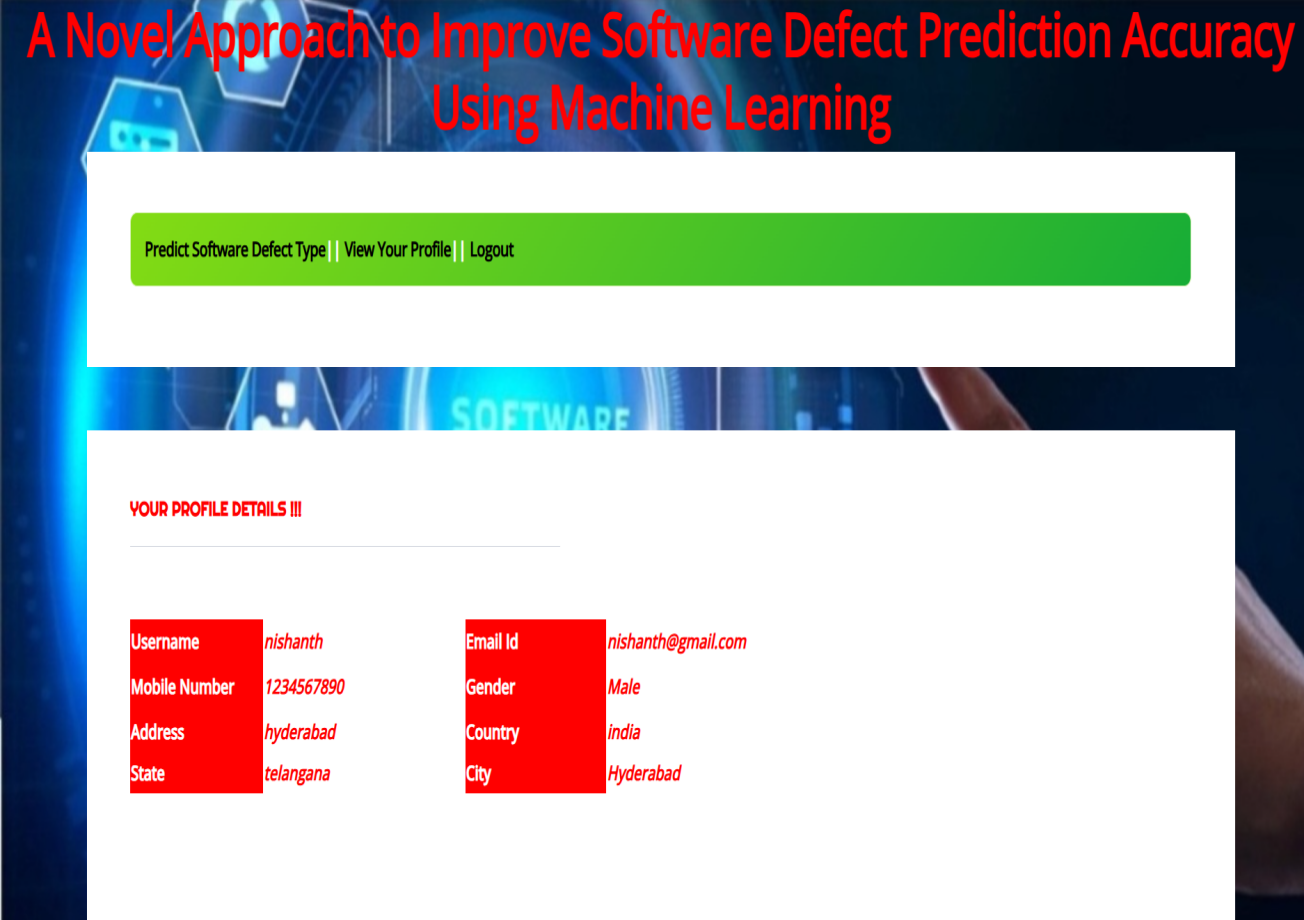
In below screen, the new user can register by entering the data in empty fields



**Figure 5.3 :** Registeration page of A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.4 Profile page of user:**

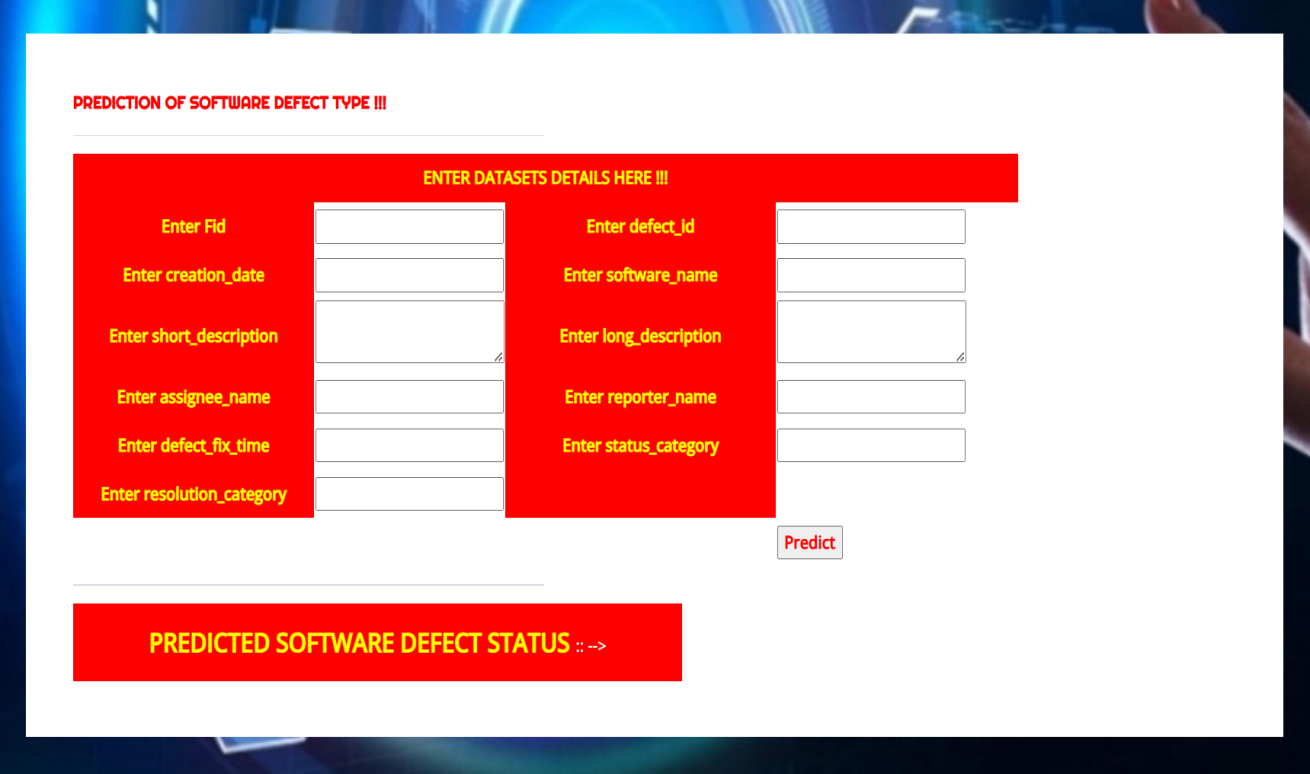
In below screen, we can see the pofile page of user(remote user).



**Figure 5.4 :** Profile page of A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.5 Form of prediction :**

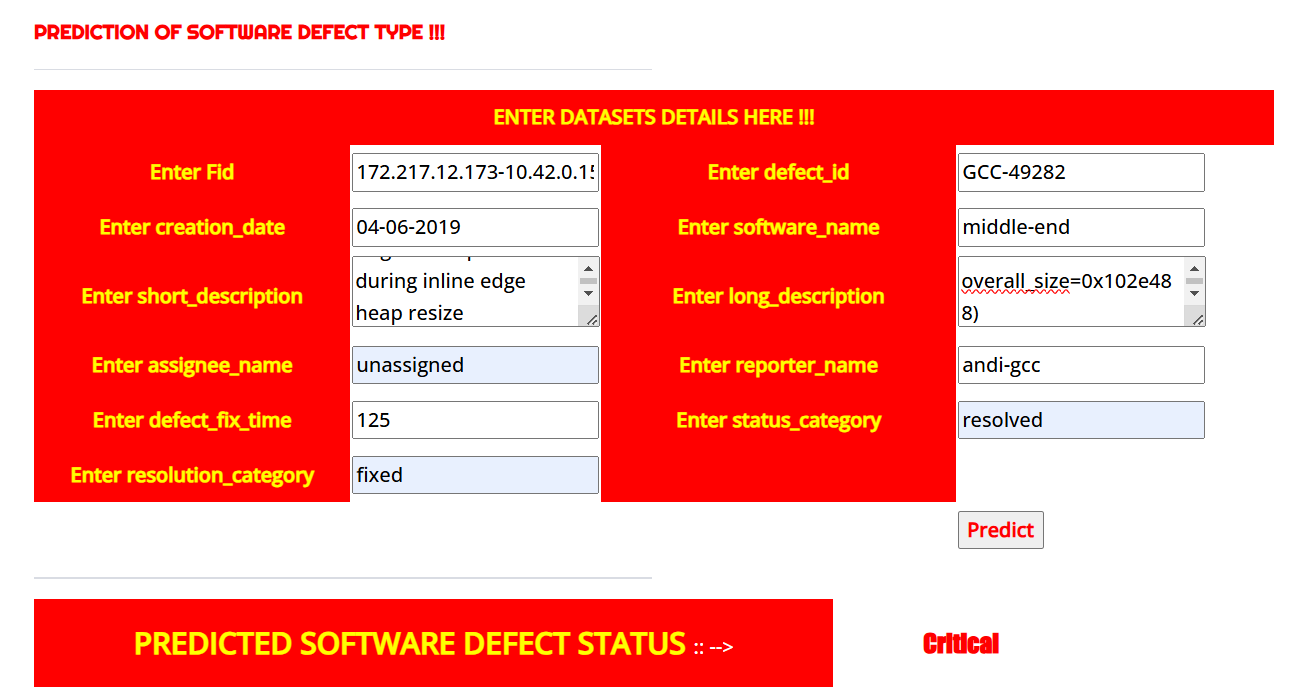
In below screen, the user enters details for prediction



**Figure 5.5 :** Form of predctionof A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.6 Defect status of given data:**

In below screen, we can see the defect status of the given data



**Figure 5.6 :** Defect status of given data of A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.7 Service provider login page :**

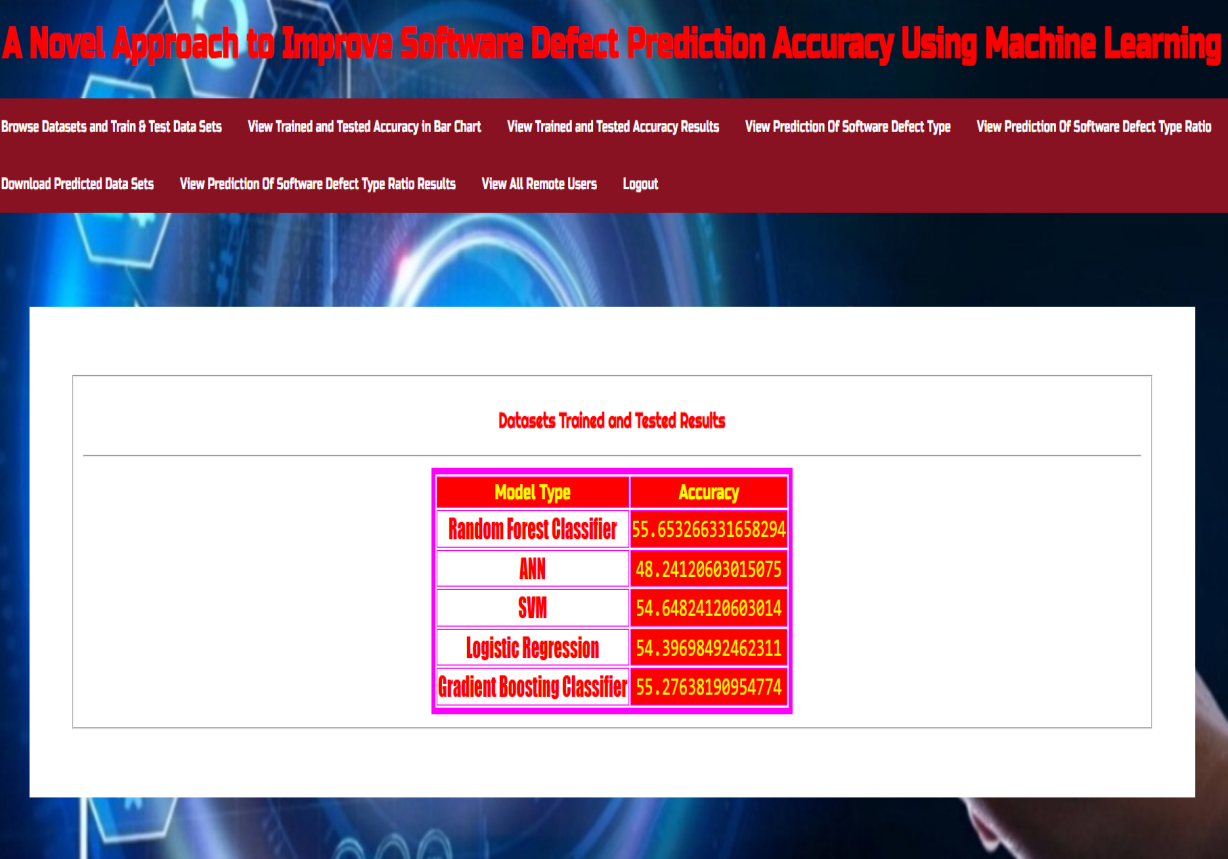
In below screen, the service provider enter login details and login to website



**Figure 5.7 :** Service provider login page of A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.8 Training and testing of data:**

In below screen, we get accuracy of trained and test data



**Figure 5.8 :** Training and testing data of A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.9 Training and testing of data accuracy in graph :**

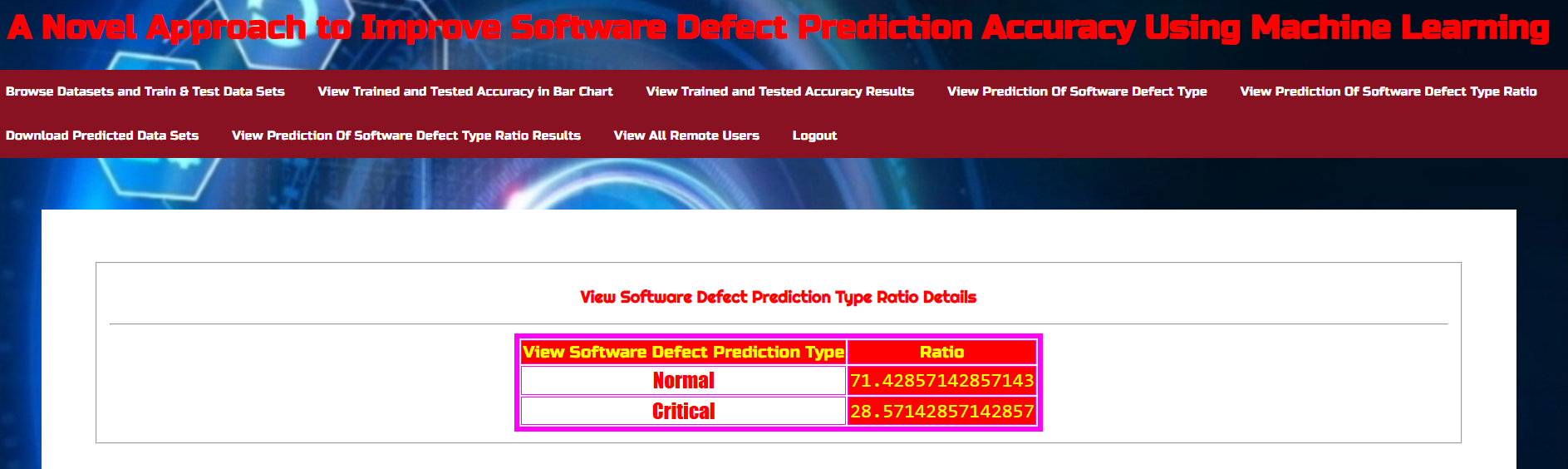
In below screen, the accuracy of trained and tested data



**Figure 5.9 :** Training and testing data accuracy of A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**5.10 Software Defect Prediction type ratio:**

In below video, we got Software Defect Prediction type ratio



**Figure 5.10 :** Software Defect Prediction type ratio for A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**6.VALIDATION**

A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

**6. VALIDATION**

The validation of this project primarily relies on extensive testing and well-defined test cases to ensure the accuracy and effectiveness of the software defect prediction system. The testing process includes multiple stages, such as dataset validation, model performance evaluation, and real-world testing. By implementing a structured validation strategy, we ensure that the system consistently delivers high accuracy in predicting software defects while minimizing false positives and false negatives.

### INTRODUCTION

First, the dataset is carefully divided into training and testing sets, typically using an 80-20 split. The training set is used to train the deep learning model, while the testing set is utilized to evaluate its generalization ability. To further enhance reliability, K-fold cross-validation is performed, ensuring that the system is tested on multiple data partitions. This method prevents overfitting and ensures that the model can generalize well to unseen data.

The accuracy of the system is measured using key performance metrics such as precision, recall, F1-score, and confusion matrix analysis. The confusion matrix provides valuable insights into correct and incorrect classifications, aiding in model refinement. Furthermore, various classifiers such as Random Forest, SVM, and Lazy IBk are compared to determine the best-performing model for software defect prediction.

Lastly, real-world deployment testing is conducted to simulate real-time defect detection in software modules. Continuous improvements are made to ensure the model remains effective in detecting software defects in live environments. This structured validation approach ensures that the proposed system is scalable and maintains high detection accuracy.

49

### TEST CASES

A Novel Approach to Improve Software Defect Prediction Accuracy using Machine learning

### TABLE 6.3.1 UPLOADING DATASET

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case ID | Test case name | Purpose | Test Case | Output |
| 1 | User uploads Dataset. | Verify dataset upload functionality | The user uploads the dataset for defect analysis | Dataset successfully loaded. |

**TABLE 6.3.2 CLASSIFICATION**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case ID | Test case name | Purpose | Input | Output |
| 1 | Classification test 1 | Check if the classifier correctly identifies non-defective modules | Non-defective software module | Classified as "Non-Defective" |
| 2 | Classification test 2 | Check if the classifier correctly identifies defective modules task | Defective software module | Classified as "Defective" |

50

**7. CONCLUSION & FUTURE ASPECTS**

**7. CONCLUSION & FUTURE ASPECTS**

In conclusion, the project has successfully achieved its objectives, showcasing significant progress and outcomes. The implementation and execution phases were meticulously planned and executed, leading to substantial improvements and insights. Looking ahead, the future aspects of the project hold immense potential. Future developments will focus on expanding the scope, integrating new technologies, and enhancing sustainability. These advancements will not only strengthen the existing framework but also open new avenues for growth and innovation, ensuring the project remains relevant and impactful in the long term. This strategic approach will drive continuous improvement and success.

**7.1 PROJECT CONCLUSION**

The proposed software defect prediction system successfully enhances software reliability by identifying defective modules before deployment. Through rigorous data preprocessing, feature selection, and model training, the system achieves high accuracy while minimizing false positives and negatives. By leveraging machine learning classifiers such as Random Forest, SVM, and Lazy IBk, the model efficiently predicts software defects, reducing maintenance costs and improving software quality. Extensive validation using K-fold cross-validation ensures the model generalizes well across various datasets, proving its robustness. The system not only streamlines software testing but also optimizes resource allocation, making it a valuable tool for software engineers.

Software defect prediction plays a critical role in software quality assurance. The proposed approach integrates feature selection and advanced machine learning models to enhance defect detection accuracy. By reducing redundant attributes and improving model efficiency, the system offers a scalable solution for real-world applications.

**7.2 FUTURE ASPECTS**

In the future, this system can be further improved by integrating deep learning models such as CNNs or LSTMs to enhance predictive accuracy. Additionally, incorporating automated hyperparameter tuning techniques can optimize model performance across diverse datasets. Real-time defect prediction through continuous learning mechanisms can make the system adaptive to evolving software structures. Furthermore, expanding the dataset to include multi-language and cross-domain software projects will enhance its applicability in real-world scenarios. Ultimately, advancements in AI-driven defect prediction will revolutionize software development, ensuring higher efficiency, reliability, and maintainability in the software industry.

52

**8. BIBLIOGRAPHY**

**8. BIBLIOGRAPHY**

**8.1 REFERENCES**

[1] M. A. Memon, M.-U.-R. Magsi, M. Memon, and S. Hyder, ‘‘Defects prediction and prevention approaches for quality software development,’’

Int. J. Adv. Comput. Sci. Appl., vol. 9, no. 8, pp. 451–457, 2018.

[2] M. Gayathri and A. Sudha, ‘‘Software defect prediction system using multilayer perceptron neural network with data mining,’’ Int. J. Recent

Technol. Eng., vol. 3, no. 2, pp. 2277–3878, 2014.

[3] R. Malhotra, L. Bahl, S. Sehgal, and P. Priya, ‘‘Empirical comparison ofmachine learning algorithms for bug prediction in open source software,’’in Proc. Int. Conf. Big Data Anal. Comput. Intell. (ICBDAC), AndhraPradesh, India, 2017, pp. 40–45, doi: 10.1109/ICBDACI.2017.8070806.

[4] M. S. Rawat and S. K. Dubey, ‘‘Software defect prediction models for quality improvement: A literature study,’’ Int. J. Comput. Sci. Issues, vol. 9, pp. 288–296, Jan. 2012.

[5] I. Singh, ‘‘A survey? Data mining techniques in software engineering,’’ Int. J. Res. IT, Manage. Eng., vol. 6, no. 3, pp. 30–34, Mar. 2016.

[6] N. Kalaivani and R. Beena, ‘‘Overview of software defect prediction using machine learning algorithms,’’ Int. J. Pure Appl. Math., vol. 118,pp. 3863–3873, Feb. 2018.

[7] M. Dhiauddin and S. Ibrahim, ‘‘A prediction model for system testing defects using regression analysis,’’ Int. J. Soft Comput. Softw. Eng., vol. 2, no. 7, pp. 55–68, Jul. 2012.

[8] R. Malhotra and A. Jain, ‘‘Fault prediction using statistical and machinelearning methods for improving software quality,’’ J. Inf. Process. Syst., vol. 8, no. 2, pp. 241–262, Jun. 2012.

**8.2 GITHUB LINK**

[https://github.com/Theachiever05/A-Novel-Approach-to-Improve-Software-Defect-Prediction-Accuracy-using-Machine-learning](https://github.com/Theachiever05/A-Novel-Approach-to-Improve-Software-Defect-Prediction-Accuracy-using-Machine-learning )