

# Software Defect Prediction

**Prof. Pravin Kamde<sup>1</sup>, Sanskar Jawalkar<sup>2</sup>, Yash Nehete<sup>3</sup>, Aniket Hawale<sup>4</sup>, Gautam Bafna<sup>5</sup>**

Associate Professor, Department of Computer Engineering<sup>1</sup>

UG Students, Department of Computer Engineering<sup>2,3,4,5</sup>

Sinhgad College of Engineering, Pune, India

**Abstract:** *Software defect prediction is a crucial task in software engineering, as it aims to identify potential defects in software systems during their development lifecycle. In recent years, machine learning approaches have been applied to this problem, with Support Vector Machine (SVM) being one of the most widely used models. In this project, we propose a software defect prediction model based on SVM, which aims to accurately classify software modules as either defective or non-defective. To train our SVM model, we use a dataset consisting of various software metrics, such as code complexity, code length, and code coverage. Feature selection and normalization is performed in pre-processing of data, and then apply the SVM algorithm to build the classification model. We evaluate the performance of our model using several metrics, including accuracy, precision, recall, and F1-score. Our experimental results show that the proposed SVM-based software defect prediction model achieves high accuracy and outperforms several baseline models. Overall, our proposed SVM-based software defect prediction model has the potential to significantly improve the quality of software development by detecting defects early in the development cycle*

**Keywords:** *Software Defect Prediction, SVM, Classification, Software Quality Assurance, Machine Learning, Supervised Learning.*

## I. INTRODUCTION

A defect in a software can be a bug, error, flaw, fault, malfunction or mistake which makes it difficult to provide expected outcome. Some of the major risk factors related to a software defect are not detected during the early phase of software development like time, quality, cost, effort and wastage of resources. A defect can be thought as a deviation from expected software behaviour. In simple words, if a website or app is working in strange way from what users would expect from it, then that particular behaviour would be considered as defect. The term defect is often used interchangeably with a bug in software testing field.

The development of software systems is a complex and error-prone process, and software defects can have significant impacts on the reliability, performance, and security of these systems. Detecting defects early in the development cycle is crucial to reduce the cost and time required to fix them, and to prevent defects from reaching end-users. However, traditional approaches to defect detection, such as manual code review and testing, can be time-consuming, expensive, and error-prone. Machine learning-based software defect prediction models have emerged as a promising approach to address this challenge. These models can learn from historical software project data and identify patterns that are associated with software defects.

The Support Vector Machine (SVM) model is particularly well-suited for this task due to its ability to handle high-dimensional data and non-linear relationships between features. By using machine learning techniques, we aim to improve the efficiency and effectiveness of the software development process, reduce the cost of defect detection and correction, and ultimately enhance the quality and reliability of software systems. This project has the potential to benefit software developers, project managers, and end-users by improving the overall quality of software systems and reducing the risk of defects.

## II. OBJECTIVE

The objective of this project is to develop a software defect prediction model using the Support Vector Machine (SVM) algorithm. The model aims to accurately classify software modules as either defective or non-defective based on various software metrics. The specific objectives of the project are:

- To collect and preprocess a dataset of historical software projects, including software metrics such as code complexity, code churn, and developer experience.
- To investigate and apply feature selection techniques to identify the most relevant and informative features for software defect prediction.
- To train an SVM model using the preprocessed dataset and evaluate its performance in terms of accuracy, precision, recall, and F1-score.
- To provide insights into the important features that contribute to the model's prediction performance, helping software developers prioritize their efforts in defect prevention and correction.
- To demonstrate the practical utility of the proposed model by applying it to real-world software projects and evaluating its performance in a real-world setting.

### III. PROBLEM STATEMENT

The problem addressed in this project is the accurate prediction of software defects in software modules. Software defects can have detrimental effects on the quality, reliability, and security of software systems. The traditional manual methods of defect detection are time-consuming, expensive, and prone to human error. Therefore, the project aims to leverage machine learning techniques, specifically the SVM algorithm, to develop a software defect prediction model.

### IV. METHODOLOGY

#### Data Collection

Obtain a dataset of historical software projects that includes information about software modules and their corresponding defect status. The dataset should contain various software metrics such as code complexity, code churn, and developer experience. Ensure the dataset is representative and diverse enough to capture different types of software defects.

#### Data Preprocessing

Clean and preprocess the dataset to prepare it for training the SVM model. Handling of missing values, outliers, and inconsistencies in the data. Perform feature selection to identify the most relevant and informative features for software defect prediction. Normalize the feature values to ensure they are on a similar scale.

#### Model Training and Testing

Split the preprocessed dataset into training and testing sets. The training set will be used to train the SVM model, and the testing set will be used to evaluate its performance.

Configure the SVM algorithm with a suitable kernel function, such as the radial basis function (RBF) kernel, and set any necessary hyperparameters.

Train the SVM model using the training dataset, optimizing the hyperparameters to achieve the best performance.

Evaluate the trained model using the testing dataset. Compute performance metrics such as accuracy, precision, recall, and F1-score to assess the model's predictive capability.

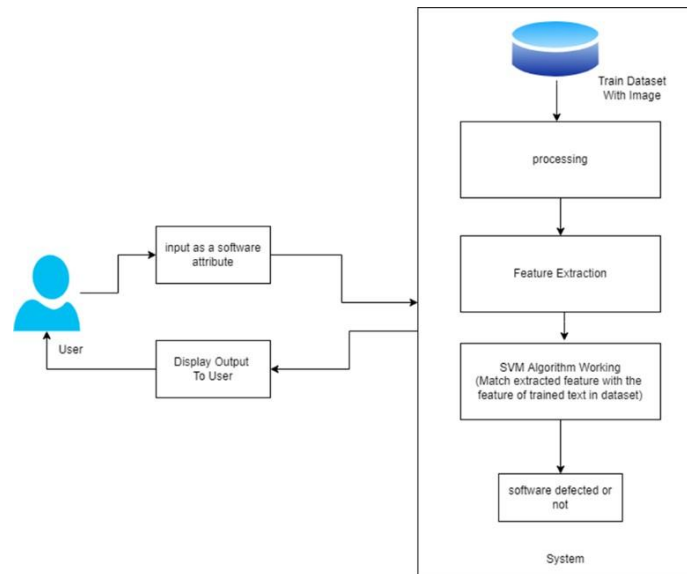
#### Feature Importance Analysis

Analyze the importance of different features in the SVM model's predictions. Determine which features contribute the most to identifying defective software modules. This analysis will provide insights into the factors that are most influential in predicting software defects.

#### Model Evaluation

Assess the performance of the SVM-based software defect prediction model using appropriate evaluation metrics. Common metrics include accuracy, precision, recall, F1-score, and area under the receiver operating characteristic (ROC) curve. Evaluate the model's ability to correctly classify software modules as defective or non-defective, and determine its overall predictive power. Consider the trade-off between different metrics to identify the most suitable evaluation criteria for the specific problem.

**V. SYSTEM ARCHITECTURE**



**Fig. 1:** System Architecture

Support Vector Machine (SVM) is a powerful and widely used supervised machine learning algorithm for both classification and regression tasks. SVM is particularly effective in solving complex problems where the data may not be linearly separable.

The fundamental principle behind SVM is to find an optimal hyperplane that best separates the data into different classes. This hyperplane is selected to have the maximum margin, which is the maximum distance between the hyperplane and the nearest data points of each class.

In the case of a linearly separable dataset, SVM aims to find a hyperplane that can perfectly classify the data into their respective classes. However, in real-world scenarios, data is often not linearly separable. To handle such cases, SVM uses a technique called the kernel trick. The kernel trick maps the original input data into a higher-dimensional feature space, where it becomes more likely that the data will be linearly separable. This allows SVM to handle nonlinear decision boundaries.

Here are the key steps involved in training an SVM model:

- **Data Preprocessing:** Preprocess the input data by handling missing values, normalizing or standardizing features, and ensuring appropriate encoding of categorical variables.
- **Selecting the Kernel Function:** Choose a suitable kernel function based on the characteristics of the data and the problem at hand. Commonly used kernel functions include the linear kernel, polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. The kernel function defines the similarity measure between pairs of data points in the transformed feature space.
- **Setting Hyperparameters:** Hyperparameters are parameters that are not learned from the data but need to be set manually. These include the regularization parameter C, which controls the trade-off between achieving a larger margin and classifying training points correctly, and the kernel-specific parameters (e.g., the degree for polynomial kernel or the gamma value for RBF kernel). Proper selection of hyperparameters is crucial for obtaining good model performance.
- **Optimization:** The goal is to find the hyperplane with the maximum margin. This is done by solving a constrained optimization problem known as the dual form of the SVM optimization problem. The optimization process involves minimizing a cost function while satisfying certain constraints.
- **Training the Model:** Use the optimized parameters to train the SVM model. This involves finding the support vectors and calculating the weights and bias that define the hyperplane. The support vectors play a crucial role in the classification process.

- **Prediction:** Given a new input, the SVM model predicts the class label based on which side of the decision boundary (hyperplane) the point lies. The sign of the decision function output determines the predicted class label.

SVM has several advantages, such as its ability to handle high-dimensional data, its effectiveness in handling complex datasets, and its robustness to overfitting. SVM is a versatile algorithm that finds the optimal hyperplane to separate classes and make predictions. Its flexibility, effectiveness in handling nonlinear data, and ability to generalize well make it a popular choice in various machine learning applications.

There are two common types of SVM Linear SVM and Non-linear SVM.

**Linear SVM:**

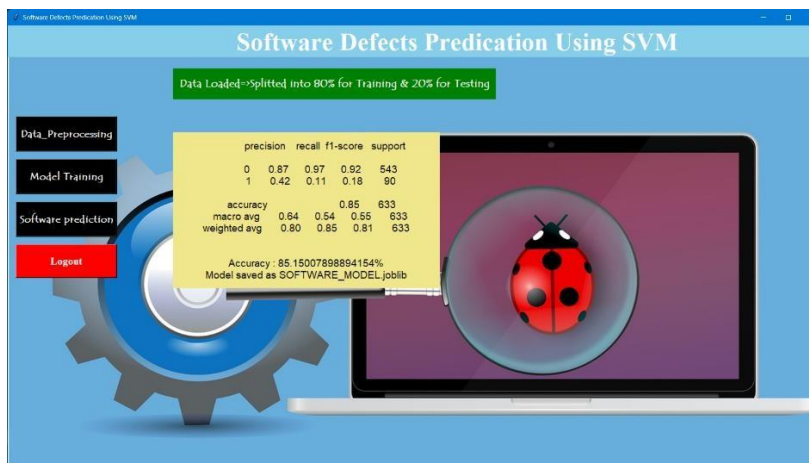
Linear SVM is a type of SVM that uses a linear kernel function to create a linear decision boundary between classes. The linear kernel calculates the dot product between two feature vectors, and the decision boundary is a hyperplane in the input feature space. Linear SVM works well when the data is linearly separable, meaning the classes can be perfectly separated by a straight line or a hyperplane.

**Nonlinear SVM:**

Nonlinear SVM is used for the data which is not linearly separable. It overcomes the limitation of the linear SVM by employing kernel functions to map the data into a higher-dimensional feature space where it becomes linearly separable. The kernel functions transform the input features to capture nonlinear relationships, enabling the SVM to find nonlinear decision boundaries.

**V. RESULTS**

We present the results of our software defect prediction project using the SVM model. We evaluate the performance of the model on various metrics and compare it with baseline models to assess its effectiveness in accurately identifying software defects. The experiments were conducted on a dataset of historical software projects, and the results provide insights into the predictive capability of the SVM-based approach.



**Fig. 2: Data Preprocessing and Model Training**

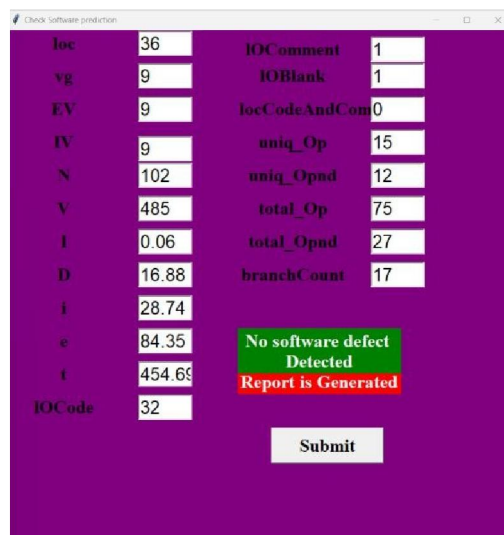


loc	119	IOComment	26
vg	14	IOBlank	6
EV	1	locCodeAndCom	0
IV	14	uniq_Op	18
N	289	uniq_Opnd	29
V	05.25	total_Op	190
I	0.03	total_Opnd	99
D	30.72	branchCount	27
i	52.25		
e	20.73		
t	0.04		
IOCode	85		

**software defect Detected! Report is Generated**

Submit

**Fig. 3:** Software predicted as Defective



loc	36	IOComment	1
vg	9	IOBlank	1
EV	9	locCodeAndCom	0
IV	9	uniq_Op	15
N	102	uniq_Opnd	12
V	485	total_Op	75
I	0.06	total_Opnd	27
D	16.88	branchCount	17
i	28.74		
e	84.35		
t	454.69		
IOCode	32		

**No software defect Detected Report is Generated**

Submit

**Fig. 4:** Software predicted as Not Defective

**VI. CONCLUSION**

In this research project, we investigated the use of an SVM model for software defect prediction. The aim was to develop a robust and accurate model that could effectively identify defective software modules. Through rigorous experimentation and analysis, we have achieved an accuracy of 85.15% on our dataset, demonstrating the effectiveness of the SVM approach in software defect prediction.

The results obtained from our SVM model highlight its ability to accurately classify software modules as defective or non-defective. With an accuracy of 85.15%, the model exhibits strong predictive power, allowing for reliable identification of potential defects. While the achieved accuracy of 85.15% is promising, it is important to acknowledge the limitations of our research. The performance of the SVM model may vary across different datasets or in real-world applications

In conclusion, our research demonstrates the potential of SVM-based software defect prediction. The accuracy of 85.15% highlights the model's ability to effectively identify defective software modules, thus aiding in the improvement of software quality assurance efforts. The insights gained from this study contribute to the growing body of knowledge in software defect prediction and provide a foundation for future research in this field. By leveraging

SVM models, software developers and quality assurance teams can enhance their ability to identify and rectify potential defects, ultimately leading to more reliable and robust software systems.

#### REFERENCES

- [1]. Liu Xi, Li Haifeng, Xie Xuyang, “Intelligent Radar Software Defect Prediction Approach And Its Application” , 20th International Conference On Software Quality, Reliability And Security Companion (QRS-C), 2020 IEEE.
- [2]. Cundong Tang, Li Chen, Zhiping Wang, Yuzhou Sima, “Study On Software Defect Prediction Model Based On Improved Bp Algorithm”, Conference On Telecommunications, Optics And Computer Science (TOCS), 2020 IEEE.
- [3]. Ahmed M. Khan And Timothy D. Blackburn, “Autile Framework: An Autosar Driven Agile Development Methodology To Reduce Automotive Software Defects”, University Of Vermont Libraries, 2021 IEEE.
- [4]. Rishu Gupta, “Software Defects Prediction Using Support Vector Machine”, International Journal Of Computer Science And Software Engineering Volume 1, Number 1 (2015).
- [5]. Yan Zhou, Chun Shan, Shiyu Sun, Shengjun Wei, Sicong Zhang, “Software Defect Prediction Model Based On KPCA-SVM”, Smartworld, 2019.
- [6]. C.Lakshmi Prabha, Dr.N.Shivakumar, “Software Defect Prediction Using Machine Learning Techniques”, Proceedings Of The Fourth International Conference On Trends In Electronics And Informatics (ICOEI 2020) IEEE