Software Defect Prediction Using Machine Learning Techniques

Under PMKVY 4.0 AI-ML Engineer

Manjari Kumari
CAN_26503909

Under Supervision of
Prof. Durga Prasad Mohapatra

Department of Computer Science & Engineering
NIT, Rourkela

March 8, 2024

DECLARATION

I here-by declare that the project work entitled *"Software Defect Prediction Using Machine Learning Techniques"*, submitted to the National Institute of Technology, Rourkela, is a record of an original work done by me under the guidance of my Project Guide, *Prof. Durga Prasad Mohapatra*, Department of *Computer Science and Engineering*, and this project work has not performed the basis for the award of any Degree or diploma / associate ship / fellowship and similar project if any.

_____ _____

(Manjari Kumari) (Prof. Durga Prasad Mohapatra)
Date: March 8, 2024 Date: March 8, 2024

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who made it possible for me to successfully complete this project. I am particularly indebted to our project supervisor for the final year, ***Professor Durga Prasad Mohapatra***, whose invaluable suggestions and unwavering support played a pivotal role in guiding me through the process of writing this report.

In addition, I wish to extend my heartfelt appreciation to the dedicated personnel of the ***Department of Computer Science and Engineering*** for granting me access to the essential equipment and materials necessary to carry out this project. Special thanks are due to my advisor, ***Biswajit Prasad Gond*** *and* ***Mohit Rathore***, whose guidance was instrumental in meeting the project's objectives within the given timeframe.

Lastly, I would like to convey my profound gratitude to my parents and friends for their continuous encouragement and support throughout my academic journey and the development of this project. I am also deeply thankful to the *National Institute of Technology, Rourkela* for their backing and for affording me the opportunity to undertake this project, along with the provision of essential facilities.

_____

(Manjari Kumari)
Date: March 8, 2024

Contents

1 Introduction

Software defect prediction is a crucial area of research and practice in software engineering aimed at identifying potential defects in software systems early in the development process [1]. By analyzing historical data and patterns, software defect prediction models can help software developers prioritize testing efforts, allocate resources efficiently, and improve overall software quality.

The primary goal of software defect prediction is to reduce the number of defects that escape into production, thus minimizing the impact on users and reducing maintenance costs. Various machine learning and data mining techniques are used in software defect prediction, including logistic regression, decision trees, random forests, [3] and neural networks. These models are trained on historical software metrics, such as code complexity, churn, and size, to predict the likelihood of a module containing defects.

Software defect prediction has applications in both industry and academia. In industry, it helps software development teams focus their testing efforts on high-risk areas, improving the overall quality of the software. In academia, researchers study and develop new techniques to enhance the effectiveness of defect prediction models. Overall, software defect prediction plays a vital role in improving software reliability [15] and reducing development costs.

1.1 Importance of defect prediction in software development.

Defect prediction plays a crucial role in software development for several reasons:

- **Early Identification of Risks:** Predicting defects early helps address issues before they escalate, minimizing impact on schedules and budgets.
- **Resource Allocation:** Allows for more effective resource allocation, focusing efforts on the most defect-prone areas.
- **Improved Software Quality:** Identifying and addressing defects before release leads to higher quality software and greater customer satisfaction.
- **Cost Reduction:** Detecting and fixing defects early is cheaper than addressing them later, reducing overall development costs.
- **Enhanced Productivity:** Prioritizing efforts based on defect prediction increases productivity and software reliability.
- **Risk Management:** Provides insights into quality-related risks, aiding in effective risk mitigation.
- **Continuous Improvement:** Facilitates a culture of continuous improvement by providing feedback on development practices and strategies.

1.2 Goals and Objectives of the Project

In this project, we aim to gain a deeper understanding of software defects, machine learning techniques, and algorithms such as Support Vector Machines (SVM), Random Forests (RF), Decision Trees (DT), and K-Nearest Neighbors (KNN). Additionally, we seek to enhance our

skills in data visualization, data preprocessing, and measuring algorithm performance using evaluation metrics and confusion matrices.

1. **Data Preprocessing:** The goal is to clean and prepare the dataset for analysis. This includes handling missing data, dealing with outliers, and converting categorical variables into a format suitable for machine learning algorithms.
2. **Data Visualization:** The objective is to create visual representations of the data to gain insights and identify patterns. This can involve using plots, charts, and graphs to illustrate key findings.
3. **Software defect prediction using Machine Learning Techniques:** We aim to apply various machine learning algorithms such as SVM, RF, DT, and KNN to predict software defects. This includes training the models, tuning hyperparameters, and evaluating their performance.
4. **Algorithm Performance:** The goal is to measure the performance of the machine learning models using evaluation metrics such as accuracy, precision, recall, and F1 score. Confusion matrices will also be used to visualize the performance of the classifiers.

2 Data Preprocessing

Data preprocessing is a critical step in the data analysis and machine learning pipeline, involving cleaning, transforming, and preparing the raw data into a format suitable for analysis or model training. The primary goal of data preprocessing is to enhance the quality and usability of the data, making it more suitable for downstream tasks such as modeling, visualization, or statistical analysis.

Here are some common techniques involved in data preprocessing:

1. **Data Cleaning:** This step involves identifying and handling missing values, outliers, and errors in the data. Missing values can be imputed using techniques such as mean, median, or mode imputation, or by using more advanced methods like predictive modeling. Outliers can be detected and treated using statistical methods or domain-specific knowledge.
2. **Data Transformation:** Data transformation techniques are applied to normalize or standardize the data distribution, making it more suitable for analysis or modeling. Common transformations include scaling features to a specific range, applying logarithmic or exponential transformations, or converting categorical variables into numerical representations through techniques like one-hot encoding or label encoding.
3. **Feature Selection and Dimensionality Reduction:** In some cases, datasets may contain a large number of features, some of which may be redundant or irrelevant for the analysis or modeling task. Feature selection techniques are used to identify the most relevant features that contribute most to the target variable. Dimensionality reduction techniques like principal component analysis (PCA) or t-distributed stochastic neighbor embedding (t-SNE) can also be applied to reduce the number of features while preserving important information.
4. **Data Integration:** Data integration involves combining data from multiple sources into a unified dataset. This may require resolving inconsistencies in data formats, units, or

identifiers across different sources. Data integration ensures that all relevant information is available for analysis or modeling.

5. **Data Normalization:** Data normalization is the process of scaling numeric features to a standard range, typically between 0 and 1 or -1 and 1. Normalization helps prevent features with larger scales from dominating the analysis or modeling process and ensures that each feature contributes proportionally to the final results.
6. **Handling Categorical Data:** Categorical variables, such as gender or product categories, need to be encoded into numerical representations before they can be used in machine learning models. Techniques like one-hot encoding or label encoding are commonly used to convert categorical variables into a format suitable for analysis or modeling.

Overall, data preprocessing is an essential step in the data analysis and modeling process, as it helps ensure the quality, consistency, and usability of the data for downstream tasks. Effective data preprocessing can significantly improve the performance and reliability of analytical models and insights derived from the data.

2.1 Data collection and source.

We have obtained our dataset for software defect prediction from NASA website using the provided [`https://figshare.com/collections/NASA_MDP_Software_Defects_Data_Sets/4054940/1`]

2.2 Data cleaning (handling missing values and outliers).

Data cleaning is the process of identifying and correcting errors, inconsistencies, and anomalies in a dataset to ensure its accuracy, reliability, and usability for analysis or modeling. Handling missing values and outliers are two important tasks within the data cleaning process:

Handling Missing Values

1. **Identification:** The first step is to identify missing values within the dataset. Missing values can occur due to various reasons, such as data entry errors, equipment failures, or non-responses in surveys.
2. **Imputation:** There are several methods for handling missing values, including:
   Mean, median, or mode imputation: Replace missing values with the mean, median, or mode of the respective feature.
3. **Forward or backward fill:** Use the value from the previous or next observation to fill in missing values.
4. **Interpolation:** Estimate missing values based on the values of neighboring observations using techniques like linear interpolation or spline interpolation.
5. **Predictive modeling:** Use machine learning algorithms to predict missing values based on other features in the dataset.
6. **Deletion:** In some cases, it may be appropriate to remove observations or features with a high proportion of missing values if they cannot be effectively imputed.

Handling Outliers

1. **Identification:** Outliers are data points that deviate significantly from the rest of the dataset and may represent errors, anomalies, or rare events. Various statistical techniques, such as z-score analysis, box plots, or scatter plots, can be used to identify outliers.
2. **Treatment:** Depending on the nature of the outliers and the specific requirements of the analysis, outliers can be treated in different ways:
3. **Correction or removal:** If outliers are due to data entry errors or measurement mistakes, they can be corrected or removed from the dataset.
4. **Transformation:** Transforming the data using techniques like log transformation or winsorization can reduce the impact of outliers on statistical analysis or modeling.
5. **Capping or flooring:** Set a threshold beyond which values are considered outliers and cap or floor them to the threshold value.
6. **Domain Knowledge:** It's essential to consider domain knowledge and context when handling outliers. Some outliers may represent genuine observations that provide valuable insights, while others may indicate errors that need to be addressed.

Effective data cleaning ensures that the dataset is free from errors and inconsistencies, allowing for more accurate and reliable analysis, modeling, and decision-making. It is an iterative process that may involve multiple steps and techniques to achieve the desired level of data quality.

2.3 Features selection and extraction.

Feature selection [16] and feature extraction [17] are two important techniques used in machine learning and data analysis to reduce the dimensionality of the dataset and improve the performance of predictive models. Here's an explanation of each:

1. Feature Selection

Feature selection is the process of selecting a subset of the most relevant features (variables or attributes) from the original dataset while discarding irrelevant or redundant features. The main goals of feature selection are to improve model performance, reduce overfitting, and decrease computational complexity. There are several techniques for feature selection, including:

- **Filter methods:** These methods evaluate the relevance of features based on statistical metrics such as correlation, mutual information, or significance tests, and select the top-ranked features.
- **Wrapper methods:** These methods involve evaluating the performance of a model trained on different subsets of features and selecting the subset that yields the best performance based on a specific evaluation metric, such as accuracy or cross-validation error.
- **Embedded methods:** These methods integrate feature selection into the model training process, where the importance of features is automatically determined during model training. Examples include decision trees, random forests, and LASSO (Least Absolute Shrinkage and Selection Operator) regularization.

Feature selection helps improve model interpretability by focusing on the most relevant features and removing noise from the dataset.

2. Feature Extraction

- Feature extraction is the process of transforming the original features into a new set of features that capture the essential information contained in the original data while reducing its dimensionality.Unlike feature selection, which selects existing features from the dataset, feature extraction creates new features by applying mathematical transformations or algorithms to the original data.Principal Component Analysis (PCA) is a widely used technique for feature extraction. PCA identifies the directions (principal components) in which the data varies the most and projects the original data onto these components, effectively reducing its dimensionality while preserving as much variance as possible.Other techniques for feature extraction include linear discriminant analysis (LDA), kernel PCA, and autoencoders.Feature extraction is particularly useful when dealing with high-dimensional data or when the original features are not directly suitable for modeling (e.g., text data or image data).

Both feature selection and feature extraction are essential preprocessing steps in machine learning and data analysis pipelines, helping to improve model performance, reduce overfitting, and enhance interpretability. The choice between feature selection and feature extraction depends on the specific characteristics of the dataset and the modeling task at hand.

2.4 Data balancing

Data balancing refers to the process of addressing class imbalance in a dataset, particularly in the context of classification tasks in machine learning. Class imbalance occurs when one class (or category) of the target variable is significantly more prevalent than others. This imbalance can lead to biased model performance, where the model may be more accurate in predicting the majority class but performs poorly on minority classes.

In data balancing, the goal is to adjust the distribution of the target variable to ensure that each class is represented more equally in the dataset. There are several techniques for data balancing:

1. Resampling Techniques

- **Undersampling:** Undersampling involves randomly selecting a subset of samples from the majority class to match the size of the minority class. This reduces the imbalance by decreasing the prevalence of the majority class.
- **Oversampling:** Oversampling involves generating synthetic samples for the minority class to increase its representation in the dataset. Techniques like SMOTE (Synthetic Minority Over-sampling Technique) [18] create new synthetic samples by interpolating between existing minority class samples.
- **Combining Over- and Undersampling:** Some approaches combine oversampling of the minority class with undersampling of the majority class to achieve a more balanced distribution.
- **Cost-Sensitive Learning**

Cost-sensitive learning assigns different costs or weights to different classes during model training to account for class imbalance. Models are penalized more for misclassifying minority class samples than majority class samples, thereby giving equal importance to all classes.

2. Ensemble Methods

Ensemble methods like boosting and bagging can be effective for handling class imbalance. Algorithms like AdaBoost and XGBoost can adjust their weights to focus more on misclassified samples from the minority class, improving their predictive performance on imbalanced datasets.

3. Anomaly Detection

In some cases, the minority class may represent anomalies or rare events. Anomaly detection techniques can be used to identify and handle these instances separately from the majority class during model training.

Data balancing is crucial for ensuring that machine learning models generalize well to all classes and produce unbiased predictions. However, it's essential to carefully select the appropriate balancing technique based on the characteristics of the dataset and the specific requirements of the modeling task. Additionally, it's important to evaluate model performance using appropriate metrics that account for class imbalance, such as precision, recall, F1-score, or area under the ROC curve (AUC-ROC).

2.5 Data Preprocessing

- Data Collection: Gather historical data related to software development, including features (e.g., code complexity, churn, developer experience) and labels indicating whether each software artifact contains defects.
- Data Cleaning: Handle missing values, outliers, and inconsistencies in the data. This may involve imputation, outlier detection, and data transformation.
- Feature Selection and Extraction: Select relevant features and extract meaningful information from raw data. Techniques such as correlation analysis, feature importance, and dimensionality reduction may be applied. Feature we have used in Figure 1 after ❶ prepossessing are as follow:

    1. **Lines of Code (LOC) Blank** - This metric counts the number of lines in a software program that are empty or contain only whitespace characters (such as spaces or tabs). It helps in measuring the overall size and complexity of a codebase.
    2. **Branch Count** - Branch Count represents the number of decision points (branches) in the control flow of a program. It is useful for analyzing the complexity of the program's logic and assessing its test coverage.
    3. **LOC Code and Comment** - This metric calculates the total number of lines in a program that contain both executable code and comments. It is valuable for evaluating the readability, maintainability, and documentation level of the codebase.

4. **LOC Comments** - LOC Comments represents the number of lines in a program that are dedicated solely to comments. Comments are essential for documenting code, explaining its functionality, and enhancing collaboration among developers.
5. **Cyclomatic Complexity** - Cyclomatic Complexity is a software metric used to measure the complexity of a program by counting the number of linearly independent paths through its source code. It helps in identifying areas of code that may be difficult to test or maintain due to their complexity.
6. **Design Complexity** - Design Complexity refers to the level of complexity inherent in the overall design of a software system. Managing design complexity is crucial for ensuring scalability, maintainability, and extensibility of the software.
7. **Essential Complexity** - Essential Complexity is a measure of the inherent complexity of a problem that a software system is designed to solve. Understanding essential complexity helps in designing more efficient and effective software solutions.
8. **LOC Executable** - LOC Executable represents the number of lines in a program that contain executable code. It is a fundamental metric for assessing the size and functionality of a software system.
9. **Halstead Content** - Halstead Content is a set of software metrics introduced by Maurice Halstead to quantify various aspects of a program's source code, such as its length, vocabulary, and volume.
10. **Halstead Difficulty** - Halstead Difficulty is a metric that measures the difficulty of understanding a program's source code. It helps in estimating the cognitive effort required by developers to work with the code.
11. **Halstead Effort** - Halstead Effort is a metric that quantifies the total effort required to develop and maintain a software system. It provides insights into the overall complexity and cost of the software project.
12. **Halstead Error Estimate** - Halstead Error Estimate is a metric that provides an estimate of the number of errors present in a software system based on its size and complexity. It helps in identifying potential areas of the code that may require more thorough testing and debugging.
13. **Halstead Length** - Halstead Length is a metric that measures the total number of tokens (operators and operands) used in a program's source code. It provides insights into the overall size and complexity of the codebase.
14. **Halstead Level** - Halstead Level is a metric that indicates the level of abstraction present in a program's source code. It helps in assessing its readability and maintainability.
15. **Halstead Program Time** - Halstead Program Time is a metric that estimates the time required to write and debug a program based on its size and complexity. It helps in project planning and scheduling.
16. **Halstead Volume** - Halstead Volume is a metric that quantifies the size and complexity of a program's source code based on the number of tokens used. It provides insights into the overall effort required to understand, develop, and maintain the software.
17. **Num Operands** - Num Operands refers to the total number of operands used in a program's source code. It is a basic metric for assessing the complexity and functionality of the software system.

18. **Num Operators** - Num Operators represents the total number of operators used in a program's source code. It provides insights into the computational complexity and functionality of the software system.
19. **Num Unique Operands** - Num Unique Operands refers to the total number of unique operands used in a program's source code. It helps in identifying the diversity and complexity of the data structures and variables employed.
20. **Num Unique Operators** - Num Unique Operators represents the total number of unique operators used in a program's source code. It provides insights into the variety and complexity of the computational operations performed.
21. **LOC Total** - LOC Total is the sum of all lines of code in a software program, including both executable code and comments. It is a fundamental metric for assessing the overall size and complexity of the codebase.
22. **Defective** - Indicates whether a particular software component is considered defective or not. In software engineering, defect prediction models are often built using various metrics to identify potentially defective code areas early in the development process, facilitating quality assurance and testing efforts.

2.5.1 Model Selection

- Choose appropriate machine learning algorithms for defect prediction, considering factors such as the nature of the data, interpretability requirements, and computational constraints. This may involve experimenting with multiple algorithms and comparing their performance.

2.5.2 Model Training

- Split the data into training and testing/validation sets to evaluate the performance of the model.
- Train the selected machine learning model(s) on the training data using appropriate hyperparameters.
- Optionally, perform hyperparameter tuning using techniques such as grid search or random search to optimize model performance.

2.5.3 Model Evaluation

- Evaluate the trained model(s) on the testing/validation set(s) using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, ROC-AUC (for classification), or mean squared error (for regression).
- Perform cross-validation to assess the robustness of the model and mitigate overfitting.

2.5.4 Deployment

- Once a satisfactory model is trained and evaluated, deploy it to production or integrate it into the software development workflow.

- Monitor the performance of the deployed model over time and periodically retrain/update the model as new data becomes available.

3 Data visualization

Data visualization is the graphical representation of data and information using visual elements such as charts, graphs, and maps. The primary goal of data visualization is to communicate insights and patterns in the data effectively, making it easier for users to understand and interpret complex datasets.

Here are some key aspects of data visualization:

- **Exploratory Analysis:** Data visualization is often used in exploratory data analysis to gain insights into the structure, patterns, and relationships within the data. By visualizing the data in different ways, analysts can uncover trends, outliers, correlations, and other patterns that may not be apparent from raw data alone.
- **Communication:** Data visualization is a powerful tool for communicating findings and insights to stakeholders, decision-makers, and other users. Visualizations help convey complex information in a concise and intuitive manner, making it easier for non-technical audiences to understand and interpret the data.
- **Types of Visualizations:** There are many types of visualizations, each suited to different types of data and analysis tasks. Common types of visualizations include:
- **Bar charts and histograms:** Used to display the distribution of categorical or numerical data.
- **Line charts:** Used to show trends and patterns over time or across categories.
- **Scatter plots:** Used to visualize the relationship between two variables.
- **Pie charts:** Used to show the proportional distribution of categories within a dataset.
- **Heatmaps:** Used to visualize the density or intensity of data values across a two-dimensional grid.
- **Maps:** Used to represent geographical data and spatial relationships.
- **Interactivity:** Many modern data visualization tools and platforms offer interactive features that allow users to explore the data dynamically. Interactivity enables users to drill down into specific details, filter the data, and customize the visualization according to their needs.
- **Data Storytelling:** Data visualization is often used as part of data storytelling, where visualizations are combined with narrative elements to tell a compelling story about the data. Data storytelling helps engage audiences, convey key messages, and guide decision-making based on data-driven insights.

- **Tools and Technologies:** There are numerous tools and technologies available for creating data visualizations, ranging from simple spreadsheet software like Microsoft Excel to more advanced data visualization libraries and platforms like Tableau, ggplot2 (in R), matplotlib (in Python), and D3.js.

Overall, data visualization plays a crucial role in data analysis, exploration, communication, and decision-making across various domains, including business, science, healthcare, finance, and more. By transforming data into visual representations, data visualization enables users to derive meaning, gain insights, and make informed decisions based on data-driven evidence.

3.1 Exploratory data analysis (EDA) to understand data distribution.

Exploratory Data Analysis (EDA) [19] is a critical step in the data analysis process that involves examining and understanding the characteristics, patterns, and relationships present in the dataset. EDA helps analysts gain insights into the data distribution, identify anomalies, and formulate hypotheses for further investigation. Here are some common techniques used in EDA to understand the data distribution:

- **Summary Statistics:** Calculate descriptive statistics such as mean, median, mode, standard deviation, minimum, maximum, and quartiles for numerical variables to understand their central tendency, dispersion, and shape of the distribution.For categorical variables, calculate frequencies and proportions to understand the distribution of categories.
- **Univariate Analysis:** Visualize the distribution of individual variables using histograms for numerical data and bar plots for categorical data. These plots provide insights into the spread and shape of the data distribution.Calculate skewness and kurtosis to assess the symmetry and peakedness of the distribution, respectively.
- **Bivariate Analysis:** Explore relationships between pairs of variables using scatter plots for numerical variables and grouped bar plots or stacked bar plots for categorical variables.Calculate correlation coefficients (e.g., Pearson correlation) to quantify the strength and direction of linear relationships between numerical variables.Use box plots or violin plots to compare the distribution of numerical variables across different categories of a categorical variable.
- **Multivariate Analysis:** Explore relationships among multiple variables simultaneously using techniques such as heatmaps for correlation matrices, parallel coordinate plots, or pair plots (scatterplot matrices).Identify clusters or patterns in high-dimensional data using techniques like principal component analysis (PCA) or t-distributed stochastic neighbor embedding (t-SNE).
- **Outlier Detection:** Identify outliers in the data distribution using visualization techniques such as box plots, scatter plots, or histograms with overlaid density plots.Calculate z-scores or interquartile range (IQR) to quantify the deviation of individual data points from the central tendency of the distribution and identify potential outliers.
- **Data Transformation:** Apply transformations such as log transformation, square root transformation, or Box-Cox transformation to make the data distribution more symmetric and suitable for analysis.

Normalize or standardize numerical variables to bring them to a common scale and facilitate comparisons between variables.By conducting exploratory data analysis, analysts can gain a deeper understanding of the data distribution, uncover patterns and trends, identify potential issues such as outliers or missing values, and generate hypotheses for further analysis. EDA serves as a foundation for more advanced statistical modeling and machine learning tasks and helps ensure the reliability and validity of subsequent analyses.

3.2 Visualizing relationship between features and target variable

Visualizing relationships between features and the target variable is crucial for understanding how different features influence the target and for identifying patterns or trends in the data. Here are several common techniques for visualizing these relationships:

1. **Scatter Plots:** Scatter plots are useful for visualizing the relationship between two numerical variables. Each data point is represented as a point on the plot, with one variable on the x-axis and the other variable on the y-axis. Scatter plots can help identify the presence of linear or non-linear relationships between features and the target variable.
2. **Line Plots:** Line plots are suitable for visualizing trends over time or across ordered categories. They can be used to visualize how a numerical feature varies with the target variable. For example, a line plot can show how the average sales revenue changes over different months of the year.
3. **Bar Plots:** Bar plots are effective for comparing the mean or median of a numerical feature across different categories of a categorical feature. They can be used to visualize how the target variable varies across different groups or levels of a categorical feature.
4. **Box Plots:** Box plots display the distribution of a numerical feature within each category of a categorical feature. They show the median, quartiles, and potential outliers of the numerical feature for each category, making it easy to compare the distributions.
5. **Violin Plots:** Violin plots combine the features of box plots and kernel density plots. They show the distribution of a numerical feature within each category of a categorical feature, along with a kernel density plot. Violin plots provide a more detailed view of the data distribution compared to box plots.
6. **Heatmaps:** Heatmaps are useful for visualizing the relationship between two categorical variables or one categorical and one numerical variable. They represent the frequency or average value of the target variable within each combination of categories, using colors to indicate the values.
7. **Pair Plots:** Pair plots, also known as scatterplot matrices, display scatter plots for all pairs of numerical features in the dataset. Pair plots are helpful for identifying relationships between multiple features and the target variable simultaneously.
8. **Correlation Matrix:** A correlation matrix displays the correlation coefficients between all pairs of numerical features in the dataset. It provides a quantitative measure of the linear relationship between features and the target variable.

3.3 Identifying patterns and trends in the data

Identifying patterns and trends in the data is crucial for gaining insights, making predictions, and informing decision-making. Here are several techniques and approaches for identifying patterns and trends in the data:

1. **Descriptive Statistics**
   - Calculate summary statistics such as mean, median, mode, standard deviation, minimum, maximum, and quartiles for numerical variables.
   - Examine frequency distributions and proportions for categorical variables.
   - Look for outliers, anomalies, or unusual patterns in the data distribution.
2. **Data Visualization**
   - Create visualizations such as histograms, bar plots, line plots, scatter plots, box plots, and heatmaps to explore relationships and patterns in the data.
   - Use time series plots to visualize trends and seasonal patterns over time.
   - Look for clusters or groups of data points using techniques like cluster analysis or dimensionality reduction (e.g., PCA, t-SNE).
3. **Correlation Analysis**
   - Calculate correlation coefficients (e.g., Pearson correlation) between pairs of numerical variables to quantify the strength and direction of linear relationships.
   - Create correlation matrices to visualize the relationships between multiple variables simultaneously.
4. **Time Series Analysis**
   - Decompose time series data into trend, seasonal, and residual components using techniques like seasonal decomposition of time series (STL) or moving averages.
   - Use autocorrelation and partial autocorrelation functions to identify dependencies and patterns in time series data.
5. **Machine Learning Models**
   - Train predictive models such as linear regression, decision trees, random forests, or neural networks to identify predictive patterns and relationships in the data.
   - Use feature importance techniques to identify the most influential variables in predicting the target variable.
6. **Association Rule Mining**
   - Use techniques like Apriori algorithm or FP-Growth algorithm to discover frequent patterns, associations, or co-occurrences among items in transactional datasets.
7. **Text Mining and Natural Language Processing (NLP)**
   - Analyze textual data using techniques such as sentiment analysis, topic modeling, or word embeddings to identify themes, trends, and patterns in the text.
8. **Spatial Analysis**
   - Use geographic information systems (GIS) and spatial analysis techniques to explore spatial patterns, relationships, and trends in geospatial data.
9. **Anomaly Detection**
   - Apply anomaly detection techniques to identify unusual patterns or outliers in the data that deviate significantly from the norm.
10. **Domain Knowledge and Expertise**

- Leverage domain knowledge and expertise to interpret findings, validate patterns, and derive actionable insights from the data.

By employing these techniques and approaches, we can uncover meaningful patterns, trends, and relationships in the data, leading to better understanding, decision-making, and predictive modeling.

4 Software Defect Prediction Using Machine Learning Algorithms

In the Figure 1 we have proposed an architecture for Software Defect Prediction using Machine Learning Algorithms. The architecture leverages the power of ML algorithms to analyze historical software metrics and identify patterns that can be indicative of potential defects. By training on labeled datasets, the ML models can learn to predict the likelihood of defects in future software releases, helping developers prioritize their efforts and allocate resources more effectively.

The proposed architecture consists of several key components, including data preprocessing, feature selection, model training, and evaluation. Various machine learning algorithms, such as k-nearest neighbors (KNN), Decision Trees, Random Forest, and Support Vector Machine (SVM) with a Sigmoid Kernel Function, are employed to build robust prediction models. Details of the preprocessing are discussed in subsection 2.5, while algorithms and their hyperparameters are mentioned in subsections 4.2 and 4.5. These algorithms are chosen for their ability to handle complex, non-linear relationships in software metrics and their proven effectiveness in SDP tasks. The effectiveness of the proposed architecture is demonstrated through experimental results, showcasing its potential to significantly enhance the SDP process.

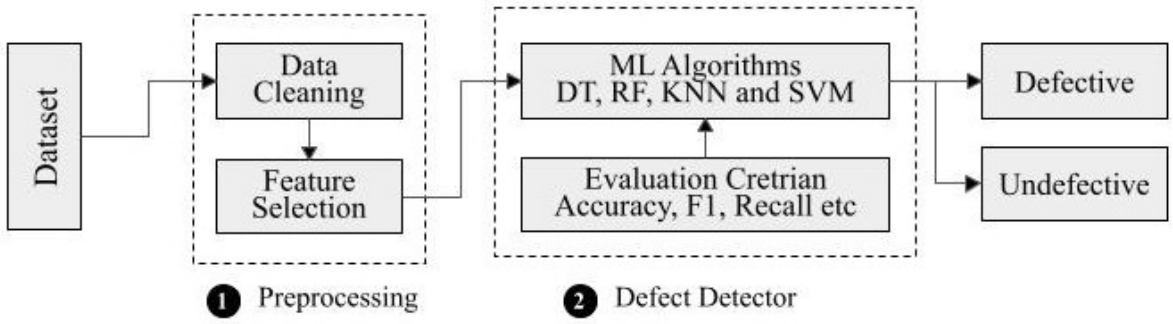


Figure 1: Proposed Architecture for Software Defect Prediction

Machine learning [5] is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and models that allow computers to learn from and make predictions or decisions based on data. Instead of being explicitly programmed to perform a task, a machine learning system is trained using large amount of data to recognize patterns and make informed decisions or predictions.

4.1 Types of Machine Learning Algorithms

Machine learning [5] algorithms are computational techniques that enable computers to learn from data and make predictions or decisions without being explicitly programmed. These algorithms can be broadly categorized into several types based on their learning approach and application:

1. **Supervised Learning**
   - Supervised learning algorithms [20] learn from labeled training data, where each example is associated with a target variable or outcome. The goal is to learn a mapping from input features to the target variable.
   - Examples of supervised learning algorithms include:
     - Linear Regression
     - Logistic Regression
     - Decision Trees
     - Random Forests
     - Support Vector Machines (SVM)
     - K-Nearest Neighbors (KNN)
     - Gradient Boosting Machines (GBM)
     - Neural Networks (including Deep Learning)
2. **Unsupervised Learning**
   - Unsupervised learning algorithms [21] learn patterns and structures in unlabeled data without explicit supervision. The goal is to discover hidden patterns, clusters, or relationships within the data.
   - Examples of unsupervised learning algorithms include:
     - K-Means Clustering
     - Hierarchical Clustering
     - Principal Component Analysis (PCA)
     - t-Distributed Stochastic Neighbor Embedding (t-SNE)
     - Autoencoders
     - Association Rule Mining (e.g., Apriori algorithm)
3. **Semi-Supervised Learning**
   - Semi-supervised learning algorithms leverage both labeled and unlabeled data to improve model performance. They use the labeled data to supervise the learning process and the unlabeled data to capture additional patterns or structures.
   - Examples of semi-supervised learning algorithms include certain variants of clustering algorithms and self-training techniques.
4. **Reinforcement Learning**
   - Reinforcement learning algorithms [22] learn through trial and error by interacting with an environment and receiving feedback in the form of rewards or penalties. The goal is to learn a policy that maximizes cumulative rewards over time.
   - Examples of reinforcement learning algorithms include:
     - Q-Learning
     - Deep Q-Networks (DQN)
     - Policy Gradient Methods (e.g., REINFORCE)
5. **Deep Learning**

- Deep learning algorithms are a subset of machine learning algorithms that use neural networks with multiple layers (deep architectures) to learn complex patterns and representations from data.
- Deep learning has achieved remarkable success in various domains, including computer vision, natural language processing, speech recognition, and reinforcement learning.
- Examples of deep learning architectures include Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformers.

These are just some examples of machine learning algorithms, and there are many more variations and extensions within each category. The choice of algorithm depends on factors such as the nature of the data, the problem to be solved, the available computational resources, and the desired performance metrics.

4.2 Overview of machine learning algorithms used for defect prediction

Machine learning algorithms are increasingly being used for defect prediction in software engineering to identify and prevent software defects before they occur. Here's an overview of some common machine learning algorithms used for defect prediction:

1. **Instance-Based Learning:** Instance-based learning algorithms like k-Nearest Neighbors (k-NN) classify data points based on the majority vote of their nearest neighbors in the feature space. k-NN is a simple and intuitive algorithm that can be effective for defect prediction tasks with small to medium-sized datasets.
2. **Decision Trees:** Decision trees are non-parametric supervised learning algorithms that recursively partition the feature space into regions based on feature values. Decision trees are intuitive and easy to interpret, making them useful for defect prediction tasks where interpretability is important. Ensemble techniques like Random Forests and Gradient Boosting Machines (GBM) can be used to improve the performance of decision trees by combining multiple trees.
3. **Random Forest:** Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the mode of the classes (classification) or the average prediction (regression) of the individual trees. Random Forests are effective for defect prediction tasks and can improve the performance of decision trees by reducing overfitting and increasing robustness.
4. **Support Vector Machines (SVM):** Support Vector Machines (SVM) are supervised learning algorithms that classify data points by finding the hyperplane that best separates the classes in the feature space. SVMs are effective for defect prediction tasks with high-dimensional feature spaces and non-linear decision boundaries. Kernel SVMs can handle non-linear relationships between features and the target variable by mapping the data into a higher-dimensional space.

These are some machine learning algorithms we have used for defect prediction in software engineering. The choice of algorithm depends on factors such as the characteristics of the data, the nature of the defect prediction task, and the desired balance between predictive performance, interpretability, and computational resources. Additionally, preprocessing techniques

such as feature selection, data balancing, and normalization are often applied to improve the effectiveness of machine learning models for defect prediction.

Detailed explanation of each algorithm.

Now, let's dive into a detailed explanation of each algorithm commonly used for defect prediction in software engineering:

4.2.1 K-Nearest Neighbors (KNN)

K-Nearest Neighbors [8](KNN) is a simple and effective classification algorithm. Given a new, unknown observation, it assigns a class based on the majority class among its k nearest neighbors.

Parameters

- k: Number of neighbors to consider.

Formula

The predicted class for a new observation is determined by a majority vote of its neighbors:

$$\hat{y} = \arg\max_{y_i} \sum_{j=1}^{k} I(y_i = y_j) \tag{1}$$

where $\hat{y}$ is the predicted class, y_i is the class of the ith neighbor, and $I(\cdot)$ is the indicator function.

4.2.2 Decision Trees (DT)

Decision Trees [10] [11] recursively split the data into subsets based on the value of a single feature, aiming to minimize impurity or maximize information gain at each split.

Parameters

- Maximum depth: Maximum depth of the tree.
- Minimum samples split: Minimum number of samples required to split a node.

Formula

The decision tree makes predictions by following the decisions in the tree from the root to a leaf node:

$$\hat{y} = \text{Tree}(X) \tag{2}$$

where $\hat{y}$ is the predicted class and $\text{Tree}(X)$ is the tree traversal based on the features of the input X.

4.2.3 Random Forests (RF)

Random Forests [3] [4] are an ensemble method that uses multiple decision trees to improve classification accuracy. Each tree is trained on a random subset of the training data and a random subset of features.

Parameters

- Number of trees: Number of decision trees in the forest.
- Maximum depth: Maximum depth of each tree.
- Minimum samples split: Minimum number of samples required to split a node.

Formula

The prediction is the majority vote of the predictions of the individual trees:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} \text{Tree}_i(X) \tag{3}$$

where $\hat{y}$ is the predicted class, N is the number of trees, and $\text{Tree}_i(X)$ is the prediction of the ith tree.

4.2.4 Support Vector Machines (SVM)

Support Vector Machines [12] find the hyperplane that best separates the classes in the feature space. Different kernels can be used to map the input data into a higher-dimensional space.

Parameters

- Kernel: Type of kernel function (e.g., linear, polynomial, radial basis function (RBF)).
- C: Penalty parameter for the error term.

Formula

For the linear kernel, the decision function is:

$$\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{4}$$

where $\hat{y}$ is the predicted class, $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input vector, and b is the bias term.

For the RBF kernel, the decision function is:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2}\right) + b\right) \tag{5}$$

where α_i are the Lagrange multipliers, y_i are the class labels, $\mathbf{x}_i$ are the support vectors, and σ is the kernel width.

For the sigmoid kernel, the decision function is:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \tanh\left(\gamma(\mathbf{w} \cdot \mathbf{x}_i + b)\right) + b\right) \tag{6}$$

where α_i are the Lagrange multipliers, y_i are the class labels, $\mathbf{x}_i$ are the support vectors, γ is the kernel coefficient, $\mathbf{w}$ is the weight vector, and b is the bias term.

For the polynomial kernel of degree 4, the decision function is:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i (\mathbf{w} \cdot \mathbf{x}_i + b)^4 + b\right) \tag{7}$$

where α_i are the Lagrange multipliers, y_i are the class labels, $\mathbf{x}_i$ are the support vectors, $\mathbf{w}$ is the weight vector, and b is the bias term.

For the polynomial kernel of degree 3, the decision function is:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i (\mathbf{w} \cdot \mathbf{x}_i + b)^3 + b\right) \tag{8}$$

where α_i are the Lagrange multipliers, y_i are the class labels, $\mathbf{x}_i$ are the support vectors, $\mathbf{w}$ is the weight vector, and b is the bias term.

4.3 Evaluating the models using appropriate metrics (e.g., accuracy, precision, recall, F1-score).

Once we have trained our machine learning models, it's crucial to evaluate their performance using appropriate metrics. Here's how we can evaluate the models using common classification metrics such as accuracy, precision, recall, and F1-score:

1. Import Libraries: Make sure we have imported the necessary libraries for evaluation metrics.

2. Calculate Metrics: Compute the evaluation metrics for each model using the predictions and true labels.

3. Display Results: Print or display the evaluation metrics for each model.

By following these steps, we can compute and evaluate the performance of our machine learning models using appropriate classification metrics. These metrics provide insights into different aspects of model performance, such as overall accuracy, precision (ability to avoid false positives), recall (ability to detect true positives), and F1-score (harmonic mean of precision and recall). Evaluating multiple metrics helps to gain a comprehensive understanding of the model's behavior and its suitability for the specific task at hand.

	Predicted Positive (P)	**Predicted Negative (N)**
Actual Positive (P)	True Positives (TP)	False Negatives (FN)
Actual Negative (N)	False Positives (FP)	True Negatives (TN)

4.4 Formulas for Evaluation Metrics

4.4.1 Accuracy:

Accuracy is given by:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

4.4.2 F1 Score:

The F1 Score is the harmonic mean of precision and recall, and it is given by:

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

4.4.3 Recall (Sensitivity or True Positive Rate):

Recall is given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{11}$$

4.4.4 Precision (Positive Predictive Value):

Precision is given by:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{12}$$

4.4.5 Sensitivity (True Positive Rate):

Sensitivity is another term for Recall and is given by the same formula:

$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{13}$$

4.4.6 Specificity (True Negative Rate):

Specificity is given by:

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{14}$$

4.4.7 Balanced Accuracy:

Balanced Accuracy is the average of Sensitivity and Specificity and is given by:

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2} \tag{15}$$

4.5 Hyper-parameters used for each of the classifiers

- **k-Nearest Neighbors (KNN)**:
  - `n_neighbors=5`
  - `weights='uniform'`
  - `leaf_size=30`
  - `p=2` (Euclidean distance)
  - `algorithm='auto'`
- **Random Forest**:
  - `n_estimators=100`
  - `criterion='gini'`
  - `min_samples_split=2`
  - `max_depth=None`
  - `min_samples_leaf=1`
- **Decision Tree**:
  - `criterion='gini'`
  - `max_depth=None`
  - `splitter='best'`
  - `min_samples_split=2`
  - `min_samples_leaf=1`
- **SVM Sigmoid**:
  - `C=1.0`
  - `kernel='sigmoid'`
  - `gamma='scale'`

5 Implementation and Results

Implementing a defect prediction model involves several steps, including data preprocessing, model selection, training, evaluation, and deployment. Here's a high-level overview of the implementation process:

5.0.1 Dataset

We have obtained our dataset for software defect prediction from NASA website using the provided [`https://figshare.com/collections/NASA_MDP_Software_Defects_Data_Sets/4054940/1`]

5.0.2 Model Selection

- Choose appropriate machine learning algorithms for defect prediction, considering factors such as the nature of the data, interpretability requirements, and computational constraints. This may involve experimenting with multiple algorithms and comparing their performance. We've decided to use k-nearest neighbors (kNN), Decision Tree, Random Forest, and Support Vector Machine (SVM) with a Sigmoid Kernel Function for our project.

5.0.3 Model Training

- Split the data into training and testing/validation in 80% and 20% sets to evaluate the performance of the model.
- Train the selected machine learning model(s) on the training data using appropriate hyperparameters.
- Optionally, perform hyperparameter tuning using techniques such as grid search or random search to optimize model performance.

5.0.4 Model Evaluation

- Evaluate the trained model(s) on the testing/validation set(s) using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, ROC-AUC (for classification), or mean squared error (for regression).
- Perform cross-validation to assess the robustness of the model and mitigate overfitting.

5.1 Splitting the data into training and testing sets

Splitting the data into training and testing sets is a critical step in machine learning model development. This process ensures that the model's performance is evaluated on unseen data, helping to assess its generalization ability and prevent overfitting. Here's how we can split the data:

1. **Import Libraries:** Start by importing the necessary libraries for data manipulation and splitting.
2. **Prepare Data:** Assuming we have our feature matrix X and target vector y ready, prepare our data.
3. **Split the Data:**
   - Use train_test_split function from scikit-learn to split the data into training and testing sets.
   - Specify the test size (e.g., 0.2 for 20% test data) and optionally set a random state for reproducibility.
4. **Verify the Split:**
   - Check the shapes of the resulting training and testing sets to ensure that the data has been split correctly.
5. **Usage:** We can now use X_train and y_train for training our machine learning model and X_test and y_test for evaluating its performance.
6. **Optional:** If we need to split the data into training, validation, and testing sets, we can do so by chaining multiple train_test_split calls.
   This splits the data into 80% training, 10% validation, and 10% testing sets. The validation set can be used for hyperparameter tuning or model selection.
   By following these steps, we can effectively split our data into training and testing sets for machine learning model development and evaluation.

5.2 Training the machine learning models using the selected algorithms.

Training machine learning models involves fitting the chosen algorithms to the training data. Here's how We can train models using selected algorithms:

1. Import Libraries: Begin by importing the necessary libraries for the chosen algorithms and evaluation metrics.

2. Instantiate Models: Create instances of the selected machine learning algorithms.

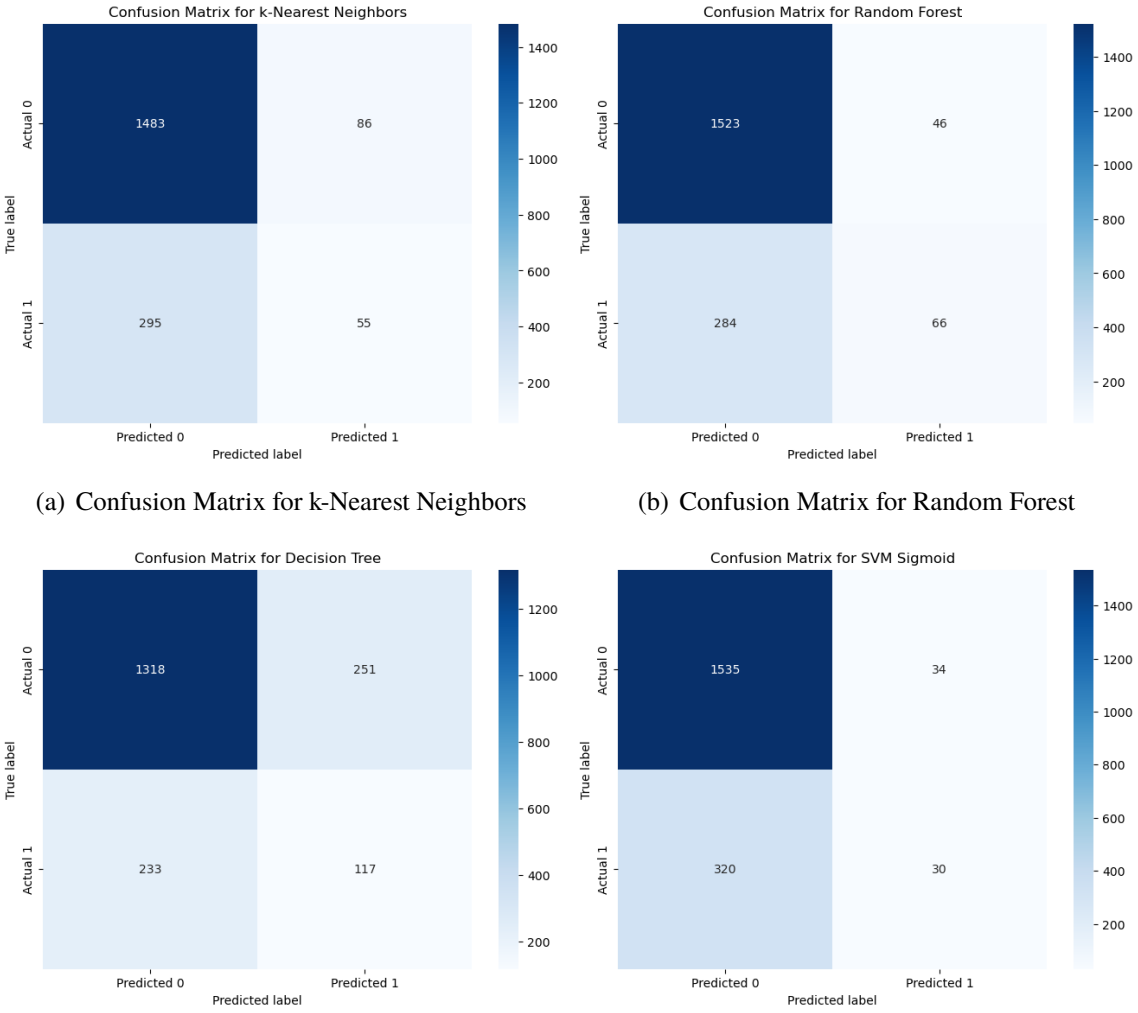
3. Train Models: Fit the models to the training data using the fit method.

4. Model Evaluation: Evaluate the trained models on the testing set using appropriate evaluation metrics.

5. Repeat: Repeat steps 3-4 for each selected algorithm, adjusting hyperparameters and evaluation metrics as necessary.

By following these steps, We can train machine learning models using the selected algorithms and evaluate their performance on the testing set. Adjustments such as hyperparameter tuning and feature engineering can be made to improve model performance further. Additionally, consider using techniques such as cross-validation for robust model evaluation.

5.3 Presenting the results of the machine learning models.



(a) Confusion Matrix for k-Nearest Neighbors

(b) Confusion Matrix for Random Forest

(c) Confusion Matrix for Decision Tree

(d) Confusion Matrix for SVM with Sigmoid KF

Figure 2: Confusion Matrix of RF, KNN, DT and SVM with sigmoid KF

Table 1: Confusion Matrix using Machine Learning Algorithms

No	Classifier	TN	TP	FP	FN
1	K.Nearest Neighbor	1483	55	86	295
2	Random Forest	1523	66	96	284
3	Decision Tree	1318	117	251	233
4	SVM Sigmoid	1535	30	34	320

From Table 2 and 3 the metrics provide an overview of the performance of each classifier. Generally, higher values for accuracy, precision, recall, and F1 score indicate better performance. Based on these metrics, we can see that the Random Forest classifier has the highest accuracy, precison and F1 Score while the Decision Tree classifier has the highest recall. How-

Table 2: Evaluation Metrics using Machine Learning Algorithms

No	Classifier	Accuracy	Precision	Recall	F1 Score
1	K.Nearest Neighbor	0.8015	0.6121	0.5512	0.5551
2	**Random Forest**	**0.8280**	**0.7161**	0.5796	**0.5940**
3	Decision Tree	0.7978	0.5839	**0.5872**	0.5854
4	SVM Sigmoid	0.8155	0.6481	0.5320	0.5208

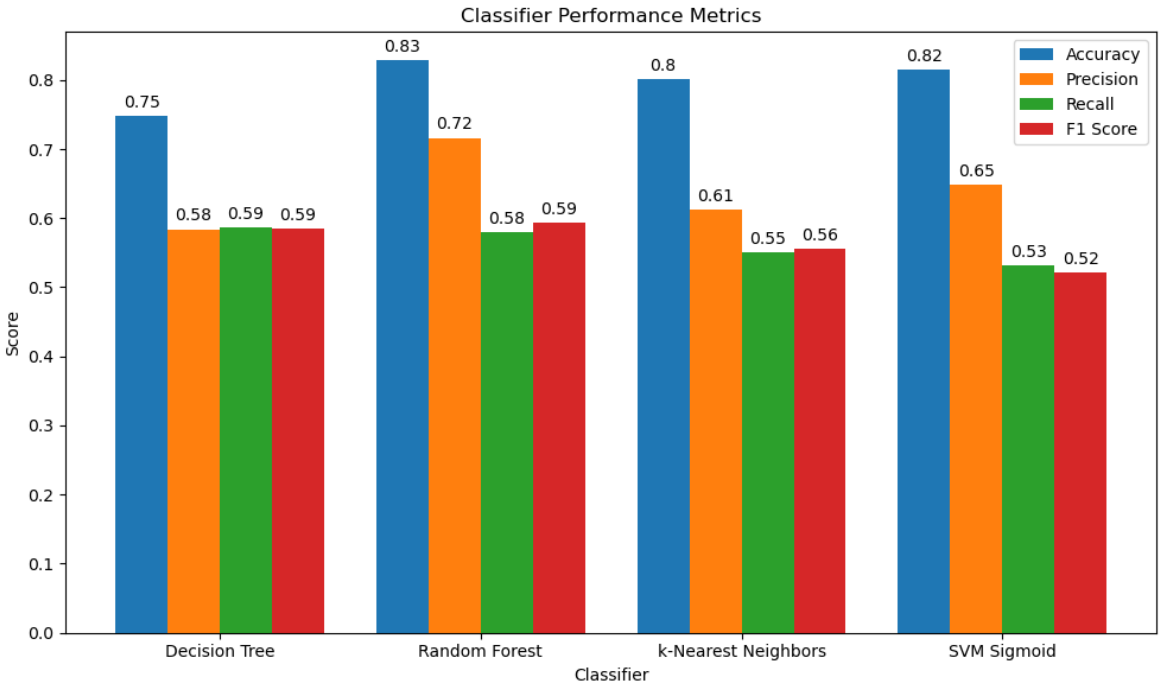


Figure 3: Comparison of four Performance Measures for SDF using different Machine Learning Classifiers

ever, it's essential to consider all metrics together to get a comprehensive understanding of model performance.

In Figure 3, we present a comparison of four performance measures for software defect prediction (SDF) using different machine learning classifiers. The performance measures considered are accuracy, precision, recall, and F1 score, which are commonly used metrics for evaluating the effectiveness of classifiers in binary classification tasks such as SDF.

The machine learning classifiers compared in this study include k-nearest neighbors (KNN), Decision Trees, Random Forest, and Support Vector Machine (SVM) with a Sigmoid Kernel Function. These classifiers were chosen based on their effectiveness in handling SDF tasks and their popularity in the machine learning community.

6 Conclusion

Random Forest performs the best among the classifiers evaluated, with the highest accuracy and F1 score. Decision Tree and SVM Sigmoid have moderate performance, while k-Nearest Neighbors shows relatively weaker performance in terms of precision, recall, and F1 score. When choosing a classifier, it's crucial to consider the specific requirements and priorities of the task at hand. For example, if balanced performance in terms of precision and recall is crucial, Random Forest might be the preferred choice.

6.1 Threats to Validity

The provided confusion matrices and evaluation metrics offer valuable insights into the performance of the classifiers. However, it's essential to acknowledge their limitations:

- **Imbalanced Classes**: The classes in the dataset may be imbalanced, meaning there is a significant difference in the number of instances between the two classes.
- **Limited Feature Set**: The performance of the classifiers may be limited by the features used for training. It's possible that important features are missing or that feature engineering techniques could further enhance the predictive power of the models.
- **Model Complexity:** Decision trees, while interpretable, may suffer from high variance and overfitting, especially if they are deep. Random forests help mitigate this issue to some extent but can still be complex and computationally expensive.
- **Interpretability:** While decision trees offer interpretability, other models like random forests and SVMs may be less interpretable, making it challenging to understand the reasoning behind their predictions. Model interpretability is essential, especially in domains where understanding the decision-making process is critical.

In conclusion, while the provided confusion matrices and evaluation metrics offer valuable information about the classifiers' performance, it's essential to consider their limitations and contextual factors when interpreting the results and making decisions based on them.

6.2 Future work

When considering future work based on the provided confusion matrices and evaluation metrics, several avenues can be explored to enhance the performance of the classifiers:

- For the further work we can use the following ML agorithms like **ADAboost, XGBoost, CatBoost and AutoML.**
- For the further work we can use the following Ensemble Methods such as bagging, boosting, and stacking to combine multiple models and improve predictive performance.
- Perform comprehensive hyperparameter tuning for each algorithm to find the optimal combination of hyperparameters that maximize performance metrics such as accuracy, precision, recall, and F1 score.

- Explore advanced machine learning techniques such as deep learning, reinforcement learning, or Bayesian optimization, depending on the nature of the problem and the available data. These techniques may uncover complex patterns and relationships in the data that traditional algorithms may miss.
- For future work, we may consider hyperparameter tuning to optimize the performance of the chosen algorithms, such as k-nearest neighbors (kNN), Decision Tree, Random Forest, and Support Vector Machine (SVM) with a Sigmoid Kernel Function, on your dataset.

7 Code Avalibility

All code of the current implementation is uploaded in GitHub repository : `https://github.com/joshimanjari/Software_Defect_Prediction`

References

[1] Giray, Görkem, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, and Bedir Tekinerdogan. "On the use of deep learning in software defect prediction." Journal of Systems and Software 195 (2023): 111537.

[2] Fenton, Norman E., and Martin Neil. "A critique of software defect prediction models." IEEE Transactions on software engineering 25, no. 5 (1999): 675-689.

[3] Belgiu, Mariana, and Lucian Drăguţ. "Random forest in remote sensing: A review of applications and future directions." ISPRS journal of photogrammetry and remote sensing 114 (2016): 24-31.

[4] Biau, Gérard, and Erwan Scornet. "A random forest guided tour." Test 25 (2016): 197-227.

[5] Liakos, Konstantinos G., Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. "Machine learning in agriculture: A review." Sensors 18, no. 8 (2018): 2674.

[6] Jordan, Michael I., and Tom M. Mitchell. "Machine learning: Trends, perspectives, and prospects." Science 349, no. 6245 (2015): 255-260.

[7] Zheng, Alice, and Amanda Casari. Feature engineering for machine learning: principles and techniques for data scientists. " O'Reilly Media, Inc.", 2018.

[8] Peterson, Leif E. "K-nearest neighbor." Scholarpedia 4, no. 2 (2009): 1883.

[9] Guo, Gongde, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. "KNN model-based approach in classification." In On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings, pp. 986-996. Springer Berlin Heidelberg, 2003.

[10] Charbuty, Bahzad, and Adnan Abdulazeez. "Classification based on decision tree algorithm for machine learning." Journal of Applied Science and Technology Trends 2, no. 01 (2021): 20-28.

[11] Song, Yan-Yan, and L. U. Ying. "Decision tree methods: applications for classification and prediction." Shanghai archives of psychiatry 27, no. 2 (2015): 130.

[12] Hearst, Marti A., Susan T. Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. "Support vector machines." IEEE Intelligent Systems and their applications 13, no. 4 (1998): 18-28.

[13] Hosmer Jr, David W., Stanley Lemeshow, and Rodney X. Sturdivant. Applied logistic regression. Vol. 398. John Wiley & Sons, (2013).

[14] Ferri, César, José Hernández-Orallo, and R. Modroiu. "An experimental comparison of performance measures for classification." Pattern recognition letters 30, no. 1 (2009): 27-38.

[15] Jelinski, Zygmunt, and P. Moranda. "Software reliability research." In Statistical computer performance evaluation, pp. 465-484. Academic press, 1972.

[16] Cai, Jie, Jiawei Luo, Shulin Wang, and Sheng Yang. "Feature selection in machine learning: A new perspective." Neurocomputing 300 (2018): 70-79.

[17] Suhaidi, Mustazzihim, R. Abdul Kadir, and Sabrina Tiun. "A review of feature extraction methods on machine learning." J. Inf. Syst. Technol. Manag. 6, no. 22 (2021): 51-59.

[18] Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.

[19] Da Poian, Victoria, Bethany Theiling, Lily Clough, Brett McKinney, Jonathan Major, Jingyi Chen, and Sarah Hörst. "Exploratory data analysis (EDA) machine learning approaches for ocean world analog mass spectrometry." Frontiers in Astronomy and Space Sciences 10 (2023): 1134141.

[20] Nasteski, Vladimir. "An overview of the supervised machine learning methods." Horizons. b 4 (2017): 51-62.

[21] Alloghani, Mohamed, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain, and Ahmed J. Aljaaf. "A systematic review on supervised and unsupervised machine learning algorithms for data science." Supervised and unsupervised learning for data science (2020): 3-21.

[22] Mehta, Deepanshu. "State-of-the-art reinforcement learning algorithms." International Journal of Engineering Research and Technology 8 (2020): 717-722.