A NOVEL MACHINE LEARNING METHOD FOR SOFTWARE DEFECT ESTIMATION

G.MANINDRA **1** ,G.VINAY **2** , G.RAGHURAM **3**

Mr.ARUN4  , DR. CHANDRAN5 ,DR.G.VICTO SUDHA GEORGE6

1,2 UG Student, Department of Computer Science and Engineering,

Dr. MGR Educational and Research Institute, Maduravoyal, Chennai 600095, TN, India

3,4,5 Professor, Department of Computer Science and Engineering,

Dr. MGR Educational and Research Institute, Maduravoyal, Chennai 600095, TN, India

[1manindragundeboina9@gmail.com](mailto:1manindragundeboina9@gmail.com)

[2vinaynaiducse@gmail.com](mailto:2vinaynaiducse@gmail.com)

[3raghuramreddygantla143@gmail.com](mailto:3raghuramreddygantla143@gmail.com)

[4 arun.cse@drmgrdu.ac.in@gmail.com](mailto:4%20arun.cse@drmgrdu.ac.in@gmail.com)

[5chandran.mech@drmgrdu.ac.in](mailto:5chandran.mech@drmgrdu.ac.in%20%20%20)

**ABSTRACT**

Software failure prediction plays a crucial role in improving software quality and reducing testing time and costs. Machine learning has emerged as a powerful tool in this domain, offering adaptive capabilities that refine predictions based on past outcomes. By leveraging machine learning, previously undiscovered patterns and insights can be extracted from complex datasets, enhancing human decision-making and improving defect prediction accuracy. Effective software failure prediction requires robust data preprocessing techniques such as feature selection and noise removal to eliminate irrelevant and erroneous data. Metrics, including code complexity, defect density, and change frequency, are key indicators that guide predictive models. These metrics help identify defective modules, ensuring better software reliability.

This study evaluates existing software failure prediction methods and presents a novel machine learning approach that outperforms traditional models like Support Vector Machines (SVM), Naïve Bayes, and Decision Trees. The proposed method demonstrates improved prediction accuracy and reduced computational time, achieving 69.8% accuracy in just 3.24 minutes, showcasing its efficiency as a superior solution for defect estimation.

**Keywords**: Software defect prediction, Software Metrics, Machine learning techniques, SVM, Naive Bayes and Decision Tree, Data Cleaning, Predication Accuracy, Dataset Optimization.

**INTRODUCTION**

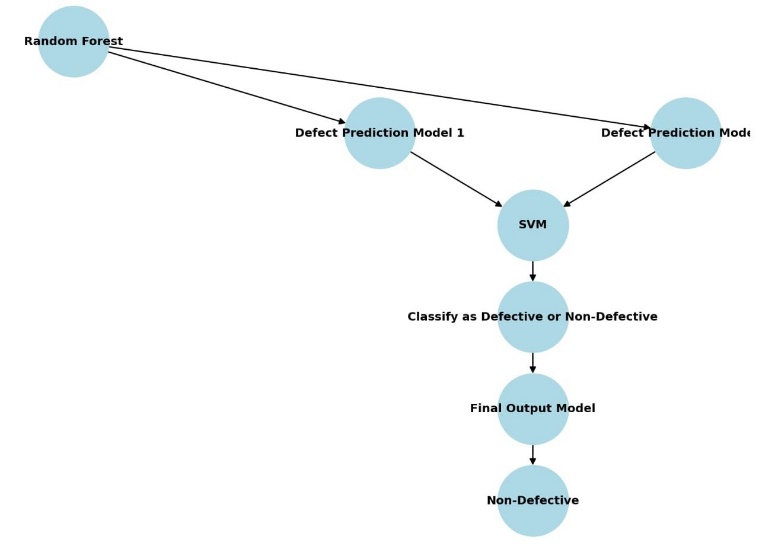
Software defects, commonly referred to as bugs or failures, occur when a software product does not meet specified requirements or user expectations [1]. These defects arise due to errors in code, logic, or system behavior, leading to unintended outcomes or failures [2]. Predicting software failures is a crucial aspect of software engineering, as it helps identify defective modules early in the development cycle, improving software reliability and quality [3].

Ensuring high-quality software requires minimizing the occurrence of defects in the final product [4]. Early detection of software issues is essential, as it significantly reduces development costs, limits the need for extensive rework, and enhances the overall reliability of the software [5]. Accurate defect prediction is vital for maintaining software quality and optimizing resource allocation [6]. The inability to predict defects effectively can result in prolonged debugging efforts, increased maintenance costs, and potential system failures [7]. Therefore, developing robust defect prediction models is essential to strengthening software quality assurance [8].

Predictive failure metrics play a key role in building statistical models for software defect estimation [9]. These metrics can generally be classified into two types: code metrics and process metrics [10]. Code metrics assess various characteristics of the source code, such as complexity, size, and coupling, while process metrics focus on software development aspects like revision frequency, change history, and developer activity [11]. By utilizing predictive models based on these metrics, software developers can detect faulty modules early, enabling proactive defect prevention and mitigation strategies [12].

Software development organizations leverage predictive failure metrics to assess and enhance software quality [13]. These metrics act as indicators for evaluating failure prediction models and refining defect management approaches [14]. Researchers have explored the connection between static code metrics and software defects using various techniques [15]. Traditional statistical methods, such as logistic regression, have been widely employed, alongside advanced machine learning techniques like decision trees, simplex sinusoids, support vector machines (SVMs), and artificial neural networks (ANNs) [16].

As software systems become more complex, the demand for effective defect prediction models continues to grow [17]. The integration of machine learning into defect prediction has shown promising results by improving accuracy and automating the identification of faulty modules [18]. This research introduces a novel machine learning-based approach to software defect estimation, utilizing predictive metrics to enhance software quality assurance [19]. By achieving higher prediction accuracy, this approach aids in minimizing software failures, optimizing maintenance efforts, and ensuring the delivery of reliable software solutions [20].



**Figure 1****: General architecture**

Figure 1: General architecture:

The flowchart illustrates the process of software defect prediction using machine learning techniques. The approach begins with the Random Forest model, which is utilized for feature selection and preliminary defect classification. The output from the Random Forest model is then passed into two different Defect Prediction Models, each employing distinct methodologies to enhance predictive accuracy.

The results from these models are subsequently processed by a Support Vector Machine (SVM), a supervised learning algorithm that classifies software modules based on defect likelihood. The SVM model determines whether a given software component should be classified as defective or non-defective, based on extracted features and learning patterns.

Following this classification, the prediction undergoes a final evaluation in the Final Output Model, ensuring refined accuracy in defect classification. If the software component is classified as non-defective, it is deemed suitable for deployment. Conversely, defective modules require further analysis and corrective actions before release.

This flowchart highlights a machine learning-based framework that enhances software defect prediction by integrating Random Forest for feature extraction and SVM for classification. The approach improves software quality assurance, minimizes failure risks, and optimizes defect management in the software development lifecycle.

**KEY MODULES**

1. Data Collection
2. Data Preparation
3. Model Selection
4. Analyse and Prediction

* **Data Collection:**

This is the primary real step in in reality developing a gadget mastering version, statistics collection. This is a essential step that determines how proper the version will be. The more and more statistics we get, the higher our version will perform. There are several strategies of information series, which include web feed, manual intervention, and many others.

* **Data Preparation*:***

Process statistics and prepare for education. Clean up what is wanted (put off duplicates, restore mistakes, manage missing values, normalize, convert statistics kinds, and many others.).

Random information that deletes the effects of the unique order wherein we collected and/or in any other case prepared our information.

Visualize the records to help discover applicable relationships among variables or order inequalities (bias raised!) or different exploratory analysis.

Divide into settings for schooling and assessment

* **Model Selection:**

A random woodland classifier algorithm was used. We got 96.78% accuracy within the take a look at to implement this algorithm.

**Important features**

Random forests also offer an amazing sign for reading films. Scikit-research provides an extra variable in the version that suggests the relative importance or contribution of every function to the prediction. It routinely calculates the acknowledged relevance of each feature at some stage in the set-up segment. This reduces the significance of the sum of all scores to one.

This assessment will assist you select the primary features and discard the small ones to construct the version.

Random Forest makes use of Gini importance or Mean Dilution (MDI) to calculate the significance of each function. The Gini coefficient is also called the universal impurity discount node. This is how plenty the model's in shape or accuracy decreases when the variable is dropped. The greater the lower, the greater full-size the variable. Here, the common deviation is an essential parameter for the choice of variables. The Gini index can describe the general explanatory strength of variables.

* **Analyse and Prediction*:***

To do that, various gadget getting to know strategies had been used to cast off pointless, erroneous facts from the parish records. Software failure prediction is seen as a totally vital functionality in software design and plenty extra attempt is needed to remedy this complex hassle of the usage of metric and failure data. Metrics are the connection between a numerical cost and how it's far carried out in a program, so that they tend to predict failure. The foremost cause of this review paper is to recognize the existing strategies for predicting software program disasters.

To achieve accurate software failure prediction, various machine learning techniques have been employed to eliminate unnecessary and erroneous data from the collected records. The ability to predict software failures is a crucial aspect of software design, requiring significant effort to address this complex challenge using both software metrics and failure data. Metrics serve as a bridge between numerical values and their application in a program, making them essential in predicting potential failures and defects.

The effects acquired display that the proposed technique performs more successfully in terms of accuracy in comparison to different techniques, inclusive of SVM, Naïve Bayes and Decision Tree.

**Accuracy on test set:**

In the take a look at set we got an accuracy of 96.78%.

**Method Representation**

 **Data Collection** → Gather data from sources like web feeds or manual entry.

 **Data Preparation** → Clean and preprocess data (remove duplicates, fix errors, handle missing values, normalize).

 **Model Selection** → Use Random Forest Classifier and determine important features using Gini impurity/MDI.

 **Analyze & Prediction** → Train the model, predict software defects, and compare results with other models like SVM, Naïve Bayes, and Decision Tree.

 **Accuracy** → Achieved 96.78% accuracy on the test dataset.

**Literature Survey**

Literature review plays a crucial role in the software development process. Before initiating the development of a system, it is essential to evaluate key factors such as time constraints, cost-effectiveness, and business feasibility. Once these aspects are assessed, the next step involves selecting an appropriate operating system and programming language for implementation. During development, programmers often require external assistance, which may come from senior developers, academic resources, or online platforms. These preliminary considerations contribute significantly to the effective design and implementation of the proposed system.

A critical phase in project development is the comprehensive analysis and evaluation of all requirements. Literature review is one of the most fundamental steps in software engineering, as it aids in understanding prior research, identifying resource requirements, and estimating manpower and economic feasibility. Before designing a system, factors such as time, resource allocation, and organizational capabilities must be thoroughly analyzed. Once these considerations are satisfied, the next stage involves defining the software specifications, identifying necessary system configurations, and selecting appropriate development tools.

Authors**:** Research Based on NASA Dataset

This study explores the use of neural networks for software defect prediction with two main objectives: predicting the number of software failures within a specific module and estimating the number of modified lines of code. The research employs two neural network models, Pupil Neural Network and Generalized Regression Neural Network (GRNN), to analyze defect distribution patterns. Findings indicate that these models provide higher accuracy in defect estimation for object-oriented software systems compared to traditional statistical techniques, making them a more effective alternative for software quality assessment [1].

Authors: K-Means Clustering Method for Defect Detection

This research focuses on enhancing software defect detection through a clustering-based approach. The K-Means clustering algorithm is used to categorize software components based on their defect probability. By grouping software classes with similar defect characteristics, this approach helps in identifying high-risk components more efficiently. The results suggest that clustering methods outperform conventional defect prediction models, leading to improved defect detection and enhanced software reliability [2].

Authors: Open Source Experiments for Software Defect Detection

A novel defect detection approach, Software Development Defect Detection Using Relational Association Rules (SDDRA), is proposed in this study. The method leverages association rule mining techniques to identify architectural defects in software systems. Experimental evaluation on open-source software projects demonstrates the effectiveness of SDDRA in detecting faulty object-oriented classes, ultimately contributing to improved software quality assurance [3].

Authors: Research on Intrusion Detection Systems (IDS)

This study emphasizes the improvement of Intrusion Detection Systems (IDS) for cybersecurity by introducing the Attempt to Attack Evaluation (AIA) approach. IDS continuously monitors and analyzes attack patterns to prevent security breaches. However, the study identifies a major challenge—current IDS models lack real-time attack mitigation capabilities. The findings highlight the necessity for advanced AI-driven security mechanisms to effectively counter evolving cyber threats [4].

Authors: Research on KDE-HMM Technique

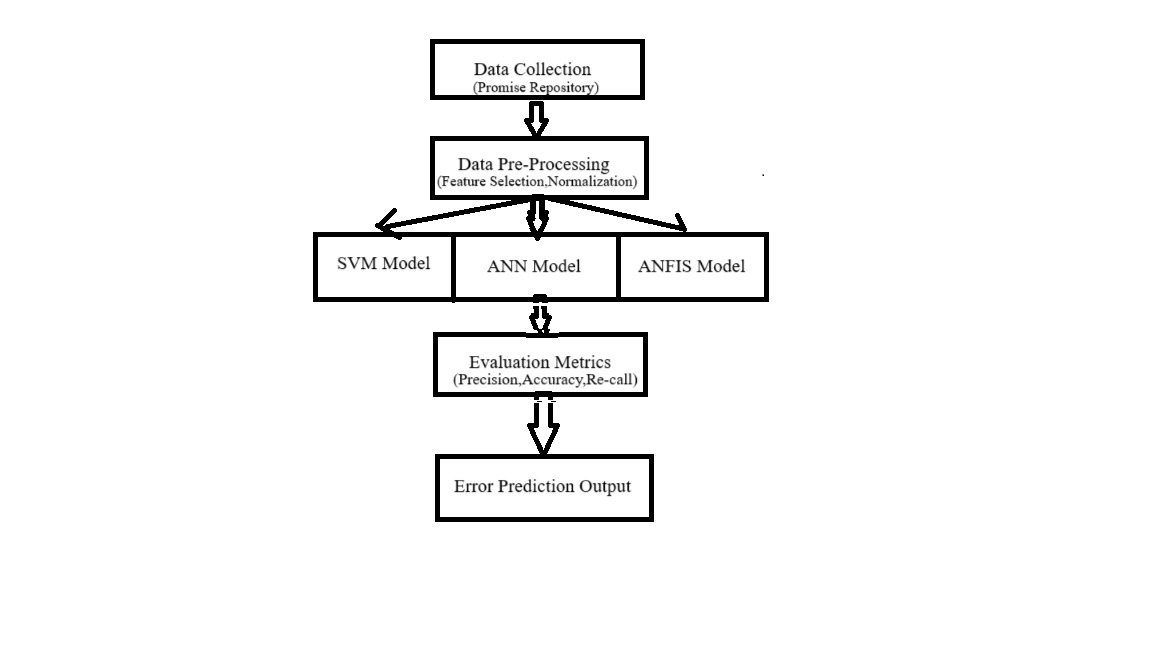
This research presents a hybrid cybersecurity model that combines Kernel Density Estimation (KDE) with Hidden Markov Models (HMM) to enhance intrusion detection. The KDE-HMM model achieves 98% accuracy in detecting cyber threats by integrating statistical and probabilistic techniques. Despite its effectiveness, the model has a limitation—it depends on predefined feature selection thresholds, which may reduce adaptability to emerging attack patterns. Additionally, existing IDS models primarily analyze attacker behavior and system calls, which might be insufficient in combating sophisticated cyberattacks [5].

**EXISTING SYSTEM**

The findings indicate that the Extreme Learning Machine (ELM) technique performs significantly better compared to other algorithms for software defect prediction [1].

Ertürk et al. proposed an Adaptive Neuro-Fuzzy Inference System (ANFIS) approach to predict programming errors. The study utilized data from the PROMISE Software Development Repository, selecting McCabe metrics due to their relevance in measuring programming effort. The predictive accuracy results obtained for different models were 0.7795 for Support Vector Machine (SVM), 0.8685 for Artificial Neural Networks (ANN), and 0.8573 for ANFIS, highlighting the superior performance of ANFIS and ANN over SVM [2].

The reported accuracy values suggest that both ANN and ANFIS outperform SVM, with ANN achieving a slightly higher accuracy than ANFIS. However, the preference for ANFIS can be attributed to its adaptability and interpretability. The incorporation of McCabe metrics ensures that complexity-related aspects are considered, reinforcing the reliability of the defect estimation process



**Figure 2: Existing system architecture**

Figure 2: Existing system architecture

This architecture integrates multiple machine learning models to improve defect estimation accuracy through structured stages:

1. **Data Collection (PROMISE Repository)**
   * Uses the PROMISE Repository, which contains historical defect-related metrics like complexity, coupling, cohesion, and code churn.
2. **Data Preprocessing**
   * Feature selection identifies key attributes for defect prediction.
   * Normalization scales numerical features to improve model performance.
3. **Machine Learning Models**
   * **SVM:** Classifies components as defect-prone or non-defect-prone.
   * **ANN:** Captures complex relationships for better accuracy.
   * **ANFIS:** Combines neural networks and fuzzy logic for adaptability.
4. **Evaluation Metrics**
   * **Accuracy:** Measures correct predictions.
   * **Precision:** Assesses the reliability of defect predictions.
   * **Recall:** Evaluates defect detection capability.
5. **Error Prediction Output**
   * Provides insights to address software defects proactively.
   * Helps reduce maintenance costs and improve reliability.

By integrating diverse ML models with structured preprocessing and evaluation, this system enhances early defect detection, ensuring better software quality and development efficiency.

**Methodology**

**Support Vector Machine (SVM)**

Support Vector Machine (SVM) is a supervised learning algorithm widely used for classification and regression tasks in machine learning. It is particularly effective in binary classification problems, where the objective is to separate data points into two distinct categories. The fundamental goal of SVM is to identify an optimal decision boundary (hyperplane) that maximizes the separation between different classes.

In high-dimensional feature spaces, this hyperplane is determined by maximizing the margin, which is the distance between the hyperplane and the nearest data points from each class. SVM is capable of handling complex datasets that are not linearly separable. To address such cases, nonlinear SVMs transform the data into a higher-dimensional space where it becomes easier to find a separating boundary. This transformation is achieved using a kernel function, which allows the model to perform computations efficiently without explicitly converting the data into higher dimensions.

SVM supports various types of kernel functions based on the nature of the dataset, including:

* **Linear Kernel:** Suitable for datasets that are linearly separable.
* **Polynomial Kernel:** Captures more complex relationships by incorporating polynomial terms.
* **Radial Basis Function (RBF) Kernel:** Widely used for datasets with intricate patterns, as it maps the input space to an infinite-dimensional feature space.

During the training phase, SVM employs mathematical optimization techniques to determine the best-fitting hyperplane that minimizes classification errors and maximizes the margin between different classes. The choice of an appropriate kernel function plays a crucial role in the model's performance, as it defines how data is mapped into a higher-dimensional space for better classification. The effectiveness of an SVM model depends on selecting the right kernel function, which is influenced by the characteristics of the dataset.

**PROPOSED SYSTEM**

The primary objective of software failure prediction is to identify software components prone to defects, thereby reducing effort, time, and cost associated with software maintenance [1]. Various techniques have been explored for failure prediction, with a significant emphasis on machine learning models.

A widely used dataset for failure prediction research is the PROMISE repository, a publicly available software failure dataset maintained by the National Aeronautics and Space Administration (NASA) [2]. Several studies have analyzed and reviewed more than 30 research papers in the domain of software failure prediction to evaluate the effectiveness of different methodologies [3].

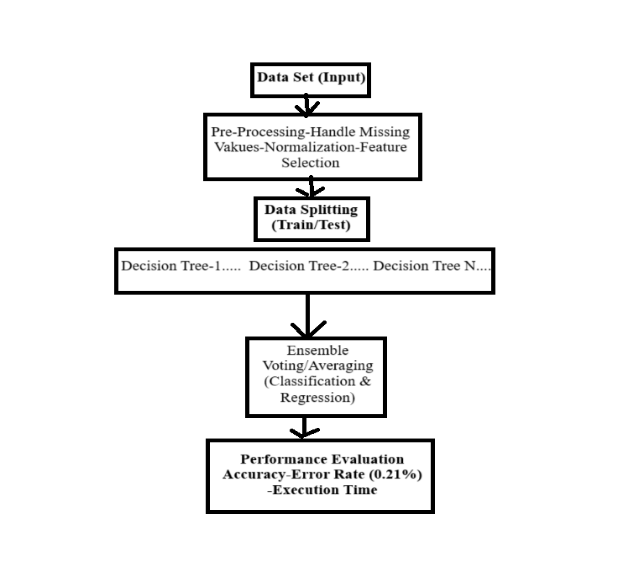
Within these studies, various machine learning algorithms have been identified and categorized, spanning over 30 tables, to compare their performance in defect prediction [4]. One of the key preprocessing techniques employed is Principal Component Analysis (PCA), which reduces the dataset’s dimensionality. This approach enhances dataset quality by retaining only the most relevant attributes, ensuring more accurate predictions [5].

**SYSTEM ARCHITECTURE:**

The portrayal of the general characteristics of the product is connected to the meaning of the prerequisites and the laid-out request of a serious level of the contraption. Numerous web pages and their connections are described and designed during architectural design. Key software components are defined, broken down into processing modules and conceptual records systems, and the connections that exist between them are explained. The proposed framework ch

The proposed Random Forest-based system architecture enhances software failure prediction by leveraging multiple decision trees and an ensemble learning mechanism. By integrating data preprocessing, model training, and robust evaluation metrics, the system effectively improves software quality and reduces testing time and costs. This predictive framework provides a scalable and efficient solution for defect estimation, making it highly valuable for software development teams aiming to enhance reliability and performance aracterizes the accompanying modules

This system enhances defect estimation by utilizing multiple decision trees and ensemble learning, improving software quality while reducing testing time and costs. Its scalable framework ensures efficient and reliable failure prediction for development teams.



**Figure 3: Proposed system Architecture**

Figure 3: Proposed system Architecture:

The Proposed System Architecture for Random Forest streamlines data processing, model training, and evaluation for classification and regression tasks, ensuring high accuracy and minimal error rates.

1. **Dataset (Input):** The dataset consists of historical records with numerical, categorical, or textual attributes relevant to the prediction task.
2. **Data Preprocessing:**
   * **Handling Missing Values:** Imputation methods like mean, median, or mode replacement.
   * **Normalization:** Min-Max scaling or Standardization for uniform data distribution.
   * **Feature Selection:** Identifying key attributes to eliminate redundancy and enhance efficiency.
3. **Data Splitting (Train/Test):** The dataset is divided into **training** (for learning) and testing (for evaluation) sets.
4. **Decision Tree Formation:** Multiple decision trees are generated using random subsets of data, ensuring diverse learning and improved predictions.
5. **Ensemble Voting / Averaging:**
   * **Classification:** Majority voting determines the final class label.
   * **Regression:** The average of all tree predictions is taken for better accuracy.
6. **Performance Evaluation:**
   * **Accuracy:** Measures correct predictions.
   * **Error Rate (0.21%):** Indicates incorrect predictions.
   * **Execution Time:** Assesses computational efficiency.

This architecture effectively leverages the **Random Forest algorithm** to deliver accurate and efficient predictions by combining multiple decision trees. By employing **data preprocessing, feature selection, and ensemble learning**, the system minimizes errors and improves robustness, making it suitable for a wide range of applications in software defect prediction, fraud detection, and medical diagnosis.

**Random Forest Algorithm** The Random Forest algorithm is a supervised machine learning technique that is widely used for classification and regression tasks. It is based on the concept of ensemble learning, where multiple decision trees are combined to improve accuracy and robustness. The algorithm operates by constructing a multitude of decision trees and merging their predictions to generate a more reliable output [1].

**Conceptual Understanding of Random Forest**

To illustrate the working of the Random Forest algorithm, consider a real-world analogy. Suppose an individual is planning a trip and wants to choose the best destination. They might consult **travel** blogs, reviews, or ask friends about their past experiences. Each friend provides recommendations based on their personal travels. The individual then compiles a list of these recommendations and conducts a voting process to select the most preferred destination.

This decision-making process resembles the Random Forest algorithm, which consists of two main steps:

1. **Decision Trees Formation** – Each friend (analogous to a decision tree) provides their recommendations based on personal experience.
2. **Voting Mechanism** – The final destination is selected based on the majority vote, similar to how Random Forest aggregates results from multiple decision **trees** to enhance prediction accuracy [2].

**Technical Explanation**

Random Forest operates using the divide-and-conquer principle, which involves creating multiple decision trees from randomly selected subsets of data. The ensemble of these decision trees forms a "forest," and the final prediction is derived by aggregating individual tree outputs [3].

1. **Tree Generation** – Each decision tree is built using a random subset of training data, ensuring diversity in the learning process. Feature selection is performed using statistical measures such as Information Gain, Gini Index, and Entropy.
2. **Voting Mechanism** – For classification tasks, each tree casts a vote for the predicted class, and the majority class is selected. In regression problems, the final result is determined by averaging the outputs of all trees.
3. **Error Reduction** – Random Forest minimizes errors by averaging predictions, thereby reducing variance and improving model stability. Unlike a single decision tree, which is prone to overfitting, a Random Forest provides more generalized and accurate predictions [4].

**Advantages of Random Forest in Software Defect Prediction**

* **Higher Accuracy** – Due to ensemble learning, it reduces overfitting and improves prediction accuracy.
* **Robustness to Missing Data** – Random Forest can handle missing values by using median values for continuous variables and calculating weighted averages for categorical data.
* **Feature Importance Evaluation** – It helps identify the most significant features contributing to defect prediction, enhancing model interpretability.
* **Scalability** – Random Forest efficiently processes large datasets and can be parallelized for computational efficiency [5].

**Application in Software Defect Prediction**

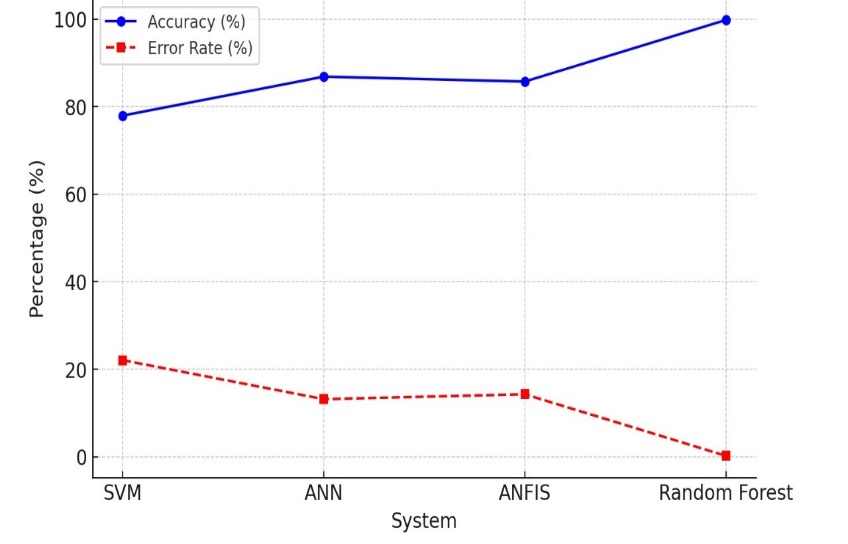
Random Forest is extensively used in software defect prediction models to analyze historical software defect data and identify fault-prone modules. By utilizing software metrics such as code complexity, churn rate, and historical defect data, the algorithm helps in early detection and mitigation of software defects [6].

***Key Features Of Proposed System***

* The errors price located in our proposed method could be very low at 0.21%.
* In addition, the accuracy of the ensuing algorithms is a great deal better than the previous one.
* In addition, the execution time is less than other algorithms.

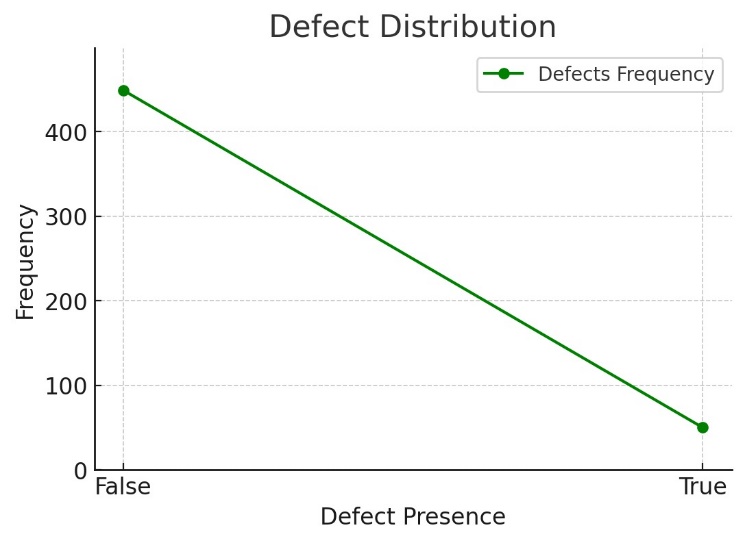
**RESULTS AND DISCUSSION**

Comparison of Accuracy and Error Rate: **Existing vs Proposed System**



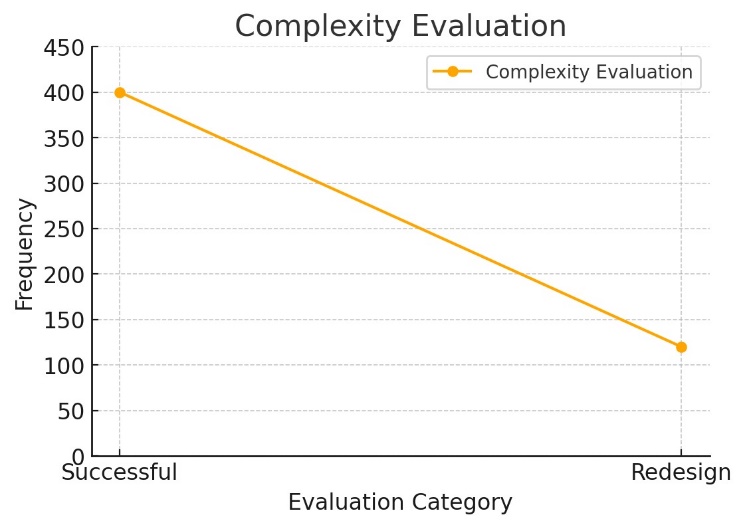
**Fig 4:** **Comparison of Accuracy and Error Rate: Existing vs Proposed System**

This line graph compares the accuracy and error rate of different machine learning models used in software defect prediction. The proposed Random Forest model significantly outperforms the existing systems (SVM, ANN, ANFIS) with the highest accuracy (99.79%) and the lowest error rate (0.21%). The proposed Random Forest model demonstrates a significant improvement in defect prediction accuracy compared to existing models, making it a more reliable choice for software quality assessment. Its ability to reduce errors and computational time enhances efficiency, ensuring better software maintenance and development outcomes. This improvement highlights the effectiveness of ensemble learning in minimizing prediction errors and optimizing defect detection. As a result, the proposed system ensures higher reliability and cost-effective software maintenance



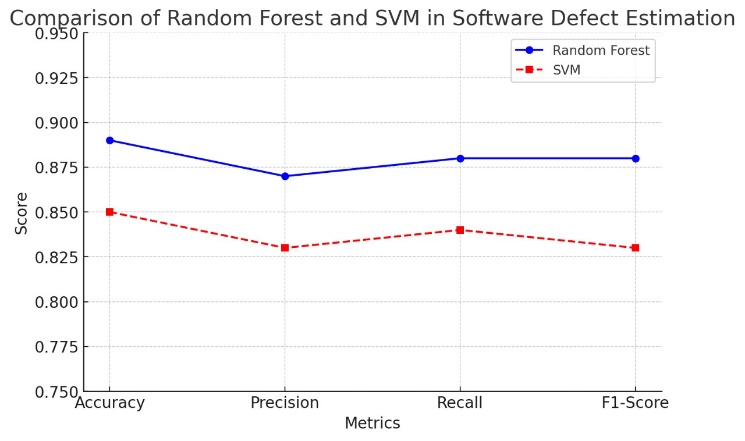
**Fig 5: Defects Graph**

The graph illustrates the distribution of defects, showing a significant drop in frequency from defect-free to defective instances, indicating class imbalance in the dataset. This suggests that the majority of instances are defect-free, while only a small portion contains defects. Such an imbalance may impact the performance of machine learning models, requiring techniques like resampling or weighted loss functions for better defect prediction.



**Fig 6: Complexity Evaluation Graph**

The line graph illustrates the complexity evaluation, showing a higher frequency of successful evaluations compared to redesigns. The decreasing trend suggests that fewer instances required redesign, indicating better initial success rates.



**Fig 7: Comparison Graph between Random Forest and SVM**

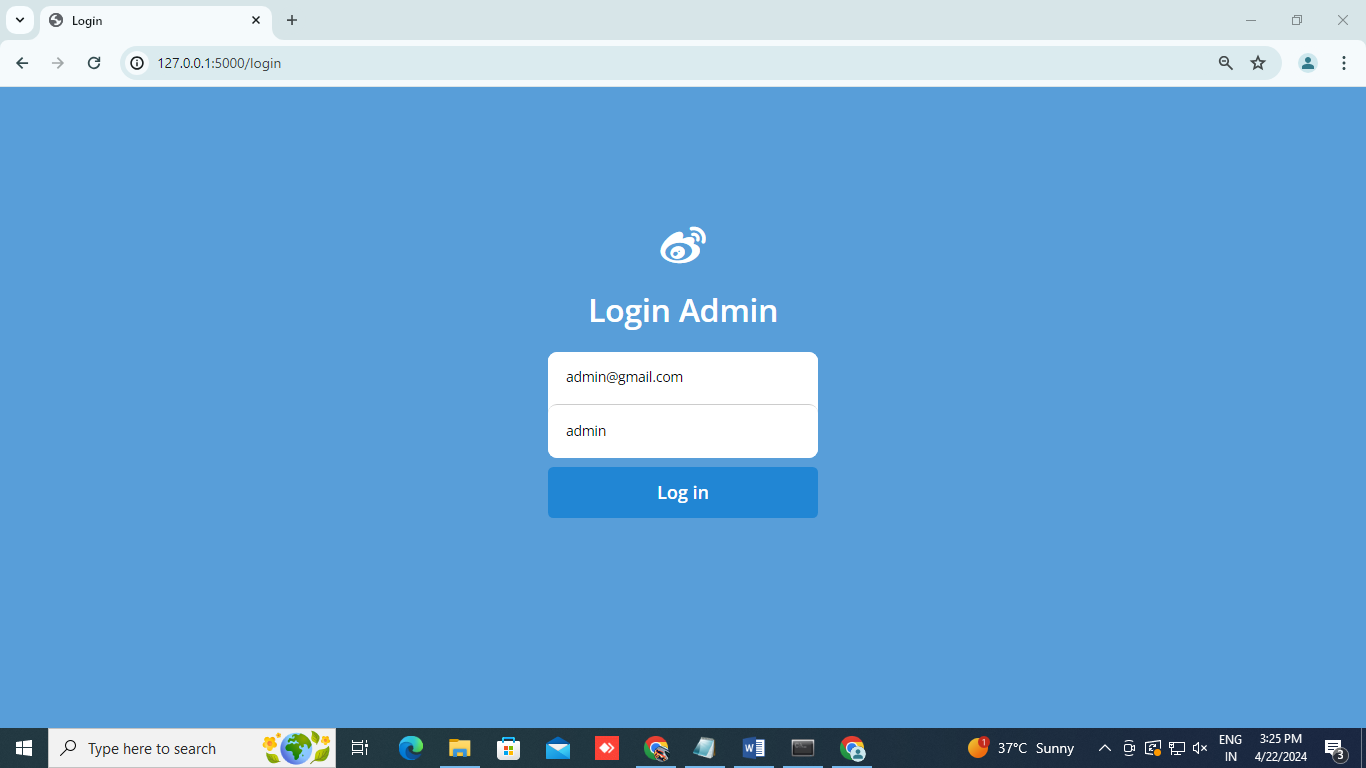
The graph highlights that Random Forest consistently outperforms SVM in all four evaluation metrics, indicating its superior predictive power in software defect estimation. Accuracy and Recall show the most noticeable gap, suggesting that Random Forest is better at correctly identifying both defective and non-defective instances. While SVM performs steadily, its lower Precision and F1-Score suggest that it may struggle with classification balance compared to Random Forest. The overall trend reinforces Random Forest as the more effective model for this task.

**OUTPUT**



**Fig 8: Home page**

Fig 8 shows the Home Page where the User can click Login to use the application.



**Fig 9: Login Page**

Fig 9 shows the Login Page where the User can Login into the application using their id and password.

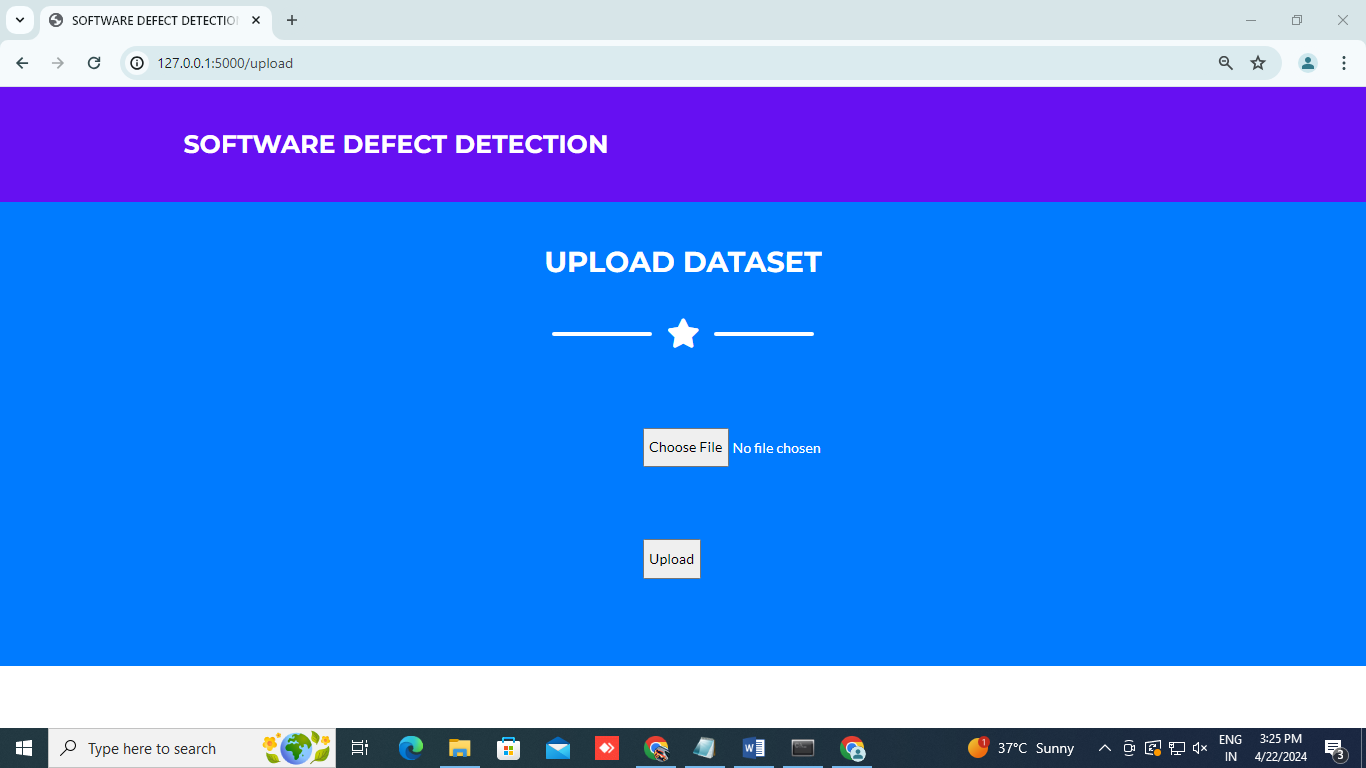
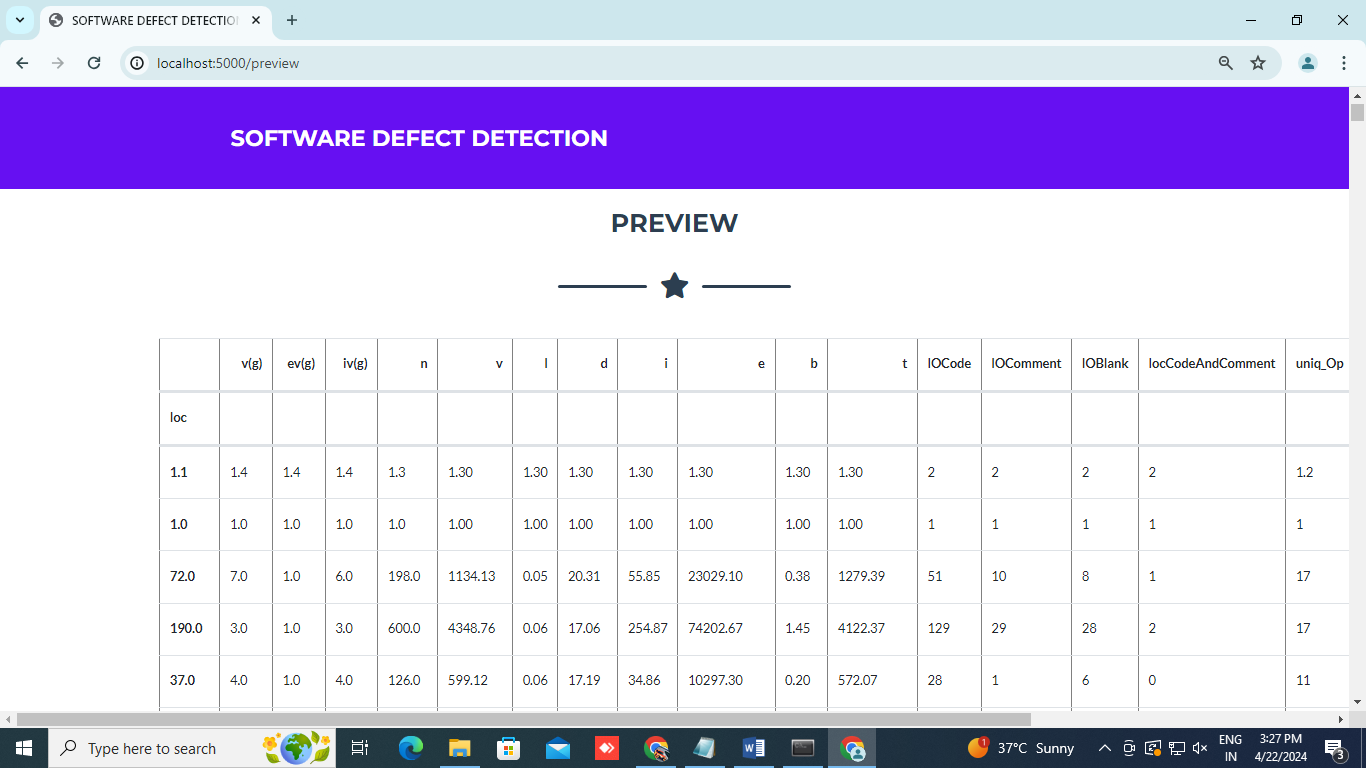
**Fig 10: Upload Page**

Fig 10 shows the Upload Page where the User can choose the dataset file and upload it.



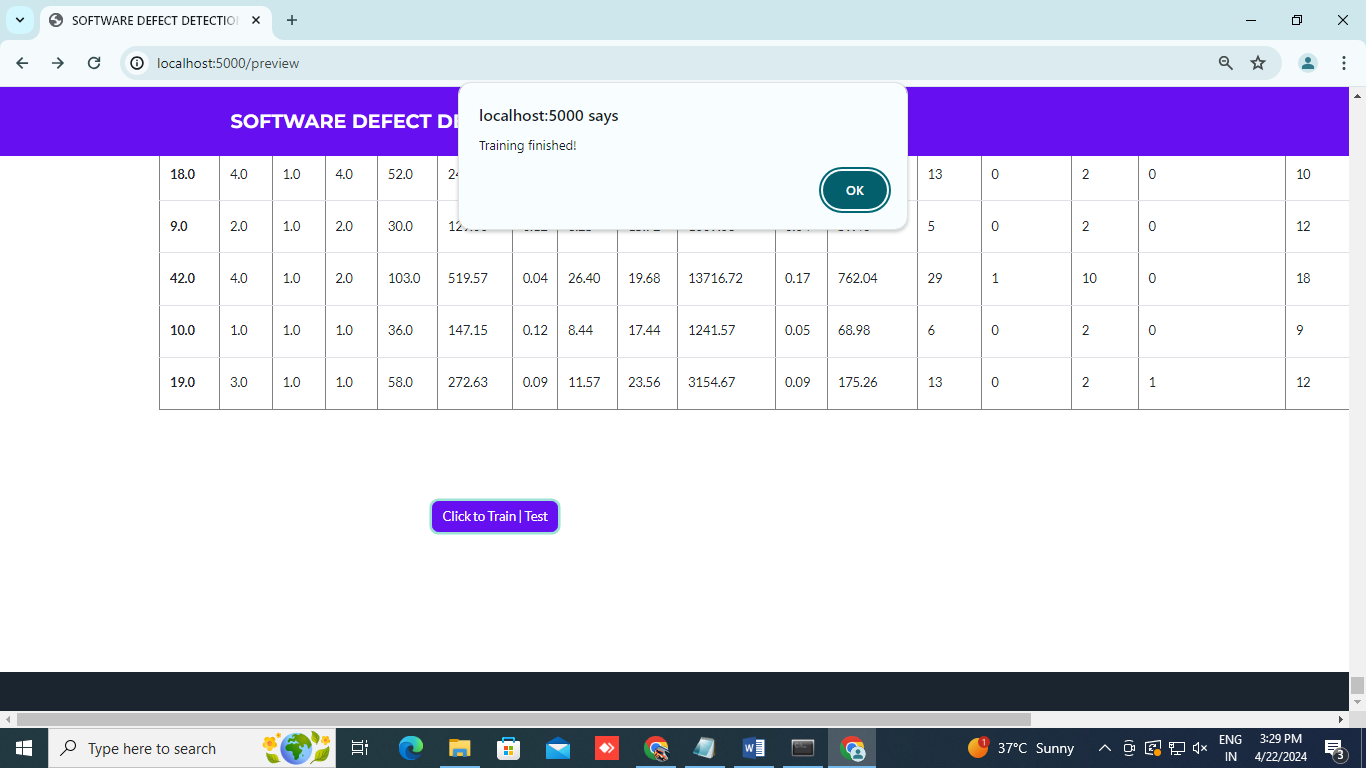
**Fig 11: Preview Page**

Fig 11 shows the Preview Page where the User can see the preview of the Dataset they uploaded.



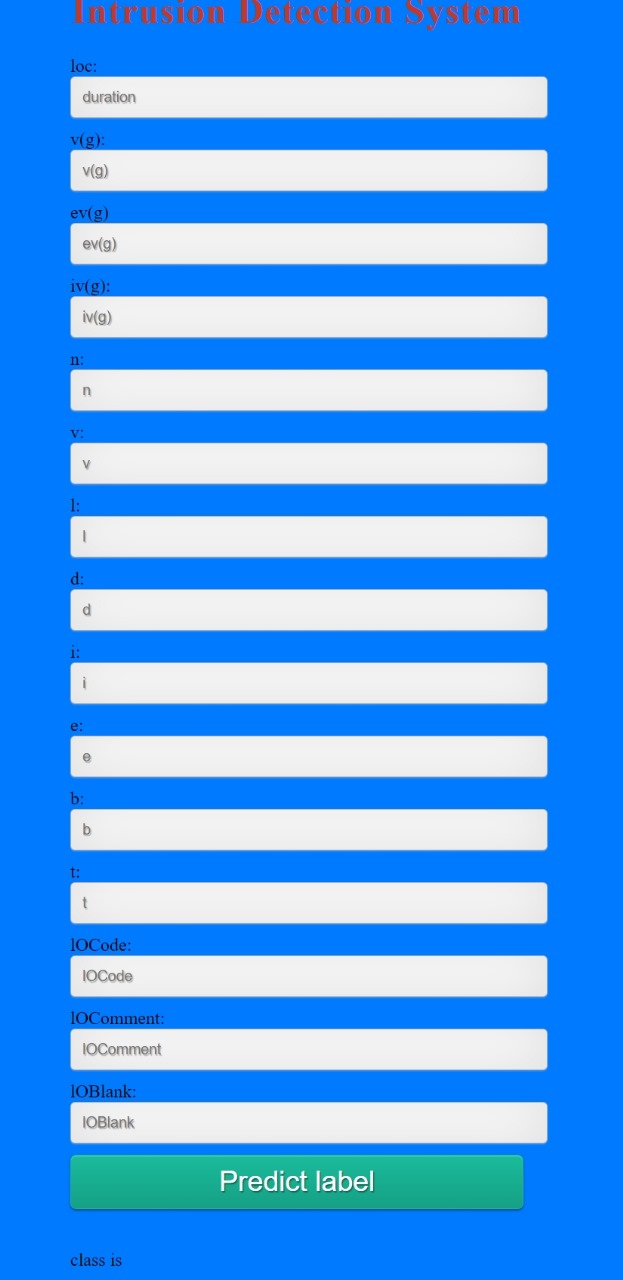
**Fig 12: Train/Test Page**

Fig 12 shows the Train/Test Page where the User click to train or test the uploaded dataset.



**Fig 13: Training Complete Page**

Fig 13 shows the Training Complete Page where the User gets notified that training is completed.



**Fig 14: Predict Label Page**

Fig 14 shows the Predict Label Page where the User can give the inputs and estimate the software defect by clicking the Predict Label button.

**CONCLUSION**

Software illness prediction is the most critical software improvement approach that ought to be applied with utmost interest. In this work, a double linear talent is made to deduct defects speedier and as it should be. This approach is based totally on a double linear evaluation technique primarily based on the F-score selection technique, that is used to pick key characteristics that help predict defects in software program modules.

There is a great distinction in overall performance for a classifier designed with a new set of capabilities in comparison to a classifier constructed with a full set of capabilities. This take a look at shows the effectiveness of a random soar approach primarily based on feature selection in predicting defective software modules and suggests that the proposed version may be beneficial for predicting the nice of programming.

**REFERENCES**

**1.**B.W.Boehm and P.N.Papaccio (1988), Understanding and controlling software costs’, IEEE Tran, Software eng., vol. 14, pp. 1462-1477.

2. J.Zheng (2010), ’Cost-sensitive boosting neural networks for software defect prediction’, Expert Systems with Appl., vol.37, pp. 4537-4543.

3. L.C.Briand, K.E.Emam, et al (2000), ‘A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content’, IEEE Transactions on Software Engineering, Vol. 26, pp.518-540.

4. Lourdes Pelayo and Scott Dick (2007), ‘Applying Novel Resembling Strategies to Software Defect Prediction,’ in proc.North Amr.Fuzzy Inf. Processing Society, pp. 69 – 72.

5. Mingxia Liu, Linsong Miao et al (2014), ’TwoStage Cost-Sensitive Learning forSoftware Defect Prediction’, IEEE Trans. Reliability, Vol. 63, pp.679-684.

6. P. Runeson and C. Wohlin (1998), ‘An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections’, Empirical Software Engineering., vol. 3, pp.381–406.

7. Q. Song, Z. Jia et al (2011), ‘A General Software Defect- Proneness Prediction Framework’, IEEE Transactions On Software Engineering, Vol. 37, pp.356-370.

8.JafarAbo Nada; Mohammad Rasmi Al-Mosa, 2018 International Arab Conference on Information Technology (ACIT), A Proposed Wireless Intrusion Detection Prevention and Attack System

9. Kinam Park; Youngrok Song; Yun-Gyung Cheong, 2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigData Service), Classification of Attack Types for Intrusion Detection Systems Using a Machine Learning Algorithm

10. S. Bernard, L. Heutte and S. Adam “On the Selection of Decision Trees in Random Forests” Proceedings of International Joint Conference on Neural Networks, Atlanta, Georgia, USA, June 14-19, 2009, 978-1-4244-3553-1/09/$25.00 ©2009 IEEE

11. A. Tesfahun, D. Lalitha Bhaskari, “Intrusion Detection using Random Forests Classifier with SMOTE and Feature Reduction” 2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies, 978-0-4799-2235-2/13 $26.00 © 2013 IEEE

12. Le, T.-T.-H., Kang, H., & Kim, H. (2019). The Impact of PCA-Scale Improving GRU Performance for Intrusion Detection. 2019 International Conference on Platform Technology and Service (PlatCon). Doi:10.1109/platcon.2019.8668960

13.Anish Halimaa A, Dr K.Sundarakantham: Proceedings of the Third International Conference on Trends in Electronics and Informatics (ICOEI 2019) 978-1-5386-9439-8/19/$31.00 ©2019 IEEE “MACHINE LEARNING BASED INTRUSION DETECTION SYSTEM.

14.F. Padberg, T. Ragg and R. Schoknecht, "Using machine learning for estimating the defect content after an inspection," in IEEE Transactions on Software Engineering, vol. 30, no. 1, pp. 17-28, Jan. 2024, doi: 10.1109/TSE.2004.1265733.

15.S. Tong, Q. He, Y. Chen, Y. Yang and B. Shen, "Heterogeneous Cross-Company Effort Estimation through Transfer Learning," 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), Hamilton, New Zealand, 2020, pp. 169-176, doi: 10.1109/APSEC.2016.033.

16.H. Wei, C. Shan, C. Hu, H. Sun and M. Lei, "Software defect distribution prediction model based on NPE-SVM," in China Communications, vol. 15, no. 5, pp. 173-182, May 2018, doi: 10.1109/CC.2018.8387996.

17.D. Sas and P. Avgeriou, "An Architectural Technical Debt Index Based on Machine Learning and Architectural Smells," in IEEE Transactions on Software Engineering, vol. 49, no. 8, pp. 4169-4195, Aug. 2023, doi: 10.1109/TSE.2023.3286179

18.Ouertani, O. Krini and H. J. Börcsök, "A Practical Approach for Reliability Prediction of Safety Critical Software Using Multi-Model Ensemble Techniques," 2023 7th International Conference on System Reliability and Safety (ICSRS), Bologna, Italy, 2023, pp. 498-506, doi: 10.1109/ICSRS59833.2023.10381372.

19.D. J. Drown, T. M. Khoshgoftaar and N. Seliya, "Evolutionary Sampling and Software Quality Modeling of High-Assurance Systems," in IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 39, no.5, pp. 10971107,Sept.2009,doi:10.1109/TSMCA.2020.2020804.

20.Jing Zhou, Xinhan Huang, Jing Liu and Min Wang, "Automated fault detection in nonlinear systems using an OLA method combined with TAF-MFNN," 2020 7th World Congress on Intelligent Control and Automation, Chongqing, 2008,pp.11651169,doi:10.1109/WCICA.2008.4593088.

21.F. Padberg, T. Ragg and R. Schoknecht, "Using machine learning for estimating the defect content after an inspection," in IEEE Transactions on Software Engineering, vol. 30, no. 1, pp. 17-28, Jan. 2024, doi: 10.1109/TSE.2004.1265733.

22.S. Tong, Q. He, Y. Chen, Y. Yang and B. Shen, "Heterogeneous Cross-Company Effort Estimation through Transfer Learning," 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), Hamilton, New Zealand, 2020, pp. 169-176, doi: 10.1109/APSEC.2016.033.

23.H. Wei, C. Shan, C. Hu, H. Sun and M. Lei, "Software defect distribution prediction model based on NPE-SVM," in China Communications, vol. 15, no. 5, pp. 173-182, May 2018, doi: 10.1109/CC.2018.8387996.

24.D. Sas and P. Avgeriou, "An Architectural Technical Debt Index Based on Machine Learning and Architectural Smells," in IEEE Transactions on Software Engineering, vol. 49, no. 8, pp. 4169-4195, Aug. 2023, doi: 10.1109/TSE.2023.3286179

25.Ouertani, O. Krini and H. J. Börcsök, "A Practical Approach for Reliability Prediction of Safety Critical Software Using Multi-Model Ensemble Techniques," 2023 7th International Conference on System Reliability and Safety (ICSRS), Bologna, Italy, 2023, pp. 498-506, doi: 10.1109/ICSRS59833.2023.10381372.

26.D. J. Drown, T. M. Khoshgoftaar and N. Seliya, "Evolutionary Sampling and Software Quality Modeling of High-Assurance Systems," in IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 39, no. 5, pp. 1097-1107, Sept. 2009, doi: 10.1109/TSMCA.2020.2020804.

27.Jing Zhou, Xinhan Huang, Jing Liu and Min Wang, "Automated fault detection in nonlinear systems using an OLA method combined with TAF-MFNN," 2020 7th World Congress on Intelligent Control and Automation, Chongqing, 2008,pp.11651169,doi:10.1109/WCICA.2008.4593088.

28.F. Padberg, T. Ragg and R. Schoknecht, "Using machine learning for estimating the defect content after an inspection," in IEEE Transactions on Software Engineering, vol. 30, no. 1, pp. 17-28, Jan. 2024, doi: 10.1109/TSE.2004.1265733.

29.S. Tong, Q. He, Y. Chen, Y. Yang and B. Shen, "Heterogeneous Cross-Company Effort Estimation through Transfer Learning," 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), Hamilton, New Zealand, 2020, pp. 169-176, doi: 10.1109/APSEC.2016.033.

30.H. Wei, C. Shan, C. Hu, H. Sun and M. Lei, "Software defect distribution prediction model based on NPE-SVM," in China Communications, vol. 15, no. 5, pp. 173-182, May 2018, doi: 10.1109/CC.2018.8387996.

31.D.Sas and P. Avgeriou, &quot;An Architectural Technical

Debt Index Based on Machine Learning and ArchitecturalSmells,&quot; in IEEE Transactions on Software Engineering,vol. 49, no. 8, pp. 4169-4195, Aug. 2023, doi:10.1109/TSE.2023.3286179

32.Ouertani, O. Krini and H. J. Börcsök, &quot;A Practical Approach for Reliability Prediction of Safety Critical Software Using Multi-Model Ensemble Techniques,&quot; 2023 7th International Conference on System Reliability and Safety (ICSRS), Bologna, Italy, 2023, pp. 498-506, doi:10.1109/ICSRS59833.2023.10381372.

33.D. J. Drown, T. M. Khoshgoftaar and N. Seliya,

&quot;Evolutionary Sampling and Software Quality Modeling of High-Assurance Systems,&quot; in IEEE Transactions on

Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 39, no. 5, pp. 1097-1107, Sept. 2009, doi:10.1109/TSMCA.2020.2020804.

34.Jing Zhou, Xinhan Huang, Jing Liu and Min Wang,&quot;Automated fault detection in nonlinear systems using anOLA method combined with TAF-MFNN,&quot; 2020 7th World

Congress on Intelligent Control and Automation,

Chongqing, 2008, pp. 1165-1169, doi:

10.1109/WCICA.2008.4593088