ABSTRACT

Software defect prediction (SDP) is an important challenge in the field of software engineering; hence much research work has been conducted, most notably using machine learning algorithms. However, class-imbalance typified by few defective components and many non-defective ones is a common occurrence causing difficulties for these methods.

Finding software bugs is a crucial step in the software development life cycle. A software defect is a flaw or shortage in a work product that prevents it from meeting requirements or specifications and necessitates repair or replacement. Early software defect detection helps the business avoid time and money losses. Numerous algorithms have been put forth for this purpose to predict software defects, but these models still have some drawbacks. This project work has been designed as a Prediction system that uses Ensemble learning and a Hybrid feature selection technique for predicting the defect in software modules to increase the efficiency in finding the defects.

LIST OF FIGURES

Figure No.	Name of Figure	Page No.
2.1.1	Ensemble model	3
2.2.1	Existing system using single model	4
3.1.1	Block diagram of proposed system	10
3.2.1	Activity diagram of proposed system	11
3.2.2	Sequence diagram of proposed system	12
3.2.3	Use case diagram of proposed system	13
3.2.4	Data flow diagram of proposed system	14
3.4.1.1	Bagging ensemble learning	17
3.4.1.2	Boosting ensemble learning	18
3.4.1.3	Stacking ensemble learning	19
3.4.2.1	SMOTE process	20
3.4.3.1	CHI Square method	21
4.1.1.1	Random Forest Model	24
4.3.1	User interface for inputting features	29
4.3.2	Submitting input features	29
4.3.3	Output of given features	30
4.3.4	Submitting input features	30
4.3.5	Output of given features	31
5.1	SMOTE process	32
5.2	Calculation metrics	33

TABLE OF CONTENTS

S. No.		Page. No
	Title Page	I
	Certificate of the Guide	II
	Declaration of the Student	III
	Acknowledgement	IV
	Abstract	V
	List of Figures	VI
1.	**INTRODUCTION**	1
	1.1 Problem Statement	1
	1.2 Objectives of the Project	1
	1.3 Scope of the Project	1
	1.4 Motivation	1
	1.5 Applications	2
2.	**LITERATURE SURVEY**	3
	2.1 Introduction to Problem Domain and Technology	3
	2.2 Existing System	3
	2.3 Related Works	4
	2.4 Tools and Technologies used	6
	2.4.1 Python	6
	2.4.2 Machine Learning	7
	2.4.3 HTML	8
	2.4.4 CSS	8
	2.4.5 Django	9
3.	**DESIGN OF THE PROPOSED SYSTEM**	10
	3.1 Proposed System	10
	3.2 UML Diagrams	11
	3.2.1 Activity Diagram	11
	3.2.2 Sequence Diagram	12
	3.2.3 Use Case Diagram	13

3.2.5 Data Flow Diagram 14

3.3 Module Description 15

3.3.1 Data Pre-Processing 15

3.3.2 Ensemble Learning 15

3.3.3 User Interface 16

3.4 Theoretical Foundation 16

3.4.1 Ensemble Learning 16

3.4.1.1 Bagging 17

3.4.1.2 Boosting 18

3.4.1.3 Stacking 19

3.4.2 SMOTE 20

3.4.3 Feature Selection 21

3.4.3.1 CHI SQUARE method 21

3.4.3.2 Sequence Backward Selection 21

4. **IMPLEMENTATION OF THE PROPOSED SYSTEM** 22

4.1 Algorithms 22

4.1.1 Random Forest Algorithm 23

4.1.2 Logistic Regression 25

4.1.3 Linear Regression 26

4.2 Dataset Description 27

4.3 Implementation 28

5. **RESULTS AND ANALYSIS** 32

6. **CONCLUSION AND FUTURE SCOPE** 34

7. **REFERENCES** 35

8. **APPENDIX** 36

1. INTRODUCTION

1.1 Problem Statement

Software Defect Prediction is the process of identifying whether defect present in the developed software or not. Software defect prediction plays a vital role in the software development process as it helps in reducing the cost of repairing defects and time for predicting the defects. To minimize the time and cost of identifying defects, a technique based on Ensemble learning, SMOTE and Feature Selection techniques were applied.

1.2 Objective of Project

The primary objective of model is to predict whether the given source code contain defect or not by using 21 features given in dataset. We are using SMOTE balancing technique, Feature selection technique and Ensemble Model for better accuracy. Using this approach, we can test software module to find defects, which helps us reduce time and cost effective.

1.3 Scope of the Project

The existing work predicts the software defect using a single machine learning model and has its limitations. We would like to improve the performance of the model by using a hybrid feature selection mechanism and SMOTE (Synthetic Minority Oversampling Technique) method. The usage of a Hybrid learning model helps to predict the defect efficiently.

1.4 Motivation

Software designers regularly use defect analysis to better access programming quality and development quality. Software defect analysis is a strategy for characterizing imperfections and mining the reasons for defects. The motivation behind software defect analysis is to enable analysts to find, locate, evaluate, and improve test efficiency. The defects analysis methods are mostly partitioned into qualitative analysis, quantitative analysis, and attribute analysis.

Software defect prediction plays an important role in improving software quality and it help to reducing time and cost for software testing. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data.The ability of a machine to improve its performance based on previous results. Machine learning improves efficiency of human learning, discover new things or structure that is unknown to humans and find important information in a document. For that purpose, different machine learning techniques are used to remove the unnecessary, erroneous data from the dataset. Software defect prediction is seen as a highly important ability when planning a software project and much greater effort is needed to solve this complex problem using a software metrics and defect dataset. Metrics are the relationship between the numerical value and it applied on the software therefore it is used for predicting defect.

1.5 Applications

Improving software quality: Software defect prediction can help identify potential defects early in the development process, allowing developers to fix them before the software is released. This can lead to higher-quality software with fewer defects.

Prioritizing testing efforts: By identifying the areas of code that are more likely to have defects, software defect prediction can help testing teams prioritize their efforts and focus on the most critical areas.

Resource allocation: Software defect prediction can also help project managers allocate resources more effectively, such as assigning more developers to areas of code with a higher likelihood of defects.

Cost reduction: By identifying and addressing defects earlier in the development process, software defect prediction can help reduce the cost of fixing defects after the software is released.

Decision making: By providing insights into the potential risks associated with different code changes, software defect prediction can help decision-makers make informed decisions about software development and deployment.

2. LITERATURE SURVEY

2.1 Introduction to the problem domain terminology

A software defect is an error or deficiency that causes a work product to fall short of requirements or specifications and necessitates repair or replacement. Software defects prediction helps the developers to identify the defects and also split the tasks accordingly.

Different software metrics are related to software defect prediction such as Lines of code, Cyclomatic complexity, Essential complexity, Volume, Effort, Difficulty, Time estimator, Intelligence, etc to detect a defect.

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Unlike a statistical ensemble in statistical mechanics, which is usually infinite, a machine learning ensemble consists of only a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives.

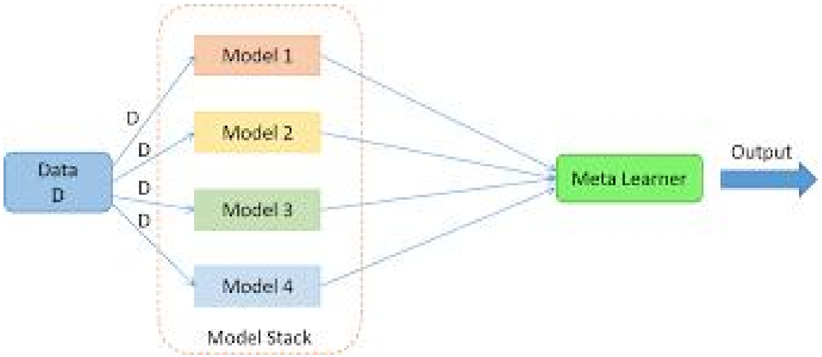


Fig:2.1.1 Ensemble model

2.2. Existing System

The existing work focuses on only using a single machine learning model. Meanwhile, most of the existing software defect prediction models cannot make predictions at the code line level, which makes it extremely arduous to provide developers with more detailed reference information. Existing model leverages abstract syntax trees (ASTs) as the intermediate representation of code snippets. Since the historical defect data has a striking characteristic of class imbalance, an approach based on Self- organizing Map (SOM) clustering is employed to handle noisy data.

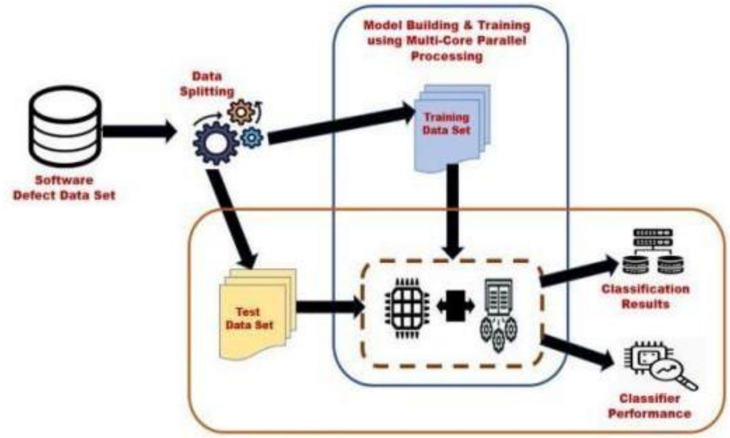


Fig:2.2.1 Existing system using single model

2.3 Related Works

Q. Song, Y. Guo and M. Shepperd, "A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction," in IEEE Transactions on Software Engineering, vol. 45, no. 12, pp. 1253-1269, 1 Dec. 2019, doi: 10.1109/TSE.2018.2836442.

Goal is to conduct a large-scale comprehensive experiment to study the effect of imbalanced learning and its complex interactions between the type of classifier, data set characteristics and input metrics in order to improve the practice of software defect prediction. We first discuss our choice of MCC as the performance measure and then describe the experimental design including algorithm evaluation, statistical methods and software defect data sets.

Z. Sun, Q. Song and X. Zhu, "Using Coding-Based Ensemble Learning to Improve Software Defect Prediction," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 6, pp. 1806-1817, Nov. 2012, doi: 10.1109/TSMCC.2012.2226152.

This proposed method addresses the class-imbalance problem in software defect prediction by turning a binary classification problem into a multiclassification problem. Specifically, the majority class data (non-defective components) are first split into several bins. Each bin then has the similar number of examples to that of the minority class (defective components), and is assigned a new class label. After that, the relabelled data are employed to build a multi classifier, which no longer suffers from the skewed data because the new class distribution is balanced. Finally, the multiclassification results are converted into the original binary, namely either defective or non-defective. Our proposed method consists of three components: data balancing, multi classifier modelling, and defect prediction.

L. Gong, S. Jiang and L. Jiang, "Tackling Class Imbalance Problem in Software Defect Prediction Through Cluster-Based Over-Sampling With Filtering," in IEEE Access, vol. 7, pp. 145725- 145737, 2019, doi: 10.1109/ACCESS.2019.2945858.

KMFOS method introduces K-Means clustering algorithm and noise filtering into over-sampling technique to balance imbalanced datasets. Not only is the imbalance between classes considered, but also the imbalance between internal defective class is considered. This can avoid the non-diverse distribution of defective instances. Moreover, using noise filtering can avoid the influence of noise instances. KMFOS includes three steps: clustering, over-sampling and filtering. In clustering step, the defective instances are clustered into k clusters by K-Means. In the over-sampling step, the new minority instances are generated based on two minority instances from two different clusters. In the filtering step, we apply CLNI algorithm to detect and clean noisy instances.

J. Zheng, X. Wang, D. Wei, B. Chen and Y. Shao, "A Novel Imbalanced Ensemble Learning in Software Defect Predication," in IEEE Access, vol. 9, pp. 86855-86868, 2021, doi: 10.1109/ACCESS.2021.3072682.

With the availability of high-speed Internet and the advent of Internet of Things devices, modern software systems are growing in both size and complexity. Software defect prediction (SDP) guarantees the high quality of such complex systems. However, the characteristics of imbalanced distribution of defect data sets have led to the deviation and loss of accuracy of most software defect prediction methods. This paper presents two novel approaches for learning from imbalanced data sets to produce a higher predictive accuracy over the minority class. These two methods differ in whether the oversampling and the misclassification cost information are utilized during training stage, and they are good at different aspects of imbalanced classification, one for dealing with highly imbalanced data sets and the other for moderately imbalanced data sets. Comparing with other state-of-the-art imbalance learning algorithms on imbalanced datasets, the experimental results show that these two methods have achieved excellent results in terms of G-mean and AUC measures, and more accurately identified the defective modules to reduce the cost of detection system.

S. Huda et al., "A Framework for Software Defect Prediction and Metric Selection," in IEEE Access, vol. 6, pp. 2844-2858, 2018, doi: 10.1109/ACCESS.2017.2785445.

Automated software defect prediction is an important and fundamental activity in the domain of software development. However, modern software systems are inherently large and complex with numerous correlated metrics that capture different aspects of the software components. This large number of correlated metrics makes building a software defect prediction model very complex. Thus, identifying and selecting a subset of metrics that enhance the software defect prediction method's performance are an important but challenging problem that has received little attention in the literature. The main objective of this paper is to identify significant software metrics, to build and evaluate an automated software defect prediction model. We propose two novel hybrid software defect prediction models to identify the significant attributes (metrics) using a combination of wrapper and filter techniques. The novelty of our approach is that it embeds the metric selection and training processes of software defect prediction as a single process while reducing the measurement overhead significantly. Different wrapper approaches were combined, including SVM and ANN, with a maximum relevance filter approach to find the significant metrics.

2.4 Tools and Technologies used

Here we will discuss various tools and technologies that are used to implement the proposed system.

2.4.1 Python

In this project, we have used Python as our main programming language. As it is simple and effective, also helps fast development. It is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. It's inbuild libraries, and vast number of third-party libraries makes development process faster and efficient. Python offers concise and readable code. While Python can be used in a vast number of domains and technologies such AI, Data Science, ML, and automation tools. Python is also very good at web development. We use Python in this project for backend as it offers a MVT (Model-View-Template) framework in Django. Which can simplify developing web applications. It provides built-in security features, such as protection against cross-site scripting (XSS) and cross-site request forgery (CSRF), making it a secure framework for web development.

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

2.4.2 Machine Learning

Machine learning can be used to predict software defects by analyzing various software metrics and historical data. Here are the basic steps involved in using machine learning for software defect prediction

Data Collection: Collect data on the software development process, including software metrics such as code complexity, code coverage, number of lines of code, and number of defects.

Feature Selection: Select the most relevant features that can be used to predict software defects. This can be done by analyzing the correlation between the software metrics and the number of defects.

Data Preprocessing: The collected data needs to be cleaned and preprocessed to ensure it is suitable for machine learning algorithms.

Model Training: Train a machine learning model using the preprocessed data. Several machine learning algorithms can be used, such as decision trees, logistic regression, and neural networks.

Model Evaluation: Evaluate the accuracy of the machine learning model using a testing dataset. This is done to ensure the model is accurate and can be used for software defect prediction.

Prediction: Once the machine learning model is trained and evaluated, it can be used to predict software defects in new software projects based on the selected features.

2.4.3 HTML

HTML stands for HyperText Markup Language. It is a standard markup language used to create web pages and web applications. HTML defines the structure and content of web pages, including text, images, videos, links, and other elements.

HTML uses tags to mark up content, which are enclosed in angle brackets, such as <html>, <head>, <title>, <body>, <p>, <img>, and <a>. Each tag represents a specific element or piece of content on the page.

2.4.4 CSS

CSS stands for Cascading Style Sheets. It is a style sheet language used to describe the presentation of HTML and XML documents, including the layout, colors, fonts, and other visual elements.

CSS separates the presentation of a web page from its content, allowing designers to create consistent and attractive layouts without changing the HTML structure of the page. By using CSS, you can control the appearance of multiple web pages at once, making it easier to maintain and update your website.

CSS works by defining style rules that are applied to specific HTML elements. Each rule consists of a selector that specifies the element(s) to style, and a set of declarations that specify the styling properties and values.

2.4.5 Django

Django is a high-level web framework for Python that follows the model-view-controller (MVC) architectural pattern. It is designed to help developers build complex, database-driven web applications quickly and easily.

Django includes a wide range of features and tools, such as an object-relational mapper (ORM) for database access, a templating engine for rendering HTML pages, a URL routing system for mapping URLs to views, and built-in support for user authentication and authorization.

3. DESIGN OF THE PROPOSED SYSTEM

The 3rd chapter describes the design of the proposed system and diagrams to support the model.

3.1 Proposed System

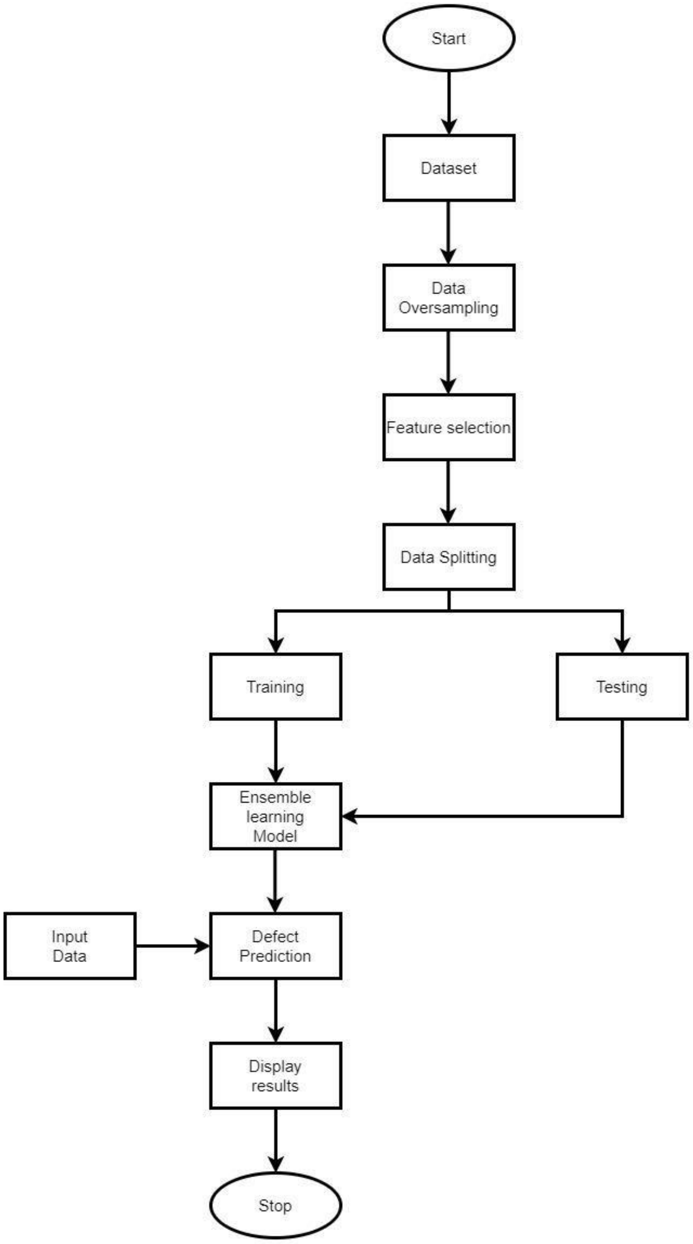


Figure 3.1.1 Block diagram of the proposed system

Fig 3.1.1 shows the block diagram of the proposed system where the input is taken and we perform the oversampling of the data using SMOTE technique later we perform the hybrid feature selection to select the appropriate features from the dataset and after that, we split the dataset into the training and testing data then we train the model using training data and we use test data to test the performance of the model. We give the input to the ensemble model to predict the output and display the results to the user.

3.2 UML Diagrams

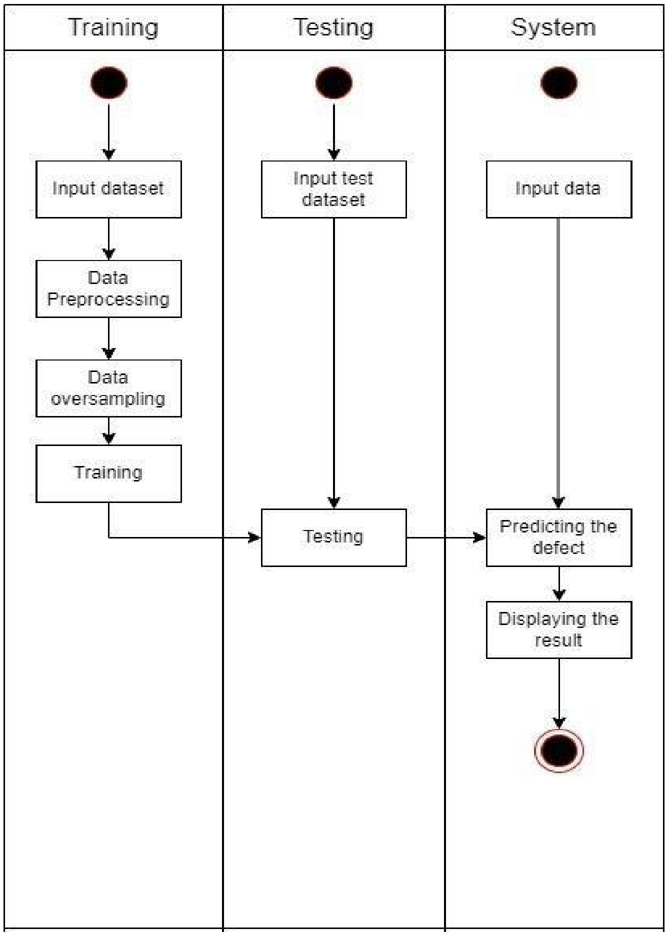


Figure 3.2.1 Activity diagram of the proposed system

Fig 3.2.1 shows the activity diagram of the proposed system which shows the series of actions involved in the proposed system. There are mainly 3 things in the system training, testing, and system are part of the activities where training involves in training the model and testing involves testing the model with the test values and also measuring the accuracy, precision, recall, and fscore metrics to measure the performance of the model. After predicting the output it displays the results to the user.

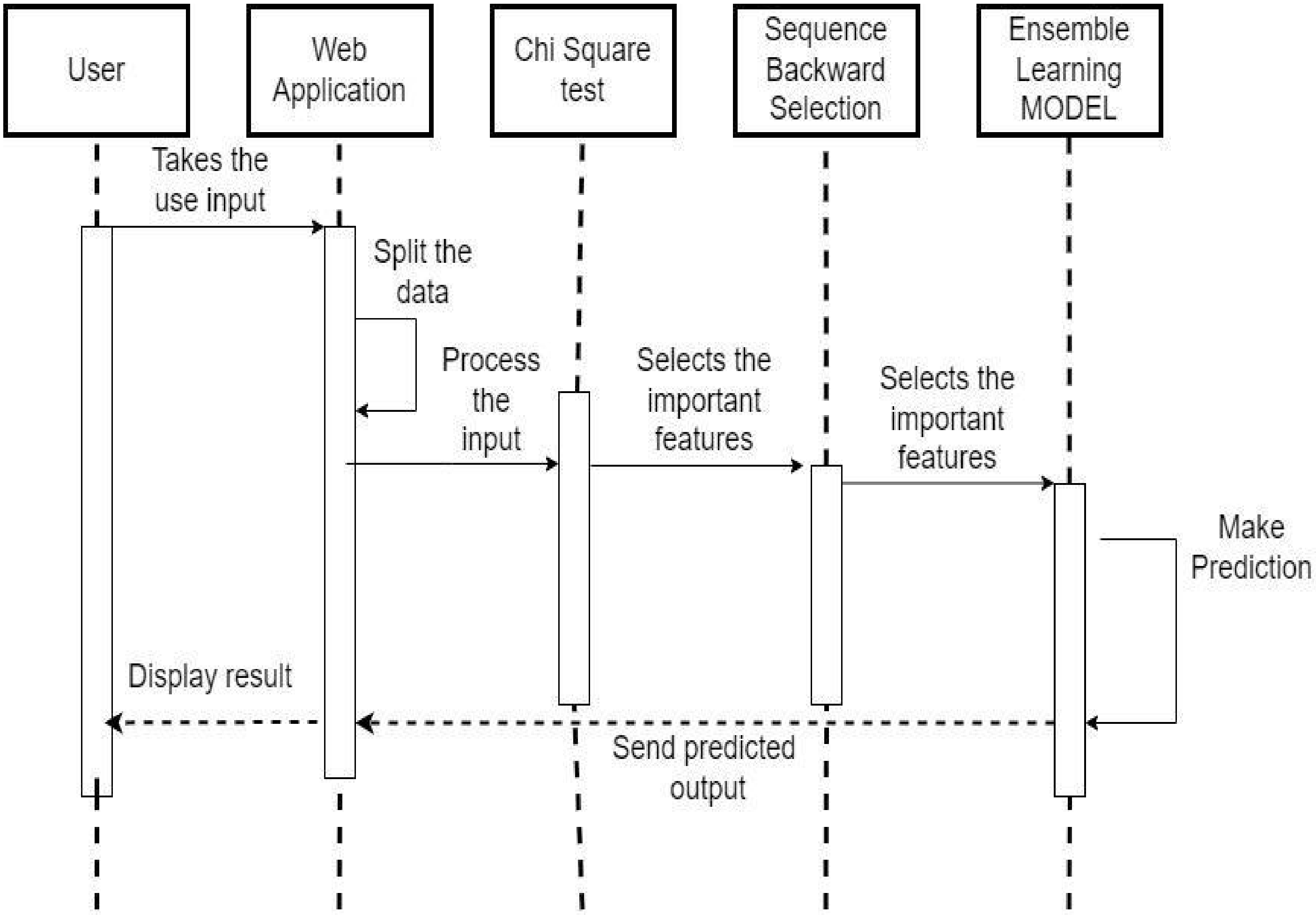


Figure 3.2.2 Sequence diagram of the proposed system

Fig 3.2.2 shows the sequence diagram of the proposed system. The sequence diagram represents the flow of messages in the system and it is also termed an event diagram. In the above diagram, we can see how the objects are working together in order to predict the output.

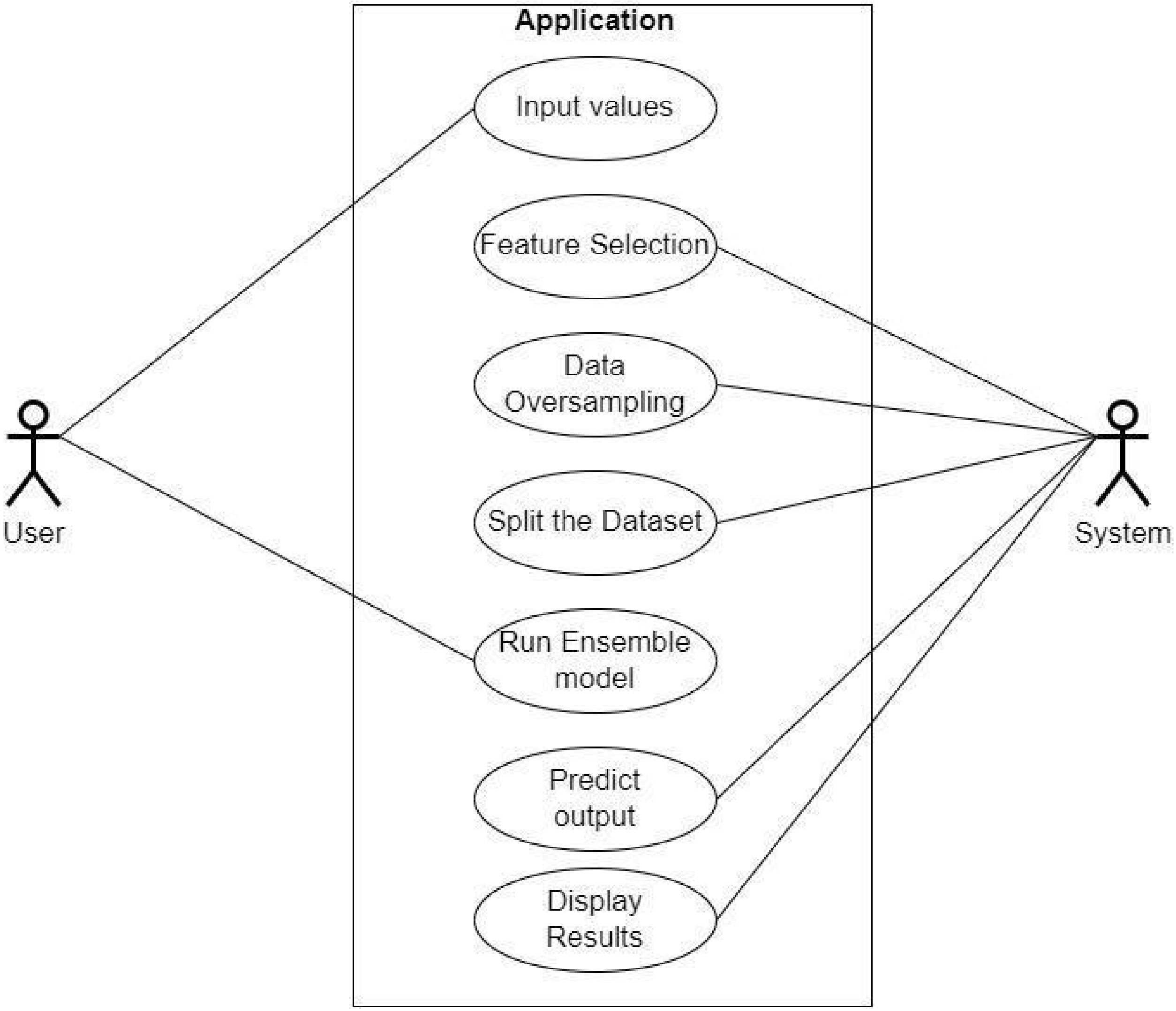


Figure 3.2.3 Usecase diagram of the proposed system

Fig 3.2.3 shows the use case diagram of the proposed system. A use case diagram represents the behavior of the system by incorporating the actors, use cases, and their relationships. In the above use case diagram we can see that user performs the actions like giving the input values and running the model and all the other processes will be done by the system like data oversampling, splitting the dataset, feature selection, and finally predicting the result and displaying them back to the user.

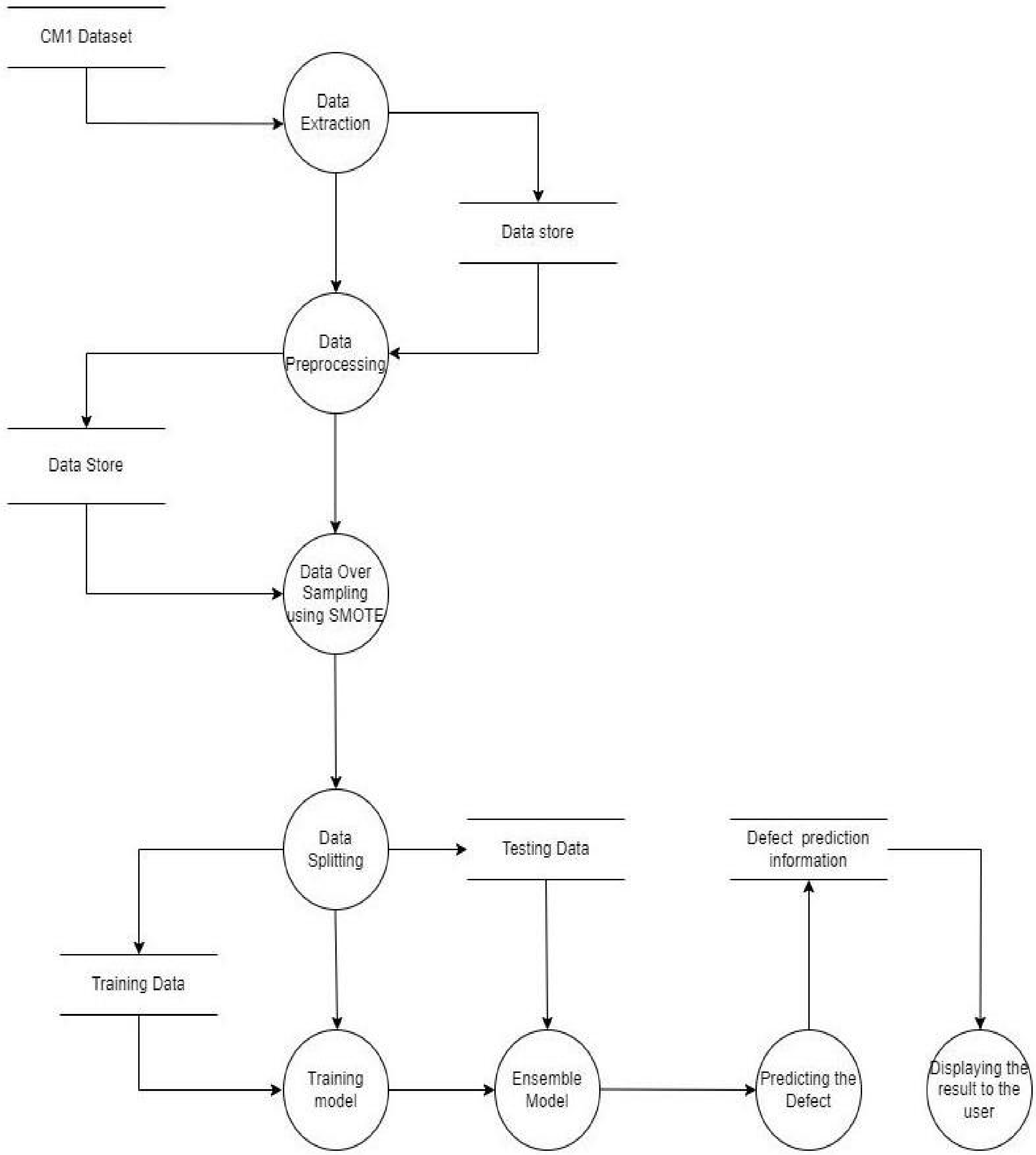


Figure 3.2.4 Data Flow diagram of the proposed system

Fig 3.2.4 shows the Data Flow diagram of the proposed system. A data flow diagram represents how the data flow in the entire system. We can see the datastore where the data is stored after processing. We also store the test and train data separately in the data stores.

3.3 Module Description

3.3.1 Data Preprocessing with SMOTE Oversampling and Feature Selection

This module is responsible for oversampling the imbalanced software defect dataset and selecting the most relevant features of raw software defect data using the Chi-Square test and Sequence Backward Selection(SBS).

This module performs SMOTE oversampling which means generating the synthetic samples of minority class to balance the class distribution. After oversampling the data we perform feature selection using hybrid feature selection. First, we select certain features from the Chi-square test, and from these features, we select the final set of features using SBS.

The input to this module is raw data in CSV format. The output of this module is preprocessed data with a balanced class distribution and also with elected features in a standardized format for use in machine learning models. This module interacts with the Data collection module to obtain the raw data and the Ensemble Learning module to provide the preprocessed data as input for training and testing the models.

3.3.2 Ensemble learning

This module is responsible for building, training, and testing an ensemble learning model for software defect prediction using a combination of random forest, linear regression, and logistic regression as the base models.

This module performs the splitting of the preprocessed data into training and testing. After splitting the preprocessed data we are using three models random forest, linear regression, and logistic regression algorithms for the model. Training the base models with training data and with respective algorithms.

The input to this module is preprocessed software defect data with balanced class distribution and selected features in a standardized format. The output of this module is a trained and tested ensemble learning model for defect prediction.

3.3.3 User Interface

This module is responsible for taking the input features from the user and obtaining a prediction of whether the software is likely to contain defects or not. The GUI should include input fields for relevant features and an output field displaying the predicted output.

3.4 Theoretical Foundation

3.4.1 Ensemble Learning

Ensemble learning can be implemented using different techniques such as bagging, boosting, and stacking. Bagging involves training multiple models on bootstrap samples of the training data and aggregating their predictions by majority voting or averaging. Boosting, on the other hand, involves sequentially training models on weighted versions of the training data, where more weight is given to the misclassified instances. Stacking involves combining the predictions of multiple models using another model, such as a logistic regression or a neural network.

Ensemble learning is a machine learning technique that combines multiple individual models to improve the overall predictive performance. The individual models can be trained using the same or different algorithms and can be combined using various methods such as voting, averaging, or stacking. The main advantage of ensemble learning is that it can reduce the risk of overfitting and improve the generalization performance of the model. By combining the predictions of multiple models, the ensemble model can capture a broader range of patterns in the data and reduce the impact of individual model errors.

Example: If you are planning to buy an air-conditioner, would you enter a showroom and buy the air-conditioner that the salesperson shows you? The answer is probably no. In this day and age, you are likely to ask your friends, family, and colleagues for an opinion, do research on various portals about different models, and visit a few review sites before making a purchase decision. In a nutshell, you would not come to a conclusion directly. Instead, you would try to make a more informed decision after considering diverse opinions and reviews. In the case of ensemble learning, the same principle applies.

The three main classes of ensemble learning methods are bagging, stacking, and boosting.

3.4.1.1 BAGGING ENSEMBLE LEARNING

Bootstrap aggregation, or bagging, is an ensemble learning method that seeks a diverse group of ensemble members by varying the training data. This involves using a single machine learning algorithm, mostly an unpruned decision tree, and training each model on a different sample of the same training dataset. The predictions made by the ensemble members are then combined using simple statistics, such as voting or averaging.

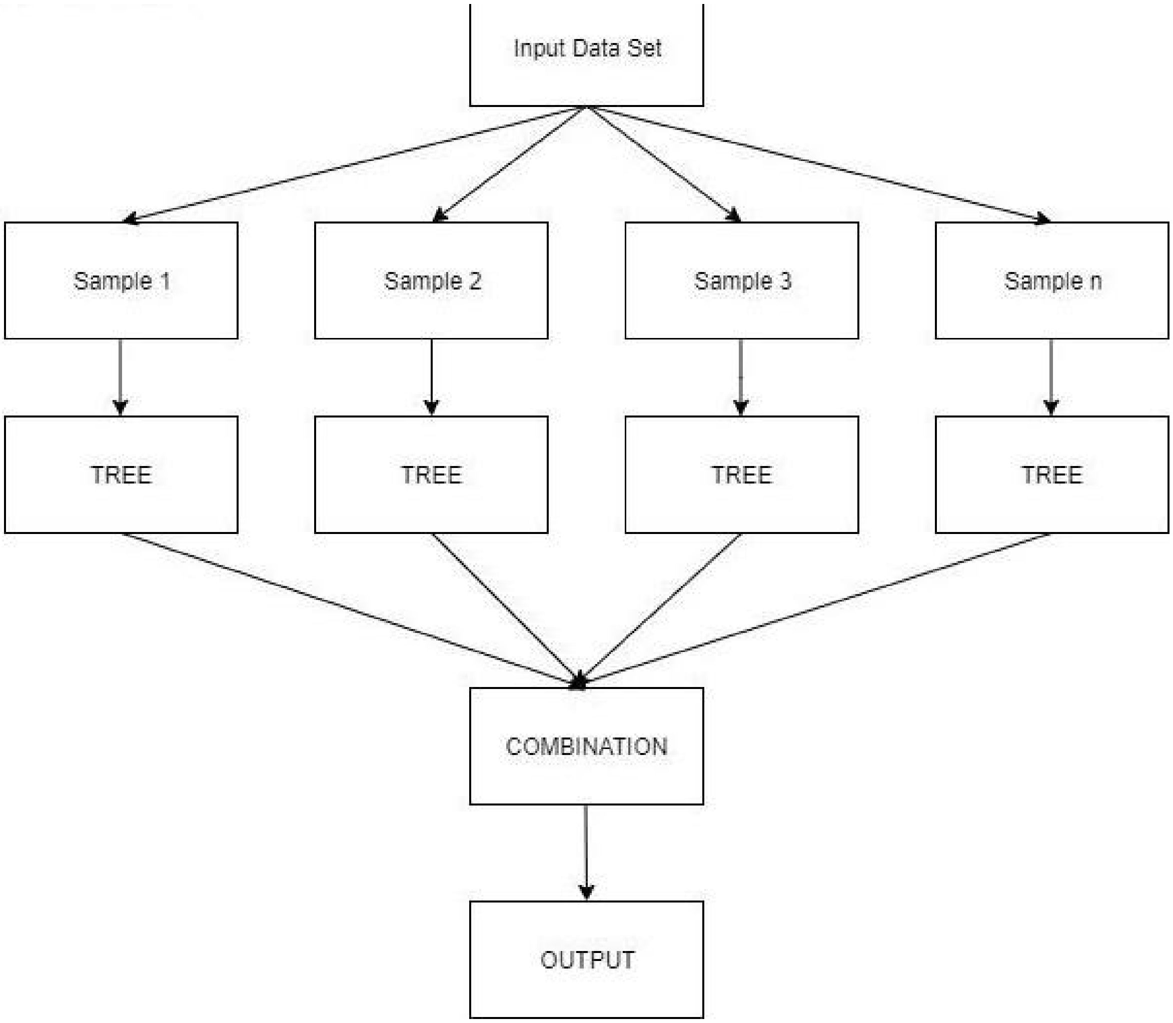


Fig 3.4.1.1 Bagging Ensemble Learning

3.4.1.2 BOOSTING ENSEMBLE LEARNING

Boosting is an ensemble method that seeks to change the training data to focus attention on examples that previous fit models on the training dataset have gotten wrong. The key property of boosting ensembles is the idea of correcting prediction errors. The models are fit and added to the ensemble sequentially such that the second model attempts to correct the predictions of the first model, the third corrects the second model, and so on

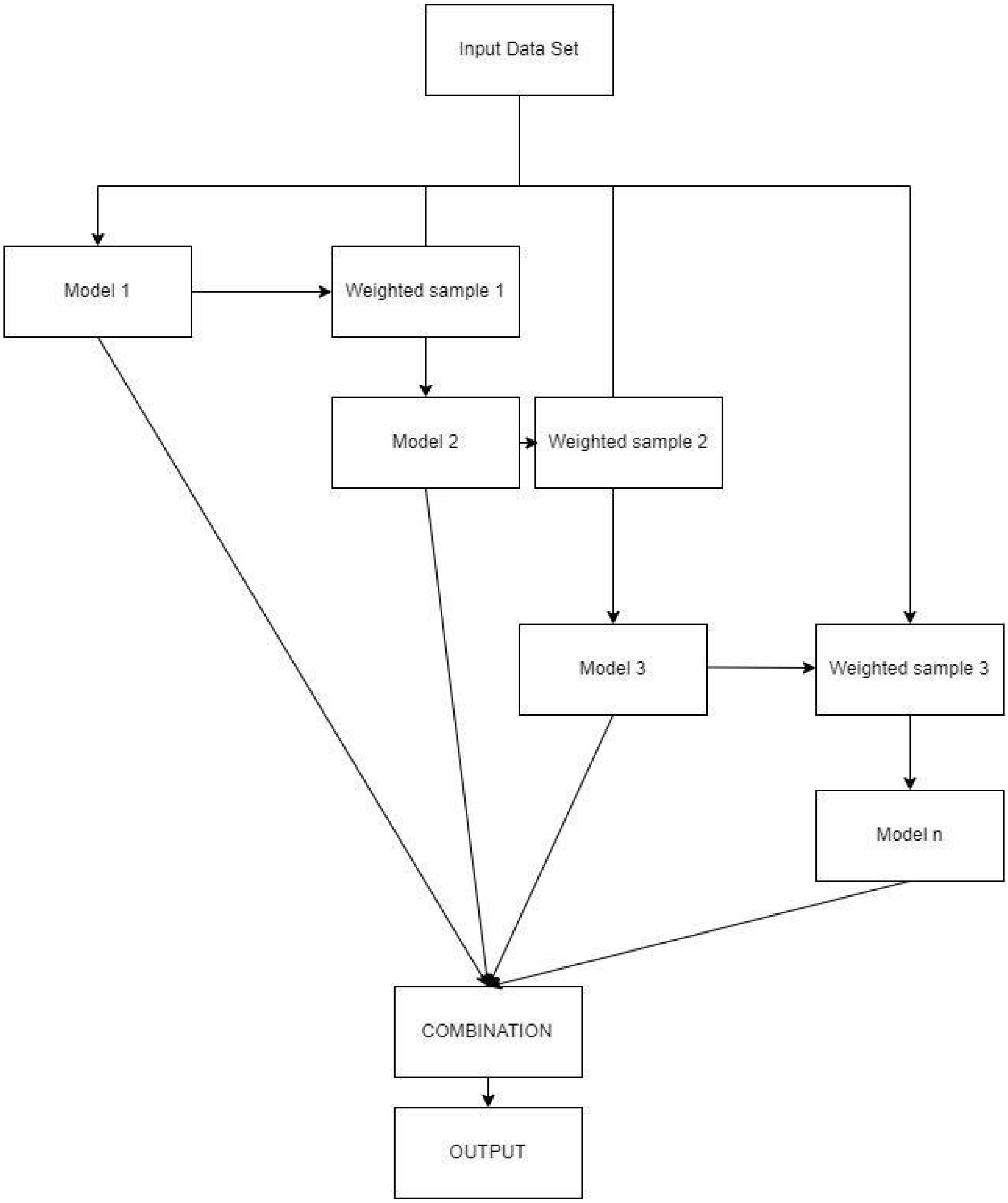


Fig 3.4.1.2 Boosting Ensemble Learning

3.4.1.3 STACKING ENSEMBLE LEARNING

Stacking is an ensemble machine learning algorithm that learns how to best combine the predictions from multiple well-performing machine learning models. The architecture of a stacking model involves two or more base models, often referred to as level-0 models and a metamodel that combines the predictions of the base models referred to as a level-1 model.

1) Level-0 Models (Base-Models): Models fit on the training data and whose predictions are compiled.
2) Level-1 Model (Meta-Model): Model that learns how to best combine the predictions of the base models.

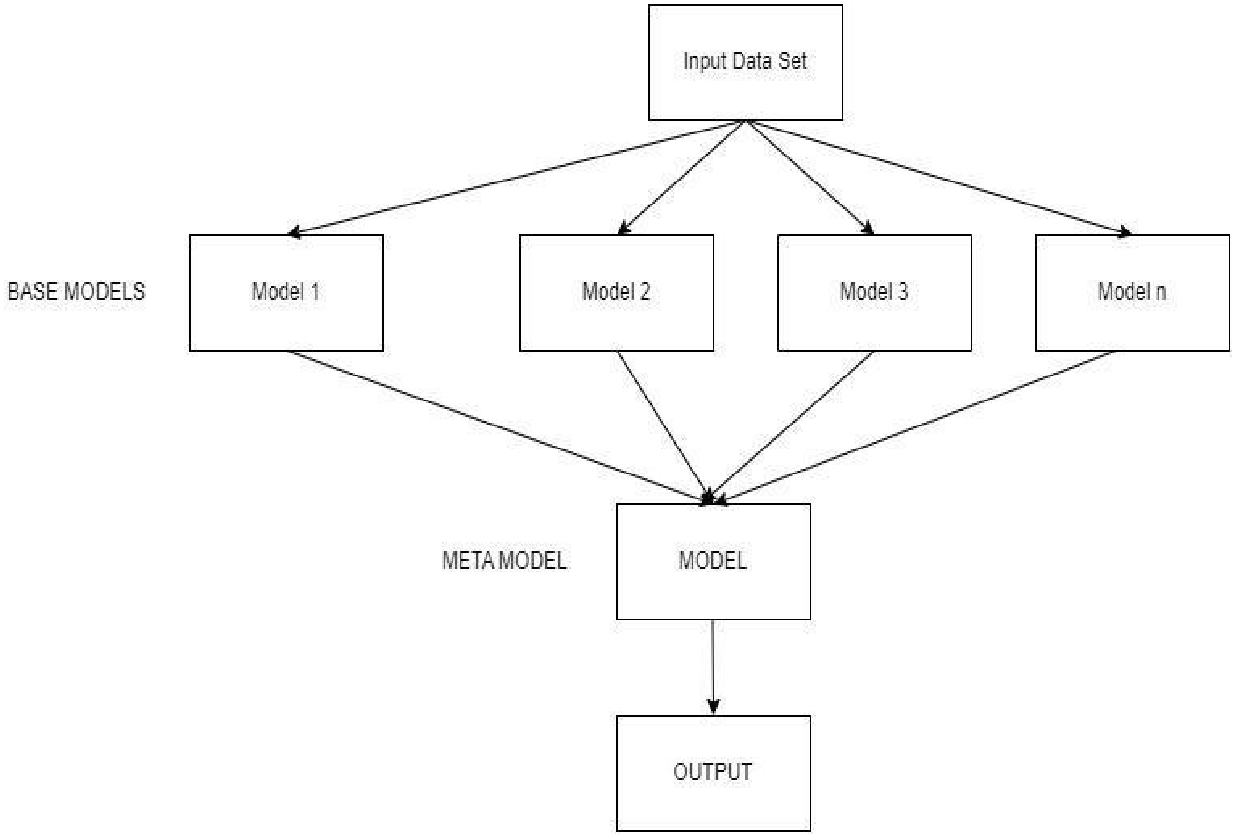


Fig 3.4.1.3 Stacking Ensemble Learning

3.4.2 SMOTE technique

Synthetic Minority Over-sampling Technique (SMOTE) is a technique used to address the class imbalance in the data. It generates synthetic samples of the minority class by interpolating between the feature vectors of the minority class samples. SMOTE can be used with various machine learning algorithms, including ensemble learning, to improve the predictive performance of the minority class.

Class imbalance occurs when the number of samples in one class is significantly smaller than the number of samples in the other class. In software defect prediction, the majority class is usually non-defective instances, while the minority class is the defective instances. Class imbalance can lead to biased model performance and reduced accuracy for the minority class.

The SMOTE algorithm works as follows:

- You draw a random sample from the minority class.
- For the observations in this sample, you will identify the k nearest neighbors.
- You will then take one of those neighbors and identify the vector between the current data point and the selected neighbor.
- You multiply the vector by a random number between 0 and 1.
- To obtain the synthetic data point, you add this to the current data point.

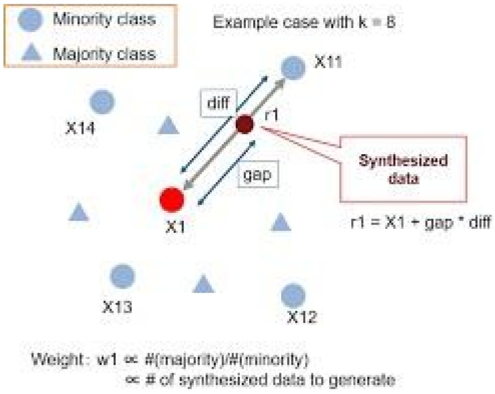
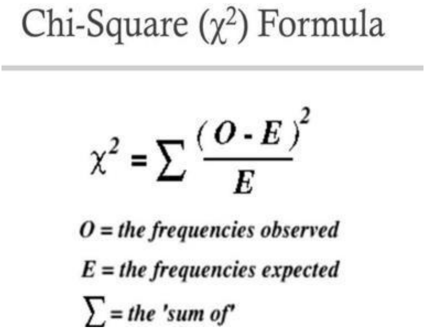


Fig 3.4.2.1 SMOTE process

3.4.3 Feature Selection

3.4.3.1 CHI SQUARE method

A chi-square test is used in statistics to test the independence of two events. Given the data of two variables, we can get observed count O and expected count E. Chi-Square measures how expected count E and observed count O deviates each other



Chi-Square (χ^2) Formula

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

O = the frequencies observed

E = the frequencies expected

$\sum$ *= the 'sum of'*

Fig 3.4.3.1 CHI SQUARE method

3.4.3.2 Sequence Backward Selection

Backward elimination is a feature selection technique while building a machine learning model. It is used to remove those features that do not have a significant effect on prediction the output

4. IMPLEMENTATION OF THE PROPOSED SYSTEM

The 4th chapter describes the implementation of the proposed system. It describes the various algorithms, datasets, and libraries used in the proposed system.

4.1 ALGORITHMS

Machine learning algorithms are a set of computational methods and techniques that enable computers to learn from data without being explicitly programmed. They are used to develop models that can identify patterns and relationships in data, and make predictions or decisions based on that information. There are many different types of machine learning algorithms, each designed to solve specific problems.

Supervised Learning Algorithms: These algorithms learn from labeled data, where the input features are associated with known output labels. They are used for classification tasks, where the goal is to predict a categorical label, or regression tasks, where the goal is to predict a continuous value.

Unsupervised Learning Algorithms: These algorithms learn from unlabeled data, where there are no predefined output labels. They are used for clustering tasks, where the goal is to group similar data points together, or dimensionality reduction tasks, where the goal is to reduce the number of input features.

Semi-supervised Learning Algorithms: These algorithms learn from a combination of labeled and unlabeled data. They are useful when labeled data is scarce or expensive to obtain.
Reinforcement Learning Algorithms: These algorithms learn through trial and error by interacting with an environment. They are used in tasks where an agent must learn to make a sequence of decisions to maximize a reward signal.

Deep Learning Algorithms: These algorithms are a subset of machine learning algorithms that use neural networks to learn from data. They are used for tasks such as image and speech recognition, natural language processing, and autonomous driving.

Making predictions and decisions based on data requires using machine learning algorithms, which are effective tools. Depending on the task at hand and the type of data being used, the best method can be chosen.

4.1.1 Random Forest Algorithm

Random forest is a machine learning algorithm that is used for both classification and regression tasks. It is an ensemble method that combines multiple decision trees to make a final prediction. The key idea behind random forest is to reduce overfitting by creating many random variations of the decision trees.

The algorithm works as follows:

- Random subsets of the training data are sampled with replacements to create multiple subsets, also known as bootstrap samples.
- For each bootstrap sample, a decision tree is built using a random subset of the available features. This helps to decorrelate the trees and make them more diverse.
- The splitting criterion for each node of the decision tree is chosen by finding the best split among a random subset of the available features.
- The process of building decision trees is repeated many times to create a forest of trees.
- To make a prediction for a new data point, the algorithm passes the data point through each decision tree in the forest and calculates the average (in case of regression) or the majority vote (in case of classification) of the individual tree predictions.

When compared to other machine learning methods, the random forest provides a number of benefits. Some of these are:

- Random forest is a powerful and flexible algorithm that can be used for a wide range of tasks, including classification, regression, and outlier detection.
- Random forest is resistant to overfitting, as the combination of multiple decision trees reduces the effect of individual trees that may overfit the data.

- The random forest can handle high-dimensional data with many features, as it only uses a random subset of the available features at each node of the decision tree.
- The random forest can provide information about the relative importance of the input features, which can be useful for feature selection and data visualization.

The random forest is a popular and effective machine-learning algorithm that is widely used in industry and research.

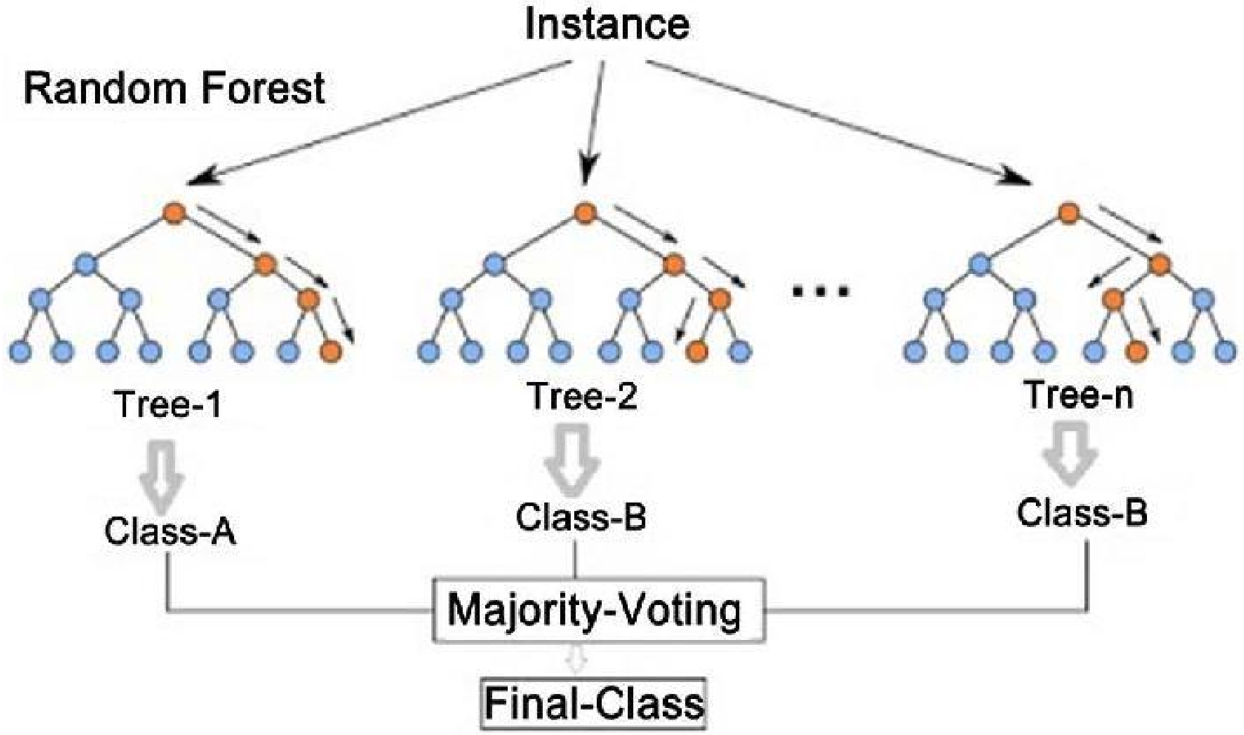


Fig 4.1.1.1 Random Forest Model

Fig 4.1.1.1 describes the random forest model used in the proposed system. So, a random forest algorithm works on building multiple decision trees and predicts the output for every decision tree and after that using the majority-voting algorithm it predicts the actual and final output of the model.

4.1.2 Logistic Regression Algorithm

Logistic regression is a statistical algorithm used in machine learning for binary classification tasks, where the goal is to predict a binary outcome (i.e., one of two possible classes) based on one or more input variables. It is a supervised learning algorithm that uses a logistic function to model the relationship between the input variables and the binary outcome.

Here's how the logistic regression algorithm works:

- Data Preparation: The first step is to prepare the data. This includes selecting the input variables and encoding the target variable (i.e., the binary outcome) as a numerical value (e.g., 0 or 1).
- Model Training: The next step is to train the logistic regression model. This involves estimating the coefficients of the logistic function that best fits the training data. The logistic function is a sigmoid-shaped curve that transforms the input variables into a probability score between 0 and 1.

- Model Testing: Once the model is trained, it is tested on a separate validation or test set. This is done to evaluate the accuracy and generalization performance of the model.

- Model Deployment: Finally, the model can be deployed to make predictions on new data points.

Here is the mathematical formulation of the logistic regression algorithm:
The logistic regression model takes the form of a sigmoid function.

$$P(y=1/X) = 1 / (1 + \exp(-z))$$

where:

- P(y=1/X) is the conditional probability of the binary outcome (y=1) given the input variables (X).

- exp() is the exponential function.
- z is a linear combination of the input variables (X) and their coefficients (W) plus a bias term (b):

$z = XW + b$

The coefficients (W) and the bias term (b) are learned during the training phase using a maximum likelihood estimation technique.

The logistic regression algorithm has several advantages:

- It is a simple and interpretable algorithm that is easy to implement and understand.
- It is computationally efficient and can handle large datasets with many input variables.
- It can handle both numerical and categorical input variables by encoding them as dummy variables.
- It can provide insight into the relative importance of the input variables by analyzing the magnitude and sign of the coefficients.
- It can be extended to handle multiclass classification tasks by using a one-vs-all or a softmax function.

4.1.3 Linear Regression Algorithm

Linear regression is a statistical algorithm used in machine learning for predicting a continuous output variable based on one or more input variables. It is a supervised learning algorithm that uses a linear function to model the relationship between the input and output variables.

Here's how the linear regression algorithm works:

- Data Preparation: The first step is to prepare the data. This includes selecting the input variables and the output variable, cleaning the data by handling missing values and outliers, and splitting the data into training and testing sets.
- Model Training: The next step is to train the linear regression model. This involves estimating the coefficients of the linear function that best fit the training data.
- Model Testing: Once the model is trained, it is tested on a separate validation or test set. This is done to evaluate the accuracy and generalization performance of the model.

- Model Deployment: Finally, the model can be deployed to make predictions on new data points.

Here is the mathematical formulation of the linear regression algorithm:
The linear regression model takes the form of a linear equation:

$$y = b0 + b1x1 + b2x2 + ... + bn*xn$$

where:

- y is the output variable to be predicted.
- x1, x2, ..., xn are the input variables.
- b0, b1, b2, ..., bn are the coefficients of the linear function.

The coefficients (b0, b1, b2, ..., bn) are learned during the training phase using a least-squares estimation technique. The goal is to minimize the sum of the squared differences between the predicted values of y and the actual values of y in the training set.

The linear regression algorithm has several advantages:

- It is a simple and interpretable algorithm that is easy to implement and understand.

- It can handle numerical and categorical input variables by encoding them as dummy variables.

- It can provide insight into the relative importance of the input variables by analyzing the magnitude and sign of the coefficients.

- It can be extended to handle multivariate linear regression using multiple input variables.

4.2 Dataset Description

We have taken the CM1 dataset of the promise repository. The dataset contains multiple features like lines of code, essential complexity, cyclomatic complexity, design complexity, volume, difficulty, intelligence, effort, etc where all the features are Mccabes and Halstead metrics.

These metrics are used to measure software complexity and the difficulty of understanding and maintaining software code.

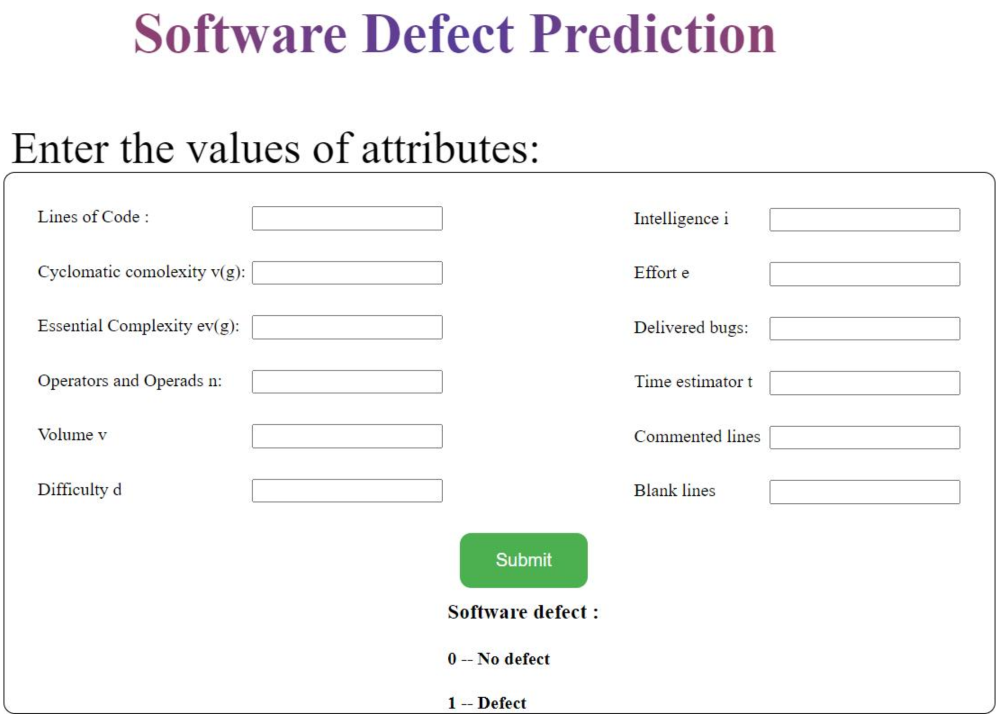
The McCabe metric, also known as Cyclomatic Complexity, measures the number of independent paths through a program's source code. It indicates the number of decision points and the complexity of the control flow in the program. Programs with higher McCabe values are more complex and harder to understand and maintain.

The Halstead metric measures software complexity in terms of the number of unique operators and operands used in the code. It takes into account the number of distinct operators and operands, as well as their occurrences, and calculates metrics such as program vocabulary, program length, volume, and difficulty. Programs with higher Halstead values are also more complex and maybe more error-prone.

Both McCabe and Halstead metrics are widely used in software engineering to help developers understand and evaluate the complexity of their code and identify potential areas for improvement

.

4.3 Implementation



Software Defect Prediction

Enter the values of attributes:

Lines of Code :		Intelligence i	
Cyclomatic comolexity v(g):		Effort e	
Essential Complexity ev(g):		Delivered bugs:	
Operators and Operads n:		Time estimator t	
Volume v		Commented lines	
Difficulty d		Blank lines	

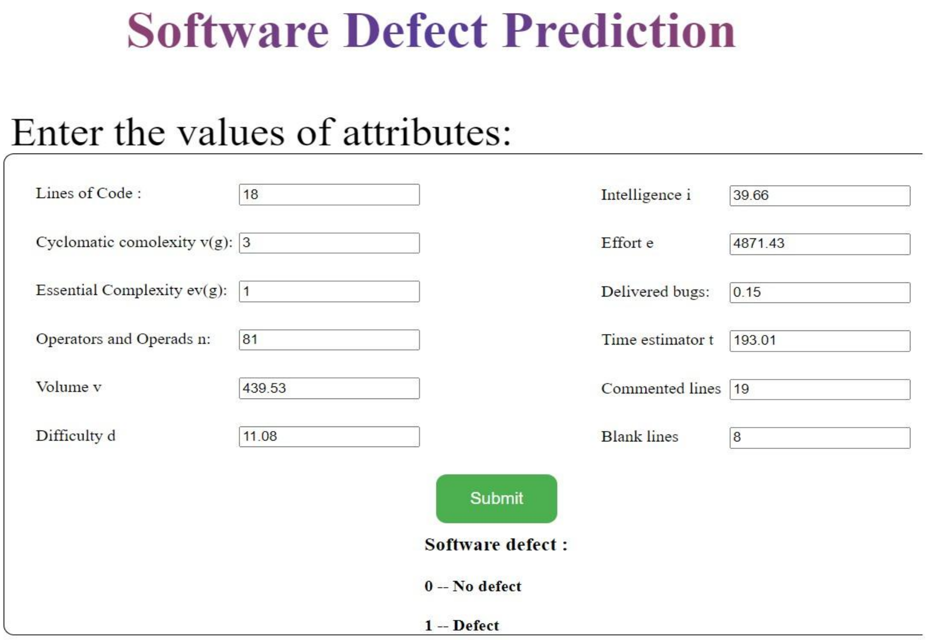
Software defect :

0 -- No defect

1 -- Defect

Fig 4.3.1: User Interface for inputing features

Fig 4.3.1 represents user interface for inputting 12 features which are selected in Feature selection techniques.



Software Defect Prediction

Enter the values of attributes:

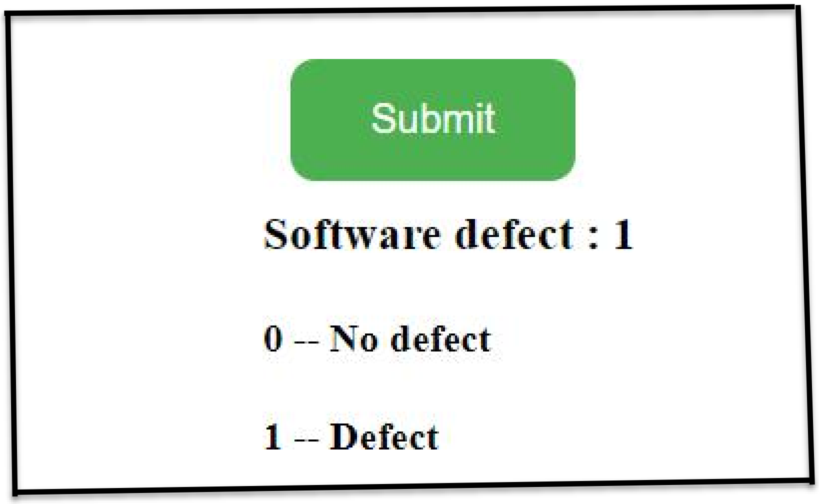
Lines of Code :	18	Intelligence i	39.66
Cyclomatic comolexity v(g):	3	Effort e	4871.43
Essential Complexity ev(g):	1	Delivered bugs:	0.15
Operators and Operads n:	81	Time estimator t	193.01
Volume v	439.53	Commented lines	19
Difficulty d	11.08	Blank lines	8

Software defect :

0 -- No defect

1 -- Defect

Fig 4.3.2: Submitting input features



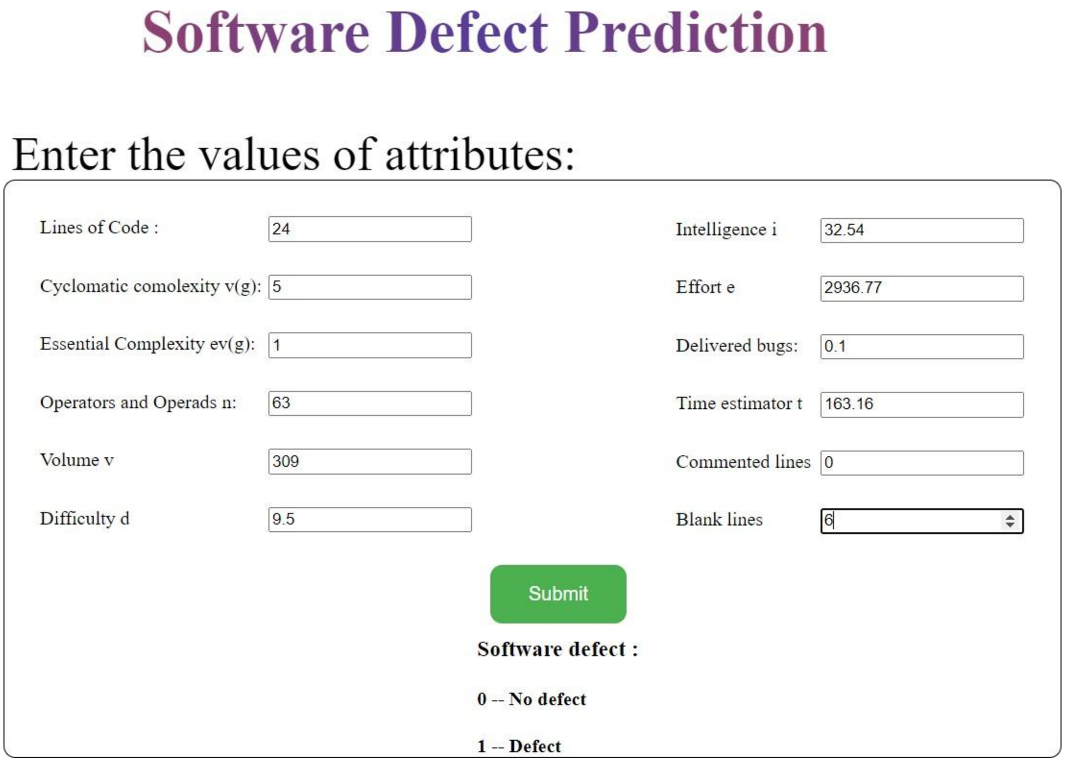
Software defect : 1

0 -- No defect

1 -- Defect

Fig 4.3.3: Output for the given features

From Fig 4.3.3 we can observe that, for given input features in Fig 4.3.2 the software has defect.



Software Defect Prediction

Enter the values of attributes:

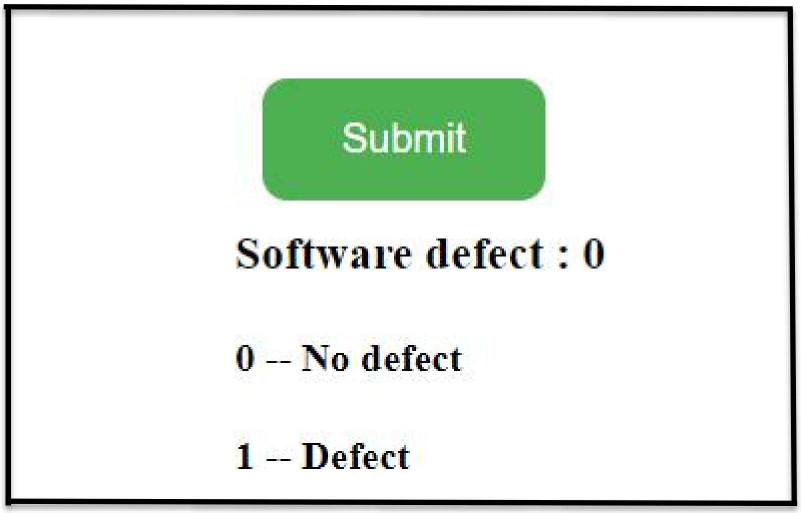
Lines of Code :	24	Intelligence i	32.54
Cyclomatic comolexity v(g):	5	Effort e	2936.77
Essential Complexity ev(g):	1	Delivered bugs:	0.1
Operators and Operads n:	63	Time estimator t	163.16
Volume v	309	Commented lines	0
Difficulty d	9.5	Blank lines	6

Software defect :

0 -- No defect

1 -- Defect

Fig 4.3.4 Submitting Features



Software defect : 0

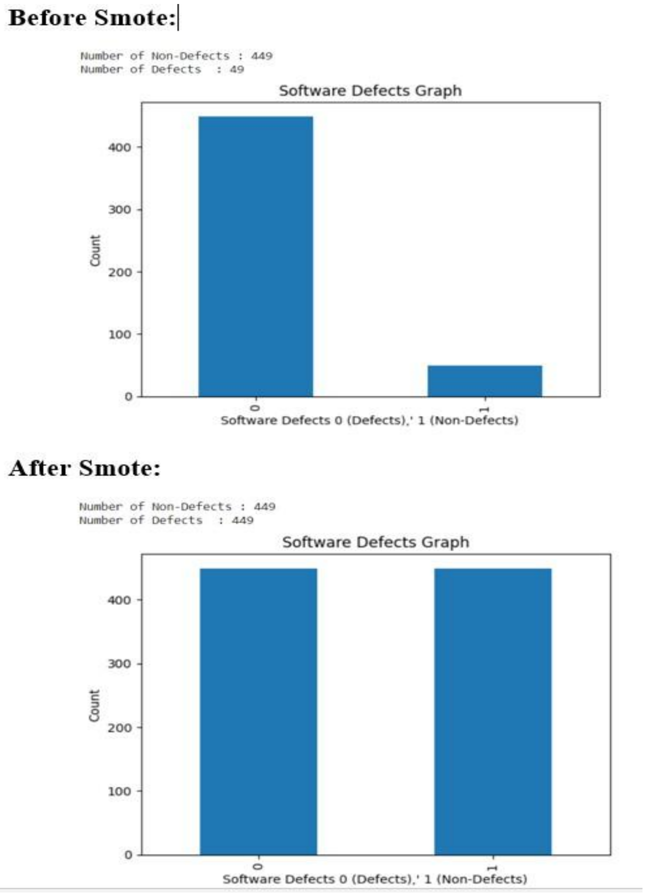
0 -- No defect

1 -- Defect

Fig 4.3.5 Output Of given Features

From Fig 4.3.5 we can observe that for given input features in 4.3.4 the software has no defect

5. Results and Analysis



Before Smote:

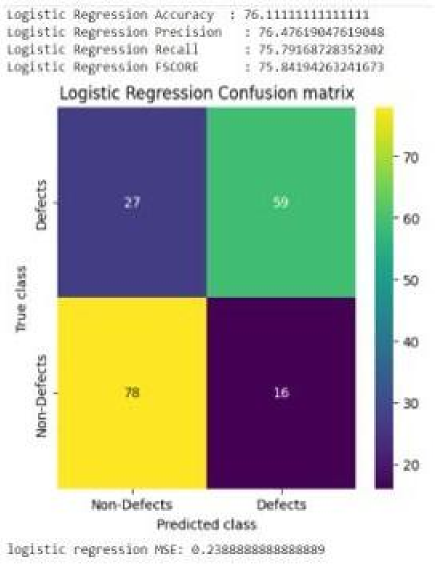
```
Number of Non-Defects : 449
Number of Defects  : 49
```

After Smote:

```
Number of Non-Defects : 449
Number of Defects  : 449
```

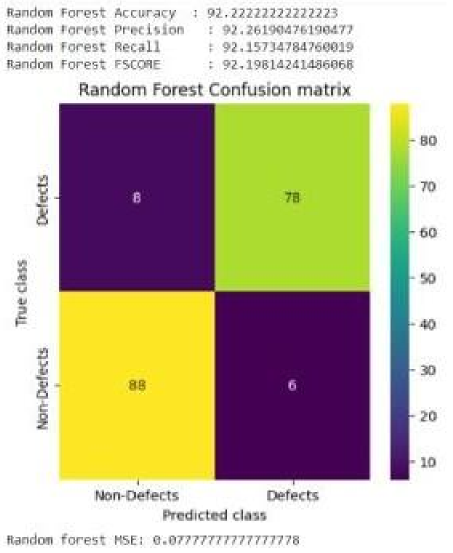
Fig 5.1: Smote process

This bar graph shows number of rows before and after SMOTE. Using feature selection Techniques, we selected 12 features out of 21.



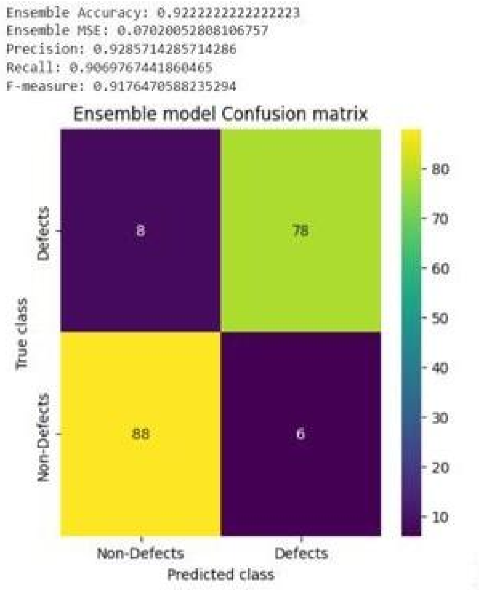
```
Logistic Regression Accuracy  : 76.11111111111111
Logistic Regression Precision   : 76.47619047619048
Logistic Regression Recall      : 75.79168728352302
Logistic Regression FSCORE      : 75.84194263241673
```

```
logistic regression MSE: 0.2388888888888889
```



```
Random Forest Accuracy  : 92.22222222222223
Random Forest Precision   : 92.26190476190477
Random Forest Recall      : 92.15734784760019
Random Forest FSCORE      : 92.19814241486068
```

```
Random forest MSE: 0.07777777777777778
```



```
Ensemble Accuracy: 0.9222222222222223
Ensemble MSE: 0.07020052808106757
Precision: 0.9285714285714286
Recall: 0.9069767441860465
F-measure: 0.9176470588235294
```

Fig 5.2: Calculation metrics

From above Confusion metrics logistic has 76.11 accuracy, Random Forest has 92.2 and finally ensemble model has 92.2 and other performance are also greater for ensemble model.

6. CONCLUSION AND FUTURE SCOPE

Software defect prediction based on SMOTE and Feature selection. This method is tested from the perspective of combining data processing and algorithms. The experimental results shows that it has certain degree of handling software defect data sample distribution imbalance and feature redundancy. The prediction performance of the ensemble learning algorithm is better than other single classifier algorithms, and it has the advantages of higher precision and recall rate.

In future research, further consideration will be given to the combination of different sampling and feature selection methods, a well as application of other classic integrated learning algorithms in software defect prediction, so as to improve software defect prediction performance.

7. REFERENCES

[1]. Liu Yang; Zhen Li, Dongsheng Wang, Hong Miao, Zhaobin Wang “Software Defects Prediction Based on Hybrid Particle Swarm Optimization and Sparrow Search Algorithm “IEEE 2021

[2]. Steffen Herbold “On the Costs and Profit of Software Defect Prediction” IEEE 2021

[3]. Zainab S. Alharthi , Abdullah Alsaeedi , Wael M.S. Yafooz “Software Defect Prediction Approaches: A Review” IEEE 2021

[4]. Mayur Jagtap; Praveen Katragadda; Pooja Satelkar “Software Reliability: Development of Software Defect Prediction Models Using Advanced Techniques IEEE2022.

[5]. S. Huda et al., "A Framework for Software Defect Prediction and Metric Selection," in IEEE Access, vol. 6, pp. 2844-2858, 2018, doi: 10.1109/ACCESS.2017.2785445.

[6]. J. Zheng, X. Wang, D. Wei, B. Chen and Y. Shao, "A Novel Imbalanced Ensemble Learning in Software Defect Predication," in IEEE Access, vol. 9, pp. 86855-86868, 2021, doi: 10.1109/ACCESS.2021.3072682

[7] Z. Sun, Q. Song and X. Zhu, "Using Coding-Based Ensemble Learning to Improve Software Defect Prediction," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 6, pp. 1806-1817, Nov. 2012, doi: 10.1109/TSMCC.2012.2226152

8. APPENDIX

Front-end

Home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Software Defect Prediction</title>
    <style type="text/css">
        header h1 {
          font-size: 50px;
          font-weight: 600;
          background-image:radial-gradient(circle, #553c9a, #ee4b2b);
          color: transparent;
          background-clip: text;
          -webkit-background-clip: text;
          text-align: center;
        }

        .button {
              background-color: #4CAF50; /* Green */
              border: none;
              color:  white;  padding:
              15px 32px; text-align:
              center; text-
              decoration: none;
              display: inline-block;
              font-size: 16px;
              border-radius:10px;
        }
        div
        {
            padding:15px;
        }
        .main
        {
            border-radius: 10px;
            border:1px solid ;
            width: 850px;
            position: relative;
            left:350px;
            top:60px;
            height: 450px;

        }
        .move
        {
            position: absolute;
            left:220px;
        }
        .move2
        {
            position: absolute;
            left: 150px;
        }
        .subcontainer{ position:
            absolute; left:
            530px; top:1px;
        }
   </style>

</head>
<body style="background-color: #ffffff;">
    <form action="{% url 'home' %}" method="POST">
        {% csrf_token %}

        <header>
            <h1>Software Defect Prediction</h1>
        </header>
         <div style="position: absolute;left: 350px; font-size:40px;">Enter the
values of attributes:</div>
        <div class='main'>

            <div>
                <label for="input1" >Lines of Code : </label>
                <input type="number" id="input1" name="input1" min="0" step="any"
class="move">
            </div>
            <div>
                <label for="input2">Cyclomatic comolexity v(g): </label>
                <input type="number" id="input2" name="input2" min="0" step="any"
class="move">
            </div>
            <div>
                <label for="input3">Essential Complexity ev(g):</label>
                <input type="number" id="input3" name="input3" min="0" step="any"
class="move">
            </div>
            <div>
                <label for="input4">Operators and Operads n: </label>
                <input type="number" id="input4" name="input4" min="0" step="any"
class="move">
            </div>
            <div>
                <label for="input5">Volume v</label>
                <input type="number" id="input5" name="input5" min="0" step="any"
class="move">
            </div>
            <div>
                <label for="input6">Difficulty  d</label>
                <input type="number" id="input6" name="input6" min="0" step="any"
class="move">
            </div>
            <div class='subcontainer'>
                <div>
                    <label for="input7">Intelligence i</label>
                    <input type="number" id="input7" name="input7" min="0"
step="any" class="move2">
                </div>
                <div>
                    <label for="input8">Effort e</label>
                    <input type="number" id="input8" name="input8" min="0"
step="any" class="move2">
                </div>
                <div>
                    <label for="input9">Delivered bugs:</label>
                    <input type="number" id="input9" name="input9" min="0"
step="any" class="move2">
                </div>
                <div>
                    <label for="input10">Time estimator t</label>
                    <input type="number" id="input10" name="input10" min="0"
step="any" class="move2">
                </div>
                <div>
                    <label for="input11">Commented lines</label>
                    <input type="number" id="input11" name="input11" min="0"
step="any" class="move2">
                </div>
                <div>
```

```
                    <label for="input12">Blank lines</label>
                    <input type="number" id="input12" name="input12" min="0"
step="any" class="move2">
                </div>
            </div>
            <div style="position: absolute;left:390px;">
                <input type="submit" class='button'>
            </div>
    </div>
    </form>
    <div style="position:relative;  left: 730px; top: -75px;">
        <h3>Software defect : {{result}}</h3>
        <h4>0 -- No defect</h4>
        <h4>1 -- Defect</h4>
    </div>
</body>
</html>
```

Views.py:

```
from django.shortcuts import render
import numpy as np
# Create your views here.
import pickle

def home(request):
    if request.method == "POST":
        input1 = request.POST["input1"]
        input2 = request.POST["input2"]
        input3 = request.POST["input3"]
        input4 = request.POST["input4"]
        input5 = request.POST["input5"]
        input6 = request.POST["input6"]
        input7 = request.POST["input7"]
        input8 = request.POST["input8"]
        input9 = request.POST["input9"]
        input10 = request.POST["input10"]
        input11 = request.POST["input11"]
        input12 = request.POST["input12"]

        input_features=[float(input1),float(input2),float(input3),float(input4),f
loat(input5),float(input6),float(input7),float(input8),float(input9),float(input1
0),float(input11),float(input12)]
        input_features=np.array(input_features).reshape(1,-1)
        with
open(r'C:\Users\dell\Desktop\nivedh\majorproject\softwaredefect\model1.pkl' ,
'rb') as file:
            rf = pickle.load(file)
        with
open(r'C:\Users\dell\Desktop\nivedh\majorproject\softwaredefect\model2.pkl' ,
'rb') as file:
            lr = pickle.load(file)
        with
open(r'C:\Users\dell\Desktop\nivedh\majorproject\softwaredefect\model3.pkl' ,
'rb') as file:
            ensemble_model = pickle.load(file)
        y_pred= rf.predict(input_features)
        y_pred2= lr.predict(input_features)
        input_linear=[y_pred[0],y_pred2[0]]
        input_linear=np.array(input_linear).reshape(1,-1)
        y_final = ensemble_model.predict(input_linear)
        result = int(y_final.round())
        print(y_final)
        #result =
ensemble_model.predict([[input1,input2,input3,input4,input5,input6,input7,input8,
input9,input10,input11,input12,input13,input14,]])

        return render(request,'home.html',{'result':result})
    return render(request,'home.html')
```

asgi.py:

```
"""
ASGI config for majorproject project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'majorproject.settings')

application = get_asgi_application()
```

settings.py:

```
"""
Django settings for majorproject project.

Generated by 'django-admin startproject' using Django 4.2.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-(cwlf5=l&@0(2_jm+t7%ml87q!r+(2!_yd$bke_@@5bv3$o%-o'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'softwaredefect.apps.SoftwaredefectConfig'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'majorproject.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors':
                [ 'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'majorproject.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Urls.py:

```
"""
URL configuration for majorproject project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('softwaredefect.urls'))
]
```

Wsgi.py:

```
"""
WSGI config for majorproject project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/
"""
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'majorproject.settings')

application = get_wsgi_application()
```

apps..py:

```
from django.apps import AppConfig
class SoftwaredefectConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'softwaredefect'
```

manage.py:

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'majorproject.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__== '__main__':
    main()
```

majorproject.py

```
pip install mlxtend


import pandas as pd
import numpy as np


pip install --upgrade mlxtend



# Attribute Information:
#  1. loc             : numeric % McCabe's line count of code
# 2. v(g)             : numeric % McCabe "cyclomatic complexity"
# 3. ev(g)            : numeric % McCabe "essential complexity"
# 4. iv(g)            : numeric % McCabe "design complexity"
#  5. n              : numeric % Halstead total operators + operands
#  6. v              : numeric % Halstead "volume"
#  7. l              : numeric % Halstead "program length"
#  8. d              : numeric % Halstead "difficulty"
#  9. i              : numeric % Halstead "intelligence"
# 10. e              : numeric % Halstead "effort"
# 11. b              : numeric % Halstead
# 12. t              : numeric % Halstead's time estimator
# 13. lOCode         : numeric % Halstead's line count
# 14. lOComment      : numeric % Halstead's count of lines of comments
# 15. lOBlank        : numeric % Halstead's count of blank lines
# 16. lOCodeAndComment: numeric
# 17. uniq_Op        : numeric % unique operators
# 18. uniq_Opnd      : numeric % unique operands
# 19. total_Op       : numeric % total operators
# 20. total_Opnd     : numeric % total operands
# 21: branchCount    : numeric % of the flow graph
# 22. defects        : {false,true} % module has/has not one or more###
data = pd.read_csv('projectdataset.csv')
data



import matplotlib.pyplot as plt

data = pd.read_csv('projectdataset.csv')
label = data.groupby('defects').size()
label.plot(kind="bar")
plt.xlabel("Software Defects 0 (Defects),' 1 (Non-Defects)")
plt.ylabel("Count")
plt.title("Software Defects Graph")
```

```
unique, count = np.unique(data['defects'], return_counts=True)
print("Number of Non-Defects : "+str(count[0]))
print("Number of Defects  : "+str(count[1]))
plt.show()
plt.savefig("dataset_before_smote.jpeg")

from imblearn.over_sampling import SMOTE
import pandas as pd

# Load the dataset
#df = pd.read_csv("your_dataset.csv")

# Separate the features and target variable
# X = X_selected_df.iloc[:, :-1]
# y = X_selected_df.iloc[:, -1]
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
# Print the class distribution before oversampling
#print("Before oversampling:\n", y.value_counts())

# Apply SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Convert the resampled data to a pandas dataframe
df_resampled = pd.concat([pd.DataFrame(X_resampled), pd.DataFrame(y_resampled)], axis=1)
df_resampled.columns = data.columns
data=df_resampled
original=data
data
```

```
label = data.groupby('defects').size()
label.plot(kind="bar")
plt.xlabel("Software Defects 0 (Defects),' 1 (Non-Defects)")
plt.ylabel("Count")
plt.title("Software Defects Graph")
unique, count = np.unique(data['defects'], return_counts=True)
print("Number of Non-Defects : "+str(count[0]))
print("Number of Defects  : "+str(count[1]))
plt.show()
plt.savefig("dataset_after_smote.png")
# Print the class distribution after oversampling
print("After oversampling:\n", df_resampled.iloc[:,-1].value_counts())

import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# data = pd.read_csv('dataset.csv')

# Separate the features and the target variable
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
# Apply chi-square feature selection
skb = SelectKBest(chi2, k=15)
X_chi2 = skb.fit_transform(X, y)
```

```
# Apply SBS feature selection
knn = KNeighborsClassifier(n_neighbors=5)
sbs = SFS(knn,
        k_features=12,
        forward=False,
        floating=False,
        verbose=2,
        scoring='accuracy',
        cv=5)
sbs.fit(X_chi2, y)
X_sbs = X.iloc[:, list(sbs.k_feature_idx_)]

# Print the selected features
#print("Chi-Square selected features: ", X.columns[skb.get_support()])
print("SBS selected features: ", X_sbs.columns)




print("Number of features beforefeature selection:",original.shape[1]-1)
selected_feature_names = X_sbs.columns
X_selected_df = pd.DataFrame(X_sbs, columns=selected_feature_names)
print("Number of features after feature selection:",X_selected_df.shape[1])

# Attribute Information:
#  loc             : numeric % McCabe's line count of code
#  v(g)            : numeric % McCabe "cyclomatic complexity"
#  ev(g)           : numeric % McCabe "essential complexity"
#  n               : numeric % Halstead total operators + operands
#  v               : numeric % Halstead "volume"
#  d               : numeric % Halstead "difficulty"
#  i                : numeric % Halstead "intelligence"
#  e                : numeric % Halstead "effort"
#  b                : numeric % Halstead
#  t                : numeric % Halstead's time estimator
#  lOComment        : numeric % Halstead's count of lines of comments
#  lOBlank          : numeric % Halstead's count of blank lines
#  defects          : {0,1} % module has/has not one or more###
data = X_selected_df
data

data = pd.concat([pd.DataFrame(data), pd.DataFrame(y)], axis=1)

## train test split
['loc','v(g)','ev(g)','iv(g)','n','v','l','d','i','e','b','t','locode','locomment','loblank','locCodeAndComment','uniq_Op','uniq_Opnd','total_Op','total_Opnd','branchCount']

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(data[['loc','v(g)','ev(g)','n','v','d','i','e','b','t','locomment','loblank']],data['defects'],test_size= 0.2)

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
import seaborn as sns

precision = []
recall = []
```

```
fscore = []
accuracy = []


def calculateMetrics(algorithm, predict, testY):
    testY = testY.astype('int')
    predict = predict.astype('int')
    p = precision_score(testY, predict,average='macro') * 100
    r = recall_score(testY, predict,average='macro') * 100
    f = f1_score(testY, predict,average='macro') * 100
    a = accuracy_score(testY,predict)*100
    print()
    print(algorithm+' Accuracy  : '+str(a))
    print(algorithm+' Precision   : '+str(p))
    print(algorithm+' Recall       : '+str(r))
    print(algorithm+' FSCORE      : '+str(f))
    accuracy.append(a)   precision.append(p)
    recall.append(r)
    fscore.append(f)
    labels = ['Non-Defects', 'Defects']
    conf_matrix = confusion_matrix(testY, predict)
    plt.figure(figsize =(5, 5))
     ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True,
     cmap="viridis" ,fmt ="g");
    ax.set_ylim([0,len(labels)])
    plt.title(algorithm+" Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()
```

```
#random forest classifier

rf=RandomForestClassifier()
rf.fit(x_train,y_train)
rf_prob=rf.predict(x_test)
calculateMetrics("Random Forest",rf_prob,y_test)
rfac_mse = mean_squared_error(y_test, rf_prob.astype(int))
print("Random forest MSE:", rfac_mse)

#logistic regression classifier
lr = LogisticRegression(random_state=42)
lr.fit(x_train, y_train)
lr_prob = lr.predict(x_test)
calculateMetrics("Logistic Regression", lr_prob, y_test)
log_mse = mean_squared_error(y_test, lr_prob.astype(int))
print("logistic regression MSE:", log_mse)

from sklearn.metrics import confusion_matrix

#combining rf and log using linear
X_combined = np.column_stack((rf_prob, lr_prob))
# Fit a Linear Regression model on the combined feature matrix to predict the target variable on the test data
reg = LinearRegression()
reg.fit(X_combined, y_test)
y_pred = reg.predict(X_combined)
from sklearn.metrics import accuracy_score, mean_squared_error
ensemble_acc = accuracy_score(y_test, y_pred.round())
ensemble_mse = mean_squared_error(y_test, y_pred)
print("Ensemble Accuracy:", ensemble_acc)
print("Ensemble MSE:", ensemble_mse)
```

```
# print("Accuracy:", accuracy_score(y_test, y_pred))
# print("Precision:", precision_score(y_test, y_pred))
# print("Recall:", recall_score(y_test, y_pred))
# print("F1 score:", f1_score(y_test, y_pred))
# Step 1: Calculate the confusion matrix
y_pred_binary = (y_pred >= 0.5).astype(int)
ensemble_cm = confusion_matrix(y_test, y_pred_binary)

# Step 2: Calculate precision, recall, and F-measure
tp = ensemble_cm[1][1]
fp = ensemble_cm[0][1]
fn = ensemble_cm[1][0]

precision = tp / (tp + fp)
recall = tp / (tp + fn)
f_measure = 2 * (precision * recall) / (precision + recall)

print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)


labels = ['Non-Defects', 'Defects']
conf_matrix = ensemble_cm
plt.figure(figsize =(5, 5))
ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True,
cmap="viridis" ,fmt ="g");
ax.set_ylim([0,len(labels)])
plt.title("Ensemble model Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
```

```
print("Ensemble Accuracy:", ensemble_acc)
print("Ensemble MSE:", ensemble_mse)
print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)
plt.show()

import pickle
with open('model1.pkl','wb') as file:
  pickle.dump(rf,file)
with open('model2.pkl','wb') as file:
  pickle.dump(lr,file)
with open('model3.pkl','wb') as file:
  pickle.dump(reg,file)
```