

*A project report on*

# **SOFTWARE DEFECT PREDICTION USING MACHINE LEARNING ALGORITHMS**

*Submitted in partial fulfillment for the award of the degree of*

## **Bachelor of Technology in Computer Science and Engineering**

*by*

**SAMARTH SINHA (19BCE1670)**



**VIT<sup>®</sup>**

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

**CHENNAI**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

April, 2023

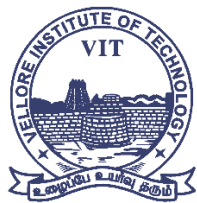
# **SOFTWARE DEFECT PREDICTION USING MACHINE LEARNING ALGORITHMS**

*Submitted in partial fulfillment for the award of the degree of*

## **Bachelor of Technology in Computer Science and Engineering**

*By*

**SAMARTH SINHA (19BCE1670)**



**VIT<sup>®</sup>**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

April, 2023



# VIT<sup>®</sup>

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

### DECLARATION

I here by declare that the thesis entitled “Software Defect Prediction using Machine Learning Algorithm” submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Vellore Institute of Technology, Chennai, is a record of bonafide work carried out by me under the supervision of Guide Name

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date:24-04-2023

Signature of the Candidate



# VIT<sup>®</sup>

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

### School of Computer Science and Engineering

## CERTIFICATE

This is to certify that the report entitled “**Software Defect Prediction using Machine learning Algorithm**” is prepared and submitted by **Samarth Sinha (19BCE1670)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** programme is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:

Name: Dr.Muthukumaran K

Date: 24-04-2023

Signature of the Examiner 1

Name:

Date:

Signature of the Examiner 2

Name:

Date:

Approved by the Head of Department  
**B. Tech. CSE**

Name: Dr. Nithyanandam P

Date: 24 – 04 – 2023

(Seal of SCOPE)

## **ABSTRACT**

In software engineering, predicting software bugs is a critical attempt that seeks to find potentially problematic modules early in the software development life cycle. In terms of foreseeing errors in software, machine learning algorithms have produced positive findings. An abstract of a study investigating the effectiveness of various machine learning methods for software defect prediction is presented in this paper. The performance of various algorithms, including as decision trees, random forests, support vector machines, and artificial neural networks, is trained on and assessed using a number of publicly accessible datasets.

The findings shows that machine learning algorithms are highly successful in predicting software bugs, and that ensemble techniques like random forests may enhance performance. Insights from the study can be used by software practitioners and engineers to select the best machine learning algorithms for software defect prediction, which can save development costs and raise software quality.

In order to solve the problems of traditional SVM classifier for software defect prediction, this paper proposes a novel dynamic SVM method based on improved cost-sensitive SVM (CSSVM) which is optimized by the Genetic Algorithm (GA).

## ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. K Muthukumaran, Senior Assistant Professor, SCOPE, Vellore Institute of Technology, Chennai, for his/her constant guidance, continual encouragement, understanding; more than all, he/she taught me patience in my endeavor. My association with him / her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of <area>.

It is with gratitude that I would like to extend thanks to our honorable Chancellor, Dr. G. Viswanathan, Vice Presidents, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan and Mr. G V Selvam, Assistant Vice-President, Ms. Kadhambari S. Viswanathan, Vice-Chancellor, Dr. Rambabu Kodali, Pro-Vice Chancellor, Dr. V. S. Kanchana Bhaaskaran and Additional Registrar, Dr. P.K.Manoharan for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dean, Dr. Ganesan R, Associate Dean Academics, Dr. Parvathi R and Associate Dean Research, Dr. Geetha S, SCOPE, Vellore Institute of Technology, Chennai, for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Nithyanandam P, Head of the Department, Project Coordinators, Dr. Abdul Quadir Md, Dr. Priyadarshini R and Dr. Padmavathy T V, B. Tech. Computer Science and Engineering, SCOPE, Vellore Institute of Technology, Chennai, for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staff at Vellore Institute of Technology, Chennai, who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: 24-04-2023  
**Sinha**

Name of the student: **Samarth**

# CONTENTS

<b>CONTENTS.....</b>	<b>iv</b>
<b>LIST OF FIGURES.....</b>	<b>ix</b>
<b>LIST OF TABLES.....</b>	<b>xi</b>
<b>LIST OF ACRONYMS.....</b>	<b>xii</b>
<b>CHAPTER 1</b>	
<b>INTRODUCTION</b>	
1.1 INTRODUCTION.....	5
1.2 LITERATURE SURVEY.....	7
1.3 PROBLEM STATEMENT.....	9
<b>CHAPTER 2</b>	
<b>Technologies used</b>	
2.2 Technologies Used.....	10
2.3 Development Methodologies.....	14
<b>CHAPTER 3</b>	
<b>Proposed System</b>	
3.1 Proposed System.....	15
3.2 System Design and Architecture.....	18
3.3 Algorithm Used.....	20
3.4 About Datasets.....	24
3.5 System Modules.....	25
3.6 Sequence Diagram for Modules.....	26
<b>CHAPTER 4</b>	
<b>Implementation</b>	
4.1 Major Libraries Used.....	27
4.2 Dataset Split.....	28
<b>APPENDICES.....</b>	<b>29</b>
<b>CHAPTER 5</b>	
<b>Results</b>	
5.1 Metrics.....	44
5.2 Results and Evaluation.....	46
5.3 Key Learning.....	47
5.4 Steps Involved in WEKA TOOL.....	49
<b>CHAPTER 6</b>	
<b>Conclusion and Future Work</b>	
6.1 Conclusion and Future Work.....	53
<b>REFERENCES.....</b>	<b>54</b>

## LIST OF FIGURES

1.1 Software defect prediction Model.....	6
1.2 Main steps of Proposed Approach.....	16
1.3 GA-CSSVM framework method.....	17
1.4 Proposed System Architecture.....	19
1.5 Dataset Architecture.....	24
1.6 Sequence diagram for Genetic Algorithm.....	26
1.7 Histogram for Instances.....	26
1.8 Scatterplot for instances .....	30
1.9 Boxplot for Unique Operators.....	30
1.10Data Normalization.....	31
1.11Output of Root Mean Square value.....	32
1.12Output of Naïve Bayes function .....	32
1.13Summary of the predictions made by the classifier.....	33
1.14Output of Logistic regression .....	33
1.15Accuracy Score for Random Forest.....	35
1.16The accuracy output for Random Forest using Ensemble Learning Technique...	35
1.17Output Score for SVM algorithm.....	36
1.18Confusion matrix for CSSVM.....	37
1.19Performance of the Cost-Sensitive SVM model.....	38
1.20Output of Genetic Algorithm.....	38
1.21Graph of Best Cost VS Number of Generations.....	41
1.22Graph of Best Cost VS Number of Generations.....	42
1.23Graph of Best Cost VS Number of Generations.....	42
1.24Graph of Best Cost VS Number of Generations.....	42
1.25Graph of Best Cost VS Number of Generations.....	43
1.26Graph of Best Cost VS Number of Generations.....	43
1.27 Mutation rate VS Number of Generations.....	43
1.28Summary of WEKA TOOL for instances.....	48
1.29Classification Report on PC1 datasets.....	50
1.30GA classification on PC1 datasets.....	50
1.31 Classification Report on PC2 datasets.....	51
1.32 Clustered Instances.....	51
1.33GA classification on PC2 datasets.....	51
1.34 Classification Report on CM1 datasets.....	52
1.35Clustered Instances.....	52
1.36 GA classification on CM1 datasets.....	52



## **LIST OF TABLES:**

2.1 Comparison of Accuracy Score Different Machine Learning Algorithm	
.....	33

## **LIST OF ACRONYMS:**

SQA	Software Quality Assurance
SDP	Software Defect Prediction
SDLC	Software Development Lifecycle
RF	Random Forest
ML	Machine Learning
SVM	Support Vector Machine
SBSE	Search based Software Engineering
GBS	Gradient Based Search
MOGA	Multi-Objective Genetic Algorithm
FS	Feature Selection
CSSVM	Cost Effective SVM
NN	Neural network
ANN	Artificial Neural Network
NB	Naïve Bayes
GA	Genetic Algorithm
MLP	MultiLayer Perceptron

## **Chapter 1**

# **Introduction**

## **1.1 SOFTWARE DEFECT PREDICTION**

The software development process or phase is a structured process. It also refers to as the software development methodology or lifecycle, where it involves many phases, that are: requirements, analysis, design, testing, deployment and maintenance. The SDLC is exposed to bugs, the undetected defects will arise after the production or deployment phase. These defects could be because of syntax error, spelling error, wrong program semantics, bad code design, requirement, design or specification errors.

Software defects may cause a failure on the system where huge losses will be risen if the product was released before validation which is not acceptable. Customers satisfaction and acceptance is an important issue in the software industry world. Therefore, the high quality software has to be satisfied through achieving customers' requirements, product developing using the rated budget, time and the available resources. Software production with bugs will demand more effort and time for the rebuild process which is not good for the Software Quality Assurance in addition to customers' anger issues. SQA task consumes (40% - 90%) of the projects' budget. Thus, the prediction of the default prone modules is highly important issue, where the focus of the testing task will be on these parts. Therefore, a reduction of the QSA cost, time and resources will occur. A reduction on the required software testing resources will occur by directing these resources on the defective modules.

The earlier defect is discovered in software, the lower the developing cost is. The higher the software quality is, the lower the maintenance cost after software released is. The goal of software defect prediction is to improve software quality and reduce the cost and effort associated with testing and debugging software. By identifying potential defects early in the development process, developers can make changes to the code to prevent defects from occurring in the first place, or catch and fix them before they are released to the public.

With growing demand and technology, the software industry is rapidly evolving. Since humans do most of the software development, defects will inevitably occur. In general, defects can be defined as undesired or unacceptable deviations in software documents, programs, and data .Defects may exist in requirements analysis because the product manager misinterprets the customer's needs, and as a result, this defect will also carry on to the system design phase. Defects may also occur in the code due to inexperienced coders. Defects significantly impact software quality, such as increased software maintenance costs, especially in healthcare, and aerospace software defects can have serious consequences. If the fault is detected after deployment, it causes an overhead on the development team as they need to re-design some software modules, which increases the development costs. Defects are nightmares for reputed organizations. Their reputation is affected due to customer dissatisfaction and hence reduces its market share.

Therefore, software testing has become one of the main focuses of industrial research. With the rise in software development and software complexity, the number of defects has increased to the extent that traditional manual methods consume much time and become inefficient. The rise of machine learning has made automatic classification of defects a research hotspot. In this paper, we initially discuss software defects in detail and their different categories available in the literature and then discuss the manual classification methods proposed by various researchers. Finally, we present the analysis of the state of the art machine learning algorithms for automatic software detection. Therefore, SDP has become an important research topic in software engineering and testing in recent years.

The defect prediction is from the measure metadata from the software module, classification, and machine learning by regression and clustering method to find out the relationship between the defect metrics, establish the appropriate model to predict the distribution of whether new software modules contain defects or the prediction model of defects

Software systems being made today are complex hence the difficulty of producing software without defects is increasing. If producing software without defects is nearly impossible to achieve, then minimization of defects needs to be done. This is not easily achievable. It is important to detect the defects early in the development lifecycle of a software in order to avoid the cost losses. Furthermore, early detection will ensure that reliable, good quality and optimum cost software are being delivered to the customers. Various Object Oriented (OOS) software metrics which quantify structural characteristics of a software which include inheritance, cohesion etc. can be used for developing prediction models for detecting defective classes in a software. Past releases of a software can be used for collecting these metrics and further these metrics are used for developing the prediction models for later releases.

Early-stage detection of these program errors is essential in reducing the expense of the testing. The idea is to forecast the defects in program metrics from the source code. Many scholars have researched numerous approaches to software faulty forecasting, including mathematical analysis, expert estimates, and Machine Learning.

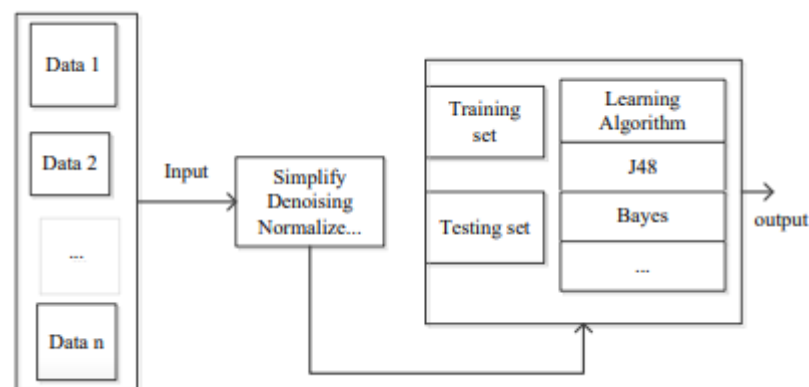


Figure 1 Software defect Prediction Model

Developing software applications without bugs is complicated. Different factors in the software development life cycle may lead to erroneous behavior of an application. If we could minimize these defects during the early development phase, then the final product will be an improved one with good quality and performance. We also find

undetected defects during the runtime of software applications. We need to model these software faults before and after updates to aid static and dynamic object-oriented metrics. It is an incredibly critical job to find the best range of measurements for the software defect estimation model due to the significant output discrepancies.

We have various Machine Learning algorithms to predict the defects in the software module automatically. Classification is the most useful type of Machine Learning algorithm. Classification means a predictive modeler problem in machine learning, where a label is expected for a particular example of input data. For classification from a modeling perspective, a training data set includes several examples of inputs and outputs from which to learn. Predictive classification modeling algorithms based on their success are checked. The accuracy of classification is a standard metric to decide the model's efficiency based on the expected labels of class. Some activities may require a prediction of the likelihood of membership rather than class labels. It makes an application more ambiguous or sees the consumer more clearly. The ROC curve is the most common diagnosis for evolutionary expectations. There are perhaps 4 major classifying tasks, including binary, multiclass, multi-label and imbalanced classification.

Binary classification applies to tasks with two class names. For example, the program module's source code is fault or not, email is a spam or not. Typically, binary classification tasks include a regular class and a separate class an odd one. Class 0 is assigned to the typical state's class and the category 1 is assigned to the class with the irregular state. Naive Bayes, Logistic Regression, Decision Tree, K-Nearest Neighbors, Support Vector Machine are common algorithms that can be used for binary classification.

In this paper, we initially discuss software defects in detail and their different categories available in the literature and then discuss the manual classification methods proposed by various researchers. Finally, we present the analysis of the state of the art machine learning algorithms for automatic software detection.

## 1.2 LITERATURE SURVEY

For the prediction of software errors, Ezgi Erturk et al. proposed the Adaptive Neuron Fuzzy Inference System (ANFIS). The PROMISE Software Engineering Repository is used for gathering data, and McCabe metrics were chosen because of their thorough examination of the programming effort. For the SVM, ANN, and ANFIS methods, the final results were 0.7795, 0.8685, and 0.8573, respectively.

Mie Thet Thwin in this paper, two kinds of neural network techniques are performed. The first focuses on predicting the number of defects in a class and the second on predicting the number of lines changed per class. Two neural network models are used those are Ward neural network and General Regression neural network (GRNN) and perform the analysis result on the NASA dataset.

David Gray et al. in this paper the main focus is on classification analysis rather than classification performance, it was decided to classify the training data rather than having some form of tester set. It involves a manual analysis of the predictions made by Support vector machine classifiers using data from the NASA Metrics Data Program repository. The purpose of this was to gain insight into how the classifiers were separating the training data.

Surndha Naidu in this paper focused on finding the total number of bugs in order to reduce the time and cost. Here for defect classification used ID3(Iterative Dichotomiser 3) algorithm. ID3 is an algorithm invented by Ross Quinlan used to generate a decision tree from a dataset. The defects were classified based on the five attribute values such as Volume, Program length, Difficulty, Effort and Time Estimator.

Martin Shepperd et al. presented a novel benchmark framework for software defect prediction. In the framework involves both evaluation and prediction. In the evaluation stage, different learning schemes are evaluated according to that scheme selected. Then, in the prediction stage, the best learning scheme is used to build a predictor with all historical data and the predictor is finally used to predict defect on the new data.

Ruchika Malhotra in this paper they analyses and compares the statistical and six machine learning methods for fault prediction. These methods (Decision Tree, Artificial Neural Network, Cascade Correlation Network, Support Vector Machine, Group Method of Data Handling Method, and Gene Expression Programming) are empirically validated to find the relationship between the static code metrics and the fault proneness of a module. In this they compare the models predicted using the regression and the machine learning methods used two publicly available data sets AR1 and AR6.

Ahmet Okutan in this paper use Bayesian networks to determine the probabilistic influential relationships among software metrics and defect proneness. The metrics used in Promise data repository, define two more metrics, i.e. Number of Developers (NOD) for the number of developers and Lack of Coding Quality (LOCQ) for the source code quality.

Jaspreet Kaur et al in the paper, k-means based clustering approach has been used for finding the fault proneness of the Object oriented systems and found that k-means based clustering techniques shows 62.4% accuracy. It also showed high and low value of probability of detection.

Martin Shepperd et al. studied on the publicly available NASA datasets have been extensively used as part of this research to classify software modules into defect prone and not defect-prone categories. In this regard, the Promise Data Repository 2 has served an important role in making software engineering data sets publicly available. For example, there are 96 software defect datasets available. Amongst these are 13 out of the 14 data sets that have been provided by NASA and which were also available for download from the NASA Metrics Data Program (MDP) website.

Bowes et al. compare the performance of four ML classifiers to investigate individual predicted defects and analyse the level of prediction uncertainty. The publication contains a detailed description of the background and a review of the literature discussing the impact of the characteristics of datasets on predictive performance. Furthermore, it considers how the vast majority of research is based on NASA and PROMISE repositories, and how validation on in vivo datasets is essential.

Robert Andrew Weaver. The safety of software: Constructing and assuring arguments. University of York, Department of Computer Science, 2003. The Goal Structuring Notation is a well established technique use in safety critical industries for identifying requirements for evidence, documenting how evidence can be combined and assessing large sets of evidence.

Shuo Wang and Xin Yao. Using class imbalance learning for software defect prediction. IEEE Transactions on Reliability, 62(2):434–443, 2013. Dynamic version of AdaBoost.NC was performed that shows better PD and overall performance than the original AdaBoost.

Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, “A general software defect-proneness prediction framework. There is a bigger difference between the evaluation performance and the actual prediction performance in MGF’s study than with our framework.

#### 1.4 PROBLEM STATEMENT

The goal of software defect prediction using machine learning is to create a reliable and accurate model that can predict the likelihood of defects in software modules based on a variety of features, such as the modules' size, complexity, level of code quality, and historical defect data. Early identification of potential faults will allow for their correction before they have a chance to cause major issues or delays. This is crucial because, in some industries, like healthcare or transportation, software flaws can be dangerous to people's lives as well as expensive in terms of time, money, and reputation.

The software defect prediction problem can be framed as a supervised learning task, where the objective is to learn a mapping between the input features of the software modules and their corresponding defect status. The input features can include various aspects of the software module, such as its size, complexity, code quality, and past performance, while the output variable is a binary or continuous variable indicating the defect status or the number of defects.

The main challenge in software defect prediction is to identify the most relevant features that are predictive of the defect status, and to select an appropriate machine learning algorithm that can capture the complex relationships between the features and the output variable. Additionally, the performance of the model needs to be evaluated using appropriate metrics, such as accuracy, precision, recall, and F1-score, to ensure that the model is reliable and accurate.

## CHAPTER 2

### Technologies used

#### 1. Python:

Python is a popular programming language for machine learning and data analysis, and it offers several libraries and frameworks, such as scikit-learn, TensorFlow, and Keras, that can be used for software defect prediction. Machine learning and AI, as a unit, are still developing but are rapidly growing in usage due to the need for automation. Artificial Intelligence makes it possible to create innovative solutions to common problems, such as fraud detection, personal assistants, spam filters, search engines, and recommendations systems.

The demand for smart solutions to real-world problems necessitates the need to develop AI further in order to automate tasks that are tedious to program without AI. Python programming language is considered the best algorithm to help automate such tasks, and it offers greater simplicity and consistency than other programming languages. Further, the presence of an engaging python community makes it easy for developers to discuss projects and contribute ideas on how to enhance their code.

#### 2. R Language:

R is a popular programming language and open-source software environment for statistical computing and graphics. It is widely used in machine learning for various tasks such as data preprocessing, feature selection, model building, and evaluation. Here are some of the ways in which R is used in machine learning:

**Data Exploration and Preprocessing:** R provides several packages and functions for data exploration and preprocessing, such as dplyr, tidyr, ggplot2, and reshape2. These packages allow users to clean, transform, and visualize data before applying machine learning algorithms.

**Feature Selection:** R provides several packages for feature selection, such as caret, Boruta, and FSelector. These packages help to identify the most important features in the dataset, which can improve the accuracy of machine learning models and reduce overfitting.

**Model Building:** R provides several packages and algorithms for model building, such as glmnet, randomForest, xgboost, and caret. These algorithms can be used for classification, regression, and clustering tasks.

**Model Evaluation:** R provides several packages for model evaluation, such as caret, pROC, and MLmetrics. These packages help to evaluate the performance of machine learning models using metrics such as accuracy, precision, recall, F1-score, ROC curve, and AUC.

**Deep Learning:** R provides several packages and frameworks for deep learning, such as Keras, TensorFlow, and MXNet. These packages allow users to build deep neural networks for image recognition, natural language processing, and other complex tasks.



Overall, R is a powerful tool for machine learning and data analysis, and it is widely used in research and industry for solving real-world problems.

### **3. WEKA TOOL**

Weka is a popular open-source data mining tool that provides a graphical user interface and several machine learning algorithms for classification, regression, clustering, and association rule mining. Here are some of the ways in which Weka is used in machine learning:

**Data Preprocessing:** Weka provides several tools for data preprocessing, such as filtering, normalization, and attribute selection. These tools allow users to clean, transform, and preprocess data before applying machine learning algorithms.

**Feature Selection:** Weka provides several algorithms for feature selection, such as Correlation-based Feature Selection (CFS), Principal Component Analysis (PCA), and ReliefF. These algorithms help to identify the most important features in the dataset, which can improve the accuracy of machine learning models and reduce overfitting.

**Model Building:** Weka provides several algorithms for model building, such as decision trees, neural networks, support vector machines, and k-nearest neighbors. These algorithms can be used for classification, regression, and clustering tasks.

**Model Evaluation:** Weka provides several tools for model evaluation, such as cross-validation, confusion matrices, and ROC curves. These tools help to evaluate the performance of machine learning models using metrics such as accuracy, precision, recall, F1-score, ROC curve, and AUC.

**Association Rule Mining:** Weka provides several algorithms for association rule mining, such as Apriori and FP-growth. These algorithms help to discover patterns and relationships between items in large datasets.

Overall, Weka is a powerful tool for machine learning and data mining, and it is widely used in research and industry for solving real-world problems. Its user-friendly interface and extensive documentation make it easy for users to experiment with different algorithms and techniques, even if they have little or no programming experience.

### **4. RapidMiner:**

The idea behind Rapid Mining tool is to create one place for everything. Starting from

providing multiple datasets to model deployment through the platform you can do it all here. Some of the facilities of this platform are:

Rapid Miner provides its own collection of datasets but it also provides options to set up a database in the cloud for storing large amounts of data. You can store and load the data from Hadoop, Cloud, RDBMS, NoSQL etc. Apart from this, you can load your CSV data very easily and start using it as well.

The standard implementation of procedures like data cleaning, visualization, pre-processing can be done with drag and drop options without having to write even a single line of code.

Rapid Miner provides a wide range of machine learning algorithms in classification, clustering and regression as well. You can also train optimal deep learning algorithms like Gradient Boost, XGBoost etc. Not only this, but the tool also provides the ability to perform pruning and tuning.

Finally, to bind everything together, you can easily deploy your machine learning models to the web or to mobiles through this platform. You just need to create user interfaces to collect real-time data and run it on the trained model to serve a task.

Because of all of the above-mentioned facilities, users find this tool very useful and easy to use when compared to platforms like Tensorflow or Keras.

## **5. KNIME:**

KNIME (Konstanz Information Miner) Studio is an open-source data analytics and machine learning tool that provides a visual workflow editor and a wide range of built-in machine learning algorithms. Here are some of the ways in which KNIME Studio is used in machine learning:

**Data Integration and Preprocessing:** KNIME Studio provides several tools for data integration and preprocessing, such as data cleaning, data transformation, and data normalization. These tools allow users to prepare the data before applying machine learning algorithms.

**Feature Selection:** KNIME Studio provides several algorithms for feature selection, such as Information Gain, Chi-squared, and Correlation-based Feature Selection. These algorithms help to identify the most important features in the dataset, which can improve the accuracy of machine learning models and reduce overfitting.

**Model Building:** KNIME Studio provides several algorithms for model building, such as decision trees, neural networks, support vector machines, and k-nearest neighbors. These algorithms can be used for classification, regression, and clustering tasks.

**Model Evaluation:** KNIME Studio provides several tools for model evaluation, such as cross-validation, confusion matrices, and ROC curves. These tools help to evaluate the performance of machine learning models using metrics such as accuracy, precision,

recall, F1-score, ROC curve, and AUC.

**Ensemble Modeling:** KNIME Studio provides several algorithms for ensemble modeling, such as bagging, boosting, and random forests. These algorithms help to improve the accuracy and stability of machine learning models by combining multiple models.

**Text Analytics:** KNIME Studio provides several algorithms for text analytics, such as sentiment analysis, topic modeling, and text classification. These algorithms help to analyze and understand large volumes of text data.

**Deep Learning:** KNIME Studio provides support for deep learning frameworks, such as TensorFlow and Keras. These frameworks allow users to build and train deep neural networks for complex tasks such as image recognition and natural language processing.

Overall, KNIME Studio is a powerful tool for machine learning and data analysis, and it is widely used in research and industry for solving real-world problems. Its user-friendly interface and extensive documentation make it easy for users to experiment with different algorithms and techniques, even if they have little or no programming experience.

## **6. H2O.ai:**

H2O.ai is a popular open-source software for data analytics and machine learning. It provides a set of powerful algorithms for predictive modeling, classification, and clustering, and it can be used with different programming languages such as R, Python, and Java. Here are some of the ways in which H2O.ai is used in machine learning:

**Data Preprocessing:** H2O.ai provides several tools for data preprocessing, such as filtering, normalization, and feature engineering. These tools allow users to clean, transform, and preprocess data before applying machine learning algorithms.

**Model Building:** H2O.ai provides several algorithms for model building, such as gradient boosting, random forests, generalized linear models, deep learning, and k-means clustering. These algorithms can be used for classification, regression, and clustering tasks.

**Model Selection and Tuning:** H2O.ai provides tools for model selection and tuning, such as grid search and random search. These tools help to identify the best set of hyperparameters for a given model and improve its performance.

**Model Interpretability:** H2O.ai provides tools for model interpretability, such as partial

dependence plots, feature importance, and SHAP values. These tools help to understand how the model works and which features are most important for making predictions.

**Automatic Machine Learning:** H2O.ai provides an automatic machine learning (AutoML) feature that automates the process of selecting the best model and hyperparameters for a given dataset. This feature can save time and effort in the model building process.

**Distributed Computing:** H2O.ai provides distributed computing capabilities, which allow users to scale their machine learning models to large datasets and parallelize their computations across multiple machines.

Overall, H2O.ai is a powerful tool for machine learning and data analysis, and it is widely used in research and industry for solving real-world problems. Its user-friendly interface and extensive documentation make it easy for users to experiment with different algorithms and techniques, even if they have little or no programming experience.

## **DEVELOPMENT METHODOLOGIES**

When it comes to developing software defect prediction models using machine learning algorithms, several software development methodologies can be used. Some of the most common methodologies:

1. **Waterfall Model:** The waterfall model is a traditional software development methodology that follows a linear and sequential approach to software development. In this approach, software development is divided into several phases such as requirements gathering, design, implementation, testing, and maintenance. In software defect prediction, the waterfall model can be used to develop a machine learning model using historical data from past projects.
2. **Agile Methodology:** The Agile methodology is an iterative and incremental software development approach that emphasizes collaboration, flexibility, and customer satisfaction. In software defect prediction, the Agile methodology can be used to develop a machine learning model that is continuously updated and refined based on feedback from end-users and stakeholders.
3. **Six Sigma:** Six Sigma is a data-driven quality management methodology that aims to improve the quality of processes by identifying and eliminating defects. In software defect prediction, the Six Sigma methodology can be used to develop a machine learning model that is designed to identify and eliminate defects in the software development process.

## Chapter 3

### PROPOSED SYSTEM

Defect classification is the basis for the quantitative analysis of defects. Due to the numerous components that cause defects, every individual distinguishes the defect type differently, for example, database type, code type, operation type, etc. Each type can keep on being classified; for example, code types can be partitioned into task errors, variable definition errors, etc. There is a wide range of classification methods. Different classification methods should be utilized for various analysis purposes, so the classification methods are also quite complicated.

#### A. Pre-processing:

There are several datasets available for software defect detection. This research used seven datasets, namely, KC1, JM1, CM1, KC2, PC1, AT, KC1 CL. All these datasets are obtained from the NASA promise dataset repository. I performed an in-depth analysis of every dataset to understand every feature's effect on the output prediction. It was found that many features had a high correlation with others, and this led to a little overfitting. Hence, these features were not considered at the time of final training. Principal Component Analysis (PCA) is a method to reduce large datasets' dimensionality while minimizing information loss. PCA creates new uncorrelated variables such that variance is maximized. Applying this technique to the KC1 CL and JM1 results in significant improvement in performance.

**Data Gathering:** The initial phase in the software development process is to collect data on bugs, code complexity, and other pertinent software metrics. The information may be gathered from bug tracking programmes, software repositories, or other sources.

**Data Preparation:** In order to make the acquired data usable for machine learning algorithms, it must be cleaned, converted, and preprocessed. Outliers must be eliminated, missing values must be filled in, and the data must be scaled and normalised. The following step is to choose the traits that are most useful in predicting faults. The most crucial features can be found using feature selection techniques including correlation analysis, principal component analysis, and decision trees.

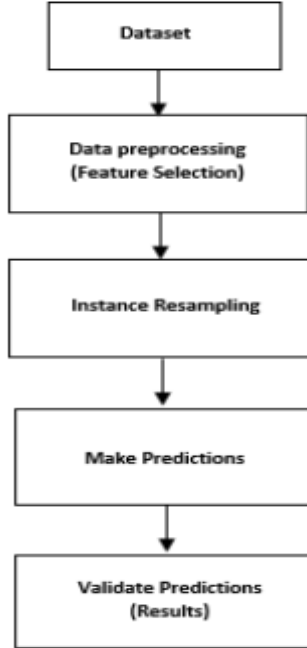
**Algorithm Selection and Training:** Following the selection of the features, the following step is to select the best machine learning algorithm for the job. For predicting defects, one can use algorithms like artificial neural networks, decision trees, and support vector machines. The preprocessed data is then used to train the selected algorithm.

**Model Evaluation:** To gauge the performance of the trained model, metrics like

precision, recall, accuracy, and F1-score are used. In order to make sure the model will adapt adequately to new data, it is additionally evaluated using a validation dataset.

Deploying the trained model in the software development process is the last stage. To forecast problems in real-time, this entails connecting the model with software development tools such as bug tracking systems and code review tools.

**Model Upkeep and Enhance:** The model needs to be updated and enhanced over time in order to perform properly. This entails gathering fresh data, updating the model, and routinely assessing how well it is performing.



*Figure 2 Main steps of the Proposed Approach*

#### B. Model Classification:

1. **Logistic Regression:** For binary classification problems logistic regression is a statistical technique. Logistic regression can be used in software defect prediction to determine from a software module's features whether it is defective or not.
2. **Random Forest:** An ensemble learning method called random forest employs many decision trees to create predictions. To produce a more accurate and reliable model, it combines the predictions of various decision trees.
3. **Naïve Bayes:** The probabilistic algorithm Naive Bayes estimates the likelihood that a given instance belongs to a particular class. It works under the assumption that each attribute is independent of the others.
4. **Support Vector Machines (SVM):** SVM is an efficient algorithm that can be applied to the prediction of software defects. It functions by identifying the hyper plane that maximally divides instances of various classes.

5. **Neural Networks:** A group of methodologies known as neural networks are used to find patterns in data. By training them on previous data and applying their predictions to new cases, they can be used to forecast software bugs.
6. **Gradient Boosting Classifier:** In machine learning contests, the Gradient Boosting Classifier is an especially popular ensemble algorithm. It involves progressively adding models, with each model correcting the performance of the one before it.
7. **CSSVM:** Cost-sensitive support vector machine (CSSVM), given that different category samples had varying misclassification costs as traditional SVM shows poor results on imbalance datasets.
8. **Genetic Algorithm:** A common optimisation method used for predicting defects in software is the genetic algorithm. It works by emulating natural selection, in which the most fit individuals are selected and bred to create the following generation. It can be used to improve the performance of a machine learning model, such as SVM or Random Forest, in the context of software defect prediction.
9. **Random Forest with Metaheuristic Algorithm:** The best set of hyperparameters for the Random Forest algorithm can be found using optimisation algorithms called metaheuristics.
10. **Linear Regression:** Linear regression is a commonly used statistical technique that can be applied to software defect prediction. The basic idea behind linear regression is to find the best line or plane that fits the relationship between the predictor variables and the response variable (faults).

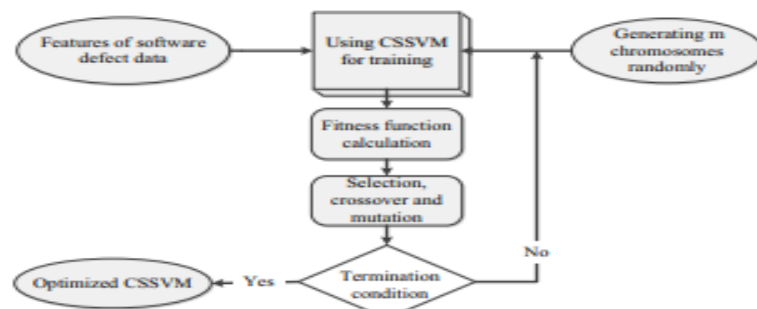


Figure 3 Framework of the GA-CSSVM method

### 3.1. PROPOSED APPROACH

This is the first step, that chose the most relevant features and eliminate the redundant ones. It comprises using a subset evaluation with a search strategy, this is presented as follows:

1. The Correlation-based Feature Selection (CFS) evaluates the subsets of features that originated from the search strategy phase and chooses the subsets that have the lowest correlation among themselves while also being highly correlated to the class.
2. Search Strategy: Bat-Search metaheuristic algorithm was used for the search purpose to reduce the high dimensionality of the search space. Using the BA and the CFS, the best feature subset will be the output from this step.
3. Resampling: Supervised instance resampling was used, it produces a random subsample of a dataset using sampling with replacement. To overcome the class imbalance problem. Therefore, the data will be balanced after this step.
4. Class prediction: This is the last step, where the class of the instances with the chosen metrics will be predicted. The Random Forest Algorithm (RF) will be used as a classifier, to classify instances into defective or non-defective classes. RF achieves high accuracy due to the nature of its' work, where it injects the randomness into its' procedure, and uses the ensemble of trees (forest of trees) that votes for the class of each instance.

### 3.2. SYSTEM DESIGN AND ARCHITECTURE

A cleaned and updated version on 2016 of four public Software Defect Prediction datasets, that are available on the Software Engineering research data repository, tera PROMISE repository, have been used; PC1, PC2, PC3 and PC4. These datasets have been collected from C functions for a flight software for earth orbiting satellite. Data was collected through extracting McCabe and Halstead features of source code, this was done on the 70s, where the characteristics of code that are associated to the software quality are placed. The datasets have three types of software metrics (Code, Design and other metrics).



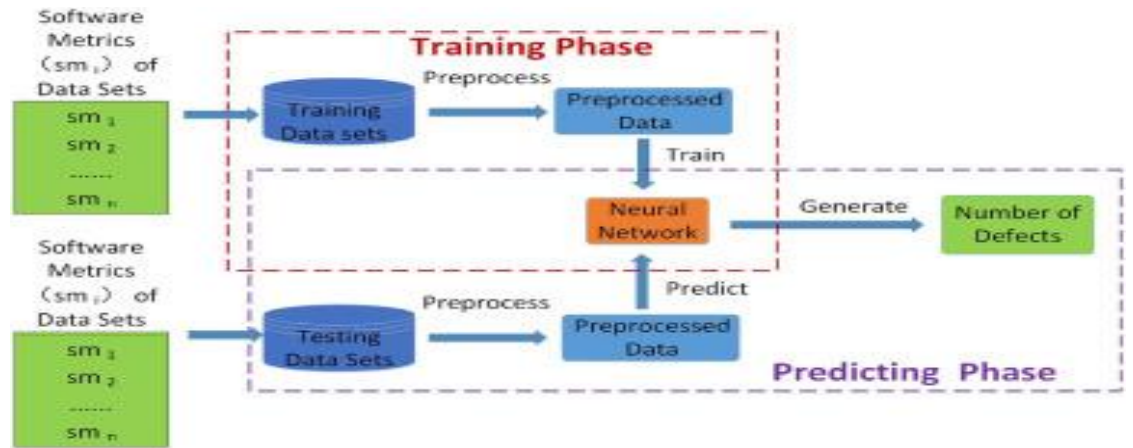


Figure 4 Proposed System Architecture

**Data Collection and Acquisition:** Data acquisition is an essential module for any project, especially in defect prediction. In this module, we collected a large set of dataset from tera PROMISE repository. These datasets have been collected from C functions for a flight software for earth orbiting satellite. Data was collected through extracting McCabe and Halstead features of source code, this was done on the 70s, where the characteristics of code that are associated to the software quality are placed.

**Importing Libraries and Dataset:** To work on any project, importing libraries and dataset is the first step. We imported necessary libraries such as Tensorflow, Keras, Numpy, and Pandas in Python for our project. The dataset was loaded into the system using the Pandas library, which helped us manage the data efficiently. We used the Pandas library to convert the dataset into a DataFrame and manipulate it for further analysis.

**Data Exploration and Pre-processing:** Before training our deep learning models, we needed to perform data exploration and pre-processing on the dataset. In this module, we removed any redundant or irrelevant data from the dataset, checked for defective data.

**Data Visualization:** Data visualization is a crucial step in any data analysis project as it helps us gain insights into the data and identify patterns. In this module, we used visualization techniques to analyze and explore the dataset. These visualizations helped us understand the distribution of data, gain insights into the data that we could use to train our models and also study and infer important information from results.

**Data Preparation for Training:** In this module, we prepared the dataset for training our machine learning models. We split the data into training and validation sets and performed data balancing to ensure that the classes were evenly distributed. We also applied various techniques such as one-hot encoding and label encoding to convert the categorical data into numerical data that the models could understand.

**Model Evaluation:**

WEKA tool that is dedicated to machine learning algorithms will be employed in the evaluation process, many feature selection algorithms have been conducted to choose the highly efficient attributes, in order to handle the high dimensionality problem. Therefore, it will be able to determine which algorithm chooses the best features. Then a resampling of instances will be done to handle the class imbalance problem. As a last step, many classifiers will be applied to classify instances into defective and not-defective classes. Therefore, the best classifier among them in terms of accuracy can be detected. Two comparisons have been conducted for the evaluation of the proposed approach; where the first comparison was done to explain why the chosen algorithms (RF, BA) were selected for the proposed SDF approach, in addition to a comparison between the proposed approach and other approaches in terms of the accuracy for the evaluation purpose.

### 3.3. ALGORITHMS USED:

#### 1. LINEAR REGRESSION:

Linear regression is a commonly used technique for software defect prediction. It involves using a mathematical model to predict the number of defects that are likely to occur in a piece of software based on certain input features.

To use linear regression for software defect prediction, the first step is to identify the input features that are most likely to be correlated with the occurrence of defects. These features could include factors such as the size of the code base, the complexity of the code, the experience level of the development team, and the amount of testing that has been done.

Once the input features have been identified, the next step is to collect data on the number of defects that have occurred in previous software projects and the corresponding values of the input features. This data can then be used to train a linear regression model, which will be able to predict the number of defects that are likely to occur in future software projects based on the values of the input features.

There are several factors to consider when using linear regression for software defect prediction, such as the quality and quantity of the training data, the choice of input features, and the accuracy of the model's predictions. It is also important to continually evaluate and refine the model to ensure that it remains accurate and effective over time.

## **2. SVM:**

Support Vector Machines (SVMs) can also be used for software fault prediction. SVMs are a type of supervised machine learning algorithm that can be used for classification tasks, such as predicting whether a piece of software is likely to contain defects or not.

To use SVMs for software defect prediction, the first step is to identify the input features that are most likely to be correlated with the occurrence of defects. These features could include factors such as the size of the code base, the complexity of the code, the experience level of the development team, and the amount of testing that has been done.

Once the input features have been identified, the next step is to collect data on the number of defects that have occurred in previous software projects and the corresponding values of the input features. This data can then be used to train an SVM model, which will be able to predict whether a new piece of software is likely to contain defects or not based on the values of the input features.

One advantage of using SVMs for software defect prediction is that they can work well with high-dimensional data and can handle non-linear relationships between the input features and the occurrence of defects. However, SVMs can be computationally expensive and require careful tuning of hyperparameters to achieve the best results.

## **3. LOGISTIC REGRESSION:**

Logistic regression is another commonly used technique for software defect prediction. Unlike linear regression, which is used to predict continuous values, logistic regression is used to predict binary outcomes, such as whether a piece of software contains defects or not.

To use logistic regression for software defect prediction, the first step is to identify the input features that are most likely to be correlated with the occurrence of defects. These features could include factors such as the size of the code base, the complexity of the code, the experience level of the development team, and the amount of testing that has been done.

Once the input features have been identified, the next step is to collect data on the number of defects that have occurred in previous software projects and the corresponding values of the input features. This data can then be used to train a logistic regression model, which will be able to predict whether a new piece of software is likely to contain defects or not based on the values of the input

features.

One advantage of using logistic regression for software defect prediction is that it is a relatively simple and interpretable model, which can make it easier to understand the factors that are driving the occurrence of defects. However, logistic regression assumes that the relationship between the input features and the occurrence of defects is linear, which may not always be the case in practice.

#### **4. RANDOM FOREST**

Random Forest is another popular machine learning technique that can be used for software defect prediction. It is a type of ensemble learning method that combines multiple decision trees to make predictions.

To use Random Forest for software defect prediction, the first step is to identify the input features that are most likely to be correlated with the occurrence of defects. These features could include factors such as the size of the code base, the complexity of the code, the experience level of the development team, and the amount of testing that has been done.

Once the input features have been identified, the next step is to collect data on the number of defects that have occurred in previous software projects and the corresponding values of the input features. This data can then be used to train a Random Forest model, which will be able to predict whether a new piece of software is likely to contain defects or not based on the values of the input features.

One advantage of using Random Forest for software defect prediction is that it can handle high-dimensional data with non-linear relationships between input features and the occurrence of defects. It can also handle missing data and outliers. Moreover, it is less prone to overfitting as compared to individual decision trees.

#### **5. NAÏVE BAYES**

Naive Bayes is another machine learning technique that can be used for software defect prediction. It is a probabilistic classifier based on Bayes' theorem, which calculates the probability of an event based on prior knowledge of conditions that might be related to the event.

To use Naive Bayes for software defect prediction, the first step is to identify

the input features that are most likely to be correlated with the occurrence of defects. These features could include factors such as the size of the code base, the complexity of the code, the experience level of the development team, and the amount of testing that has been done.

Once the input features have been identified, the next step is to collect data on the number of defects that have occurred in previous software projects and the corresponding values of the input features. This data can then be used to train a Naive Bayes model, which will be able to predict whether a new piece of software is likely to contain defects or not based on the values of the input features.

One advantage of using Naive Bayes for software defect prediction is that it can handle high-dimensional data and can be trained quickly. It is also relatively simple and interpretable, making it easier to understand the factors that are driving the occurrence of defects.

## **6. CSSVM**

Cost-sensitive SVM (Support Vector Machine) is a modified version of SVM that can be used for software defect prediction with an emphasis on cost-effectiveness. It is particularly useful when the cost of misclassification of the defect-prone and non-defect-prone software is not equal.

Cost-sensitive SVM introduces the notion of cost into the standard SVM framework by assigning different misclassification costs to different types of errors. For example, the cost of misclassifying a non-defective software as defective may be lower than the cost of misclassifying a defective software as non-defective, and the cost-sensitive SVM accounts for this by modifying the weights in the optimization function.

One advantage of using cost-sensitive SVM for software defect prediction is that it can account for the cost of misclassification and provide more accurate results when the cost of errors is not equal. However, it is important to carefully choose the misclassification costs, which may require domain knowledge or a cost-benefit analysis.

## **7. NEURAL NETWORK**

Neural Networks are a type of machine learning model that are inspired by the structure and function of the human brain. They can also be used for software defect prediction.

\One advantage of using neural networks for software defect prediction is their

ability to capture complex non-linear relationships between input features and the occurrence of defects. They can also handle missing data and outliers and are relatively robust to noisy data.

However, neural networks require a large amount of data to train effectively, and they can be computationally expensive to train and evaluate. It is also important to avoid overfitting, which can occur when the model is too complex and fits the training data too closely, resulting in poor generalization to new data.

It is important to continually evaluate and refine the neural network model to ensure that it remains accurate and effective over time, and to optimize the hyperparameters such as the number of layers, the number of neurons per layer, the activation functions, and the learning rate, etc. Techniques such as regularization can also be used to prevent overfitting.

### 3.4. ABOUT DATASETS

There are many datasets for predicting software defects in the well-known PROMISE Repository, which contains datasets for software engineering. The NASA measurements Data Program is one such collection that contains measurements from various software development projects.

```
@attribute loc numeric
@attribute v(g) numeric
@attribute ev(g) numeric
@attribute iv(g) numeric
@attribute n numeric
@attribute v numeric
@attribute l numeric
@attribute d numeric
@attribute i numeric
@attribute e numeric
@attribute b numeric
@attribute t numeric
@attribute lOCode numeric
@attribute lOComment numeric
@attribute lOBlank numeric
@attribute locCodeAndComment numeric
@attribute uniq_Op numeric
@attribute uniq_Opnd numeric
@attribute total_Op numeric
@attribute total_Opnd numeric
@attribute branchCount numeric
@attribute defects {false,true}
```

*Figure 5 Structure of datasets*

### 3.5. SYSTEM MODULES USED

**Module 1:** Initialization: In this module, an initial sample of potential solutions is generated.

**Module 2:** Fitness evaluation: Based on a number of factors, including prediction accuracy, interpretability, and computing cost, this module assesses the fitness of each feature subset.

**Module 3:** Selection: For mating and producing the following population, this module selects the best feature subsets from the current population.

**Module 4:** Crossover: With the help of this module, new offspring are created by combining the chosen feature subsets.

**Module 5:** Mutation: This module makes small changes to the offspring to increase the diversity of the population.

**Module 6:** Non-dominated sorting: This module sorts the population based on their non-domination levels, which are used to determine the Pareto optimal front.

**Module 7:** Elitism: To prevent the loss of progress, this module saves the best solutions so far.

**Module 8:** Termination: When a termination criterion has been met, such as reaching a certain generational threshold or identifying a suitable set of solutions, this module ends the process.

The set of Pareto optimal solutions that represent the trade-off between various objectives for software defect prediction is the end result of these modules being executed in a loop until the termination conditions is satisfied.

### 3.6. Sequence diagram of the different components of the Proposed System.

Steps Involved:

1. Identify the system components
2. Define the interactions:
3. Add timing and synchronization
4. Analyze the interactions
5. Refine and iterate

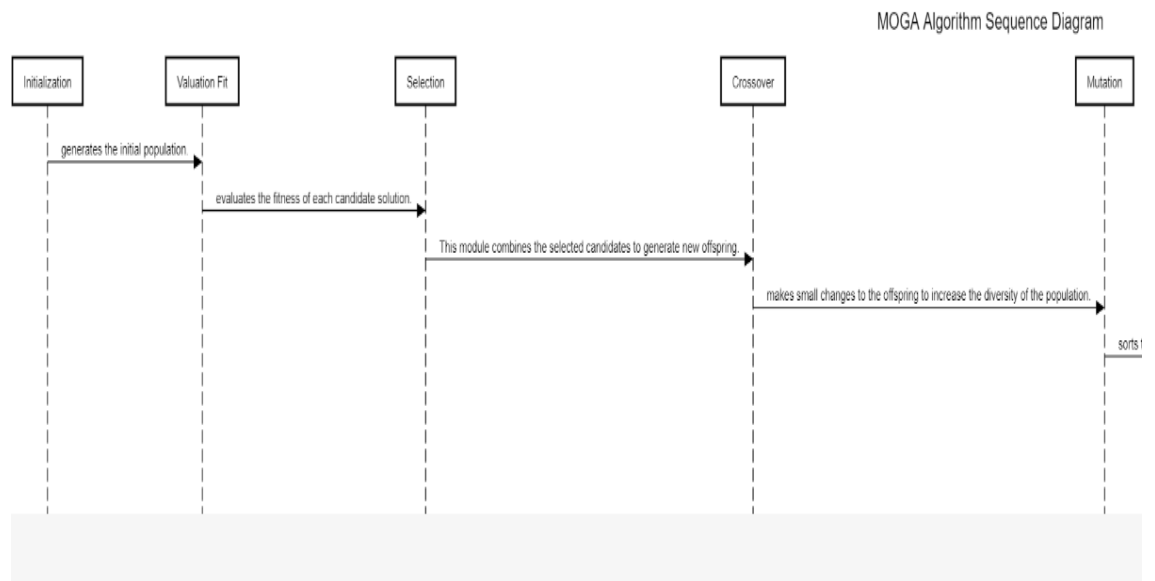


Figure 6 Sequence Diagram for different component of Genetic algorithm.

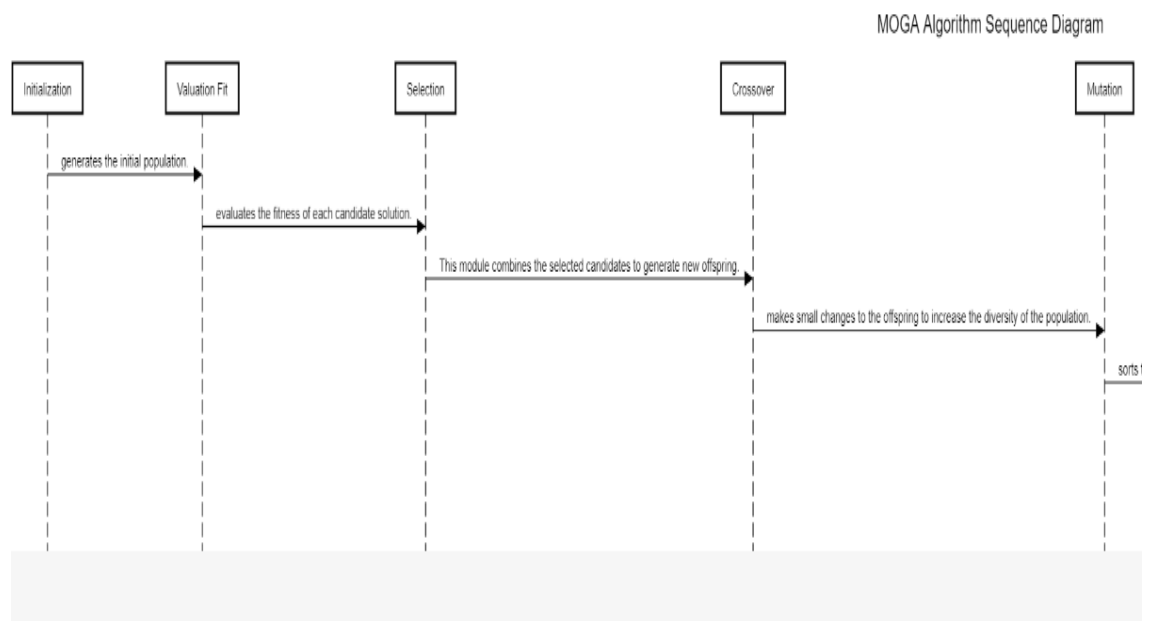


Figure7. Sequence Diagram for different component of Genetic algorithm.



## Chapter 4

### IMPLEMENTATION DETAILS

The proposed system for doing this project is using a CSSVM (Cost Effective SVM) using Genetic algorithm (GA). We will be building a classifier using CSSVM which would be able to classify data into defective and non-defective class. This project was done on Jupyter Notebook (Python) and the dataset was obtained from tera PROMISE repository. Several libraries were used in the development of this project. For example, important libraries such as numpy, pandas, sklearn, matplotlib would help in handling data and perform several tasks. Some necessary data preprocessing was also done on the data such as removing or correcting missing or corrupted data, dealing with outliers, and removing duplicates.

#### 4.1. MAJOR LIBRARIES AND FUNCTIONS USED

The implementation of our machine learning models for detecting defect in software involved the use of several Python libraries and frameworks. In this section, we will provide an overview of the libraries and frameworks that were used in our implementation and their respective roles in the pipeline.

**numpy:** It's a library used for working with arrays and matrices in Python. It provides a convenient interface for performing numerical operations on multi-dimensional arrays.

**pandas:** It's a library used for data manipulation and analysis. It provides data structures for efficiently storing and manipulating large datasets.

**matplotlib:** It's a data visualization library used for creating static, interactive, and animated visualizations in Python.

**tensorflow:** It's a popular open-source machine learning framework used for building and training deep learning models.

**sklearn:** It's a library used for machine learning tasks such as classification, regression, and clustering. It provides a wide range of algorithms and tools for data preprocessing, model selection, and evaluation.

**Plotly:** It is an open-source data visualization library that allows you to create interactive and publication-quality graphs and charts.

**Chart-Studio:** It is a cloud-based platform provided by Plotly that allows users to create and share interactive visualizations created with Plotly. The Chart Studio library is a Python library that provides an interface for creating and uploading visualizations to Chart Studio.

**Keras:** Keras is a high-level neural networks API written in Python that can be run on top of TensorFlow. It provides a user-friendly interface for building and training deep learning models, which can be used for software defect prediction.

**os:** It's a library used for interacting with the operating system. It provides a way to perform tasks like navigating directories, creating and deleting files, etc.

**Flatten:** It's a class in the Keras library used for flattening the output of a convolutional layer into a 1D array.

**Dense:** It's a class in the Keras library used for creating a fully connected layer in a deep learning model.

**MaxPool1D:** It's a function from the Keras library that performs 1D pooling in a neural network. 1D pooling is a common operation used in convolutional neural networks (CNNs) for feature extraction.

**Conv1D:** It's a function from the Keras library that performs 1D convolution in a neural network. 1D convolution is a common operation used in convolutional neural networks (CNNs) for feature extraction.

## 4.2. DATASET SPLIT

Data splitting is a crucial step in machine learning and deep learning, where we divide our available data into different subsets, such as training, validation, and testing sets, to ensure that our model can generalize well to new, unseen data. The training set is the largest subset, consisting most of the available data, which is used to train the model's parameters. The model learns the underlying patterns in the training data and adjusts its weights and biases accordingly to minimize the loss function.

In my project I used a 70-15-15 data split, where 70% of the data was used for training, and 15% each for validation and testing. The reason behind this split was to ensure that the model is able to learn from a sufficiently large amount of data during training, and also to prevent overfitting of the model. During the training phase, the model was able to learn from the training data, and the validation data was used to evaluate the performance of the model and tune its hyperparameters, such as the learning rate and batch size. This was done to prevent the model from overfitting to the training data and to improve its generalization ability.

Once the model was trained and the hyper parameters were tuned, it was evaluated on the test data to determine its performance on unseen data. The use of a separate test set ensures that the model's performance is not biased towards the validation set or the training set. Overall, the 70-15-15 data split allowed us to train and evaluate our model effectively, and ensured that it had good performance on both the training and testing data.

## Appendices

### SOURCE CODE

```
// importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
pip install plotly
pip install chart-studio

// loading datasets
data= pd.read_csv('C:/Users/User/Pictures/Software-Defect
Prediction/Data/data/jm1.csv')

// data info
defects_true_false = data.groupby('defects')['b'].apply(lambda x: x.count())
#defect rates (true/false)
print('False : ', defects_true_false[0])
print('True : ', defects_true_false[1])

// Visualisation Part
trace = go.Histogram(
    x = data.defects,
    opacity = 0.75,
    name = "Defects",
    marker = dict(color = 'green'))

hist_data = [trace]
hist_layout = go.Layout(barmode='overlay',
    title = 'Defects',
    xaxis = dict(title = 'True - False'),
    yaxis = dict(title = 'Frequency'),
)
fig = go.Figure(data = hist_data, layout = hist_layout) iplot(fig)
```

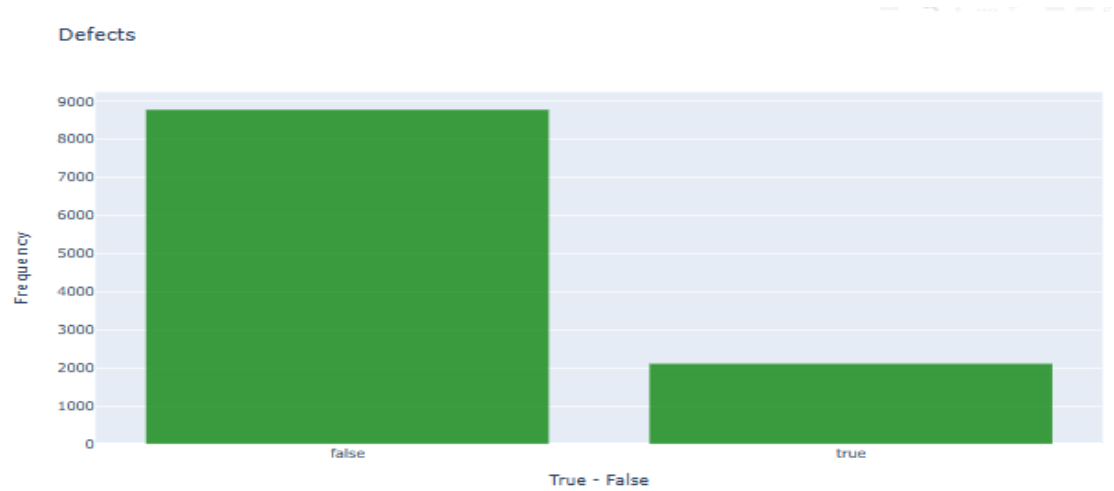


Figure 8 The histogram for instances

#Heatmap

f,ax = plt.subplots(figsize = (15, 15))

sns.heatmap(data.corr(), annot = True, linewidths = .5, fmt = '.2f')

plt.show()

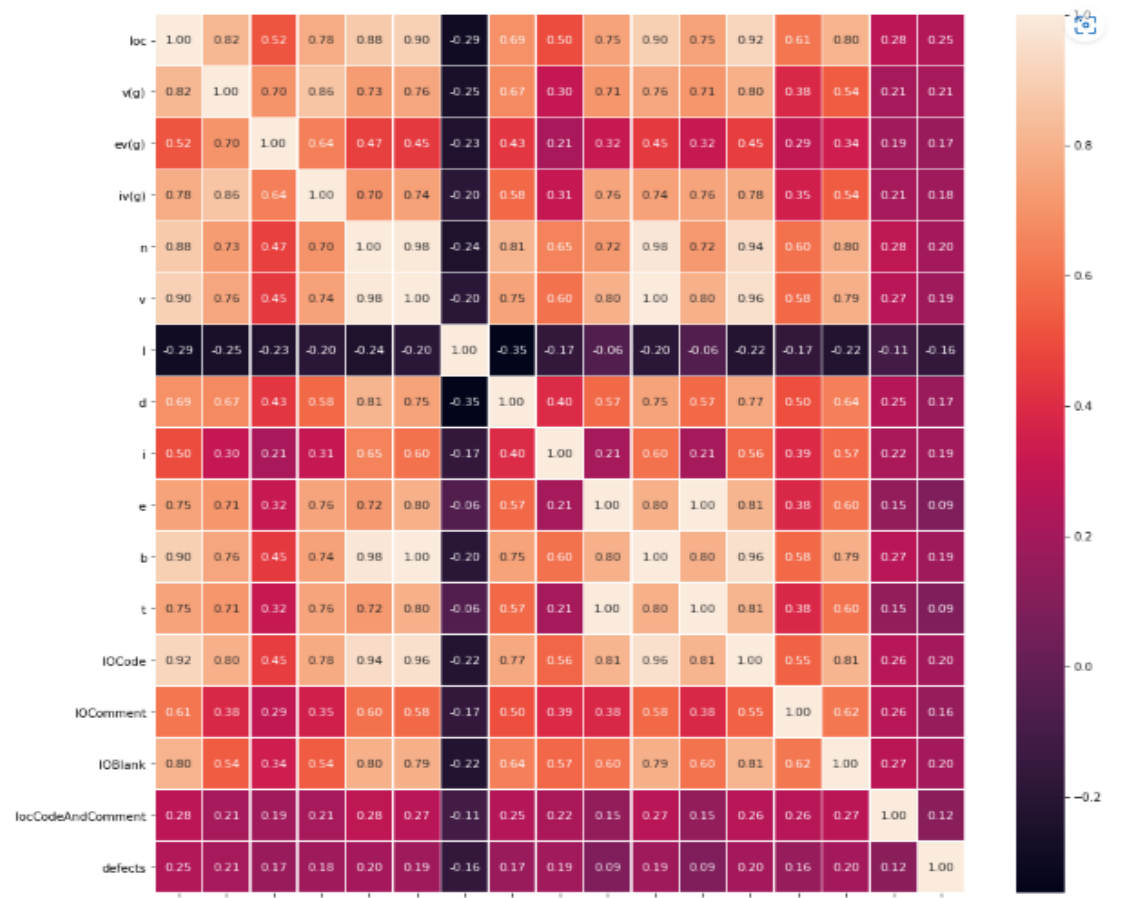


Figure 9 The Heatmap for instances

```

#Scatterplot
trace = go.Scatter(
    x = data.v,
    y = data.b,
    mode = "markers",
    name = "Volume - Bug",
    marker = dict(color = 'darkblue'),
    text = "Bug (b)")

scatter_data = [trace]
scatter_layout = dict(title = 'Volume - Bug',
    xaxis = dict(title = 'Volume', ticklen = 5),
    yaxis = dict(title = 'Bug' , ticklen = 5),
    )
fig = dict(data = scatter_data, layout = scatter_layout)
iplot(fig)

```

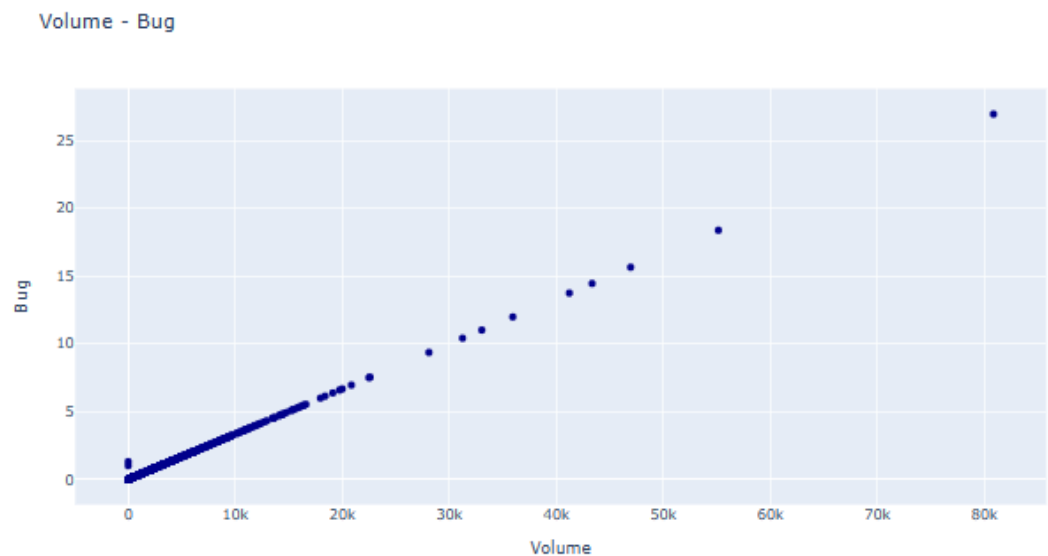


Figure 10 The Scatter Plot for instances.

```

// data preprocessing
data.isnull().sum()

// outlier detection
#Boxplot
trace1 = go.Box(

```

```

x = data.uniq_Op,
name = 'Unique Operators',
marker = dict(color = 'blue')
)
box_data = [trace1]
iplot(box_data)

```

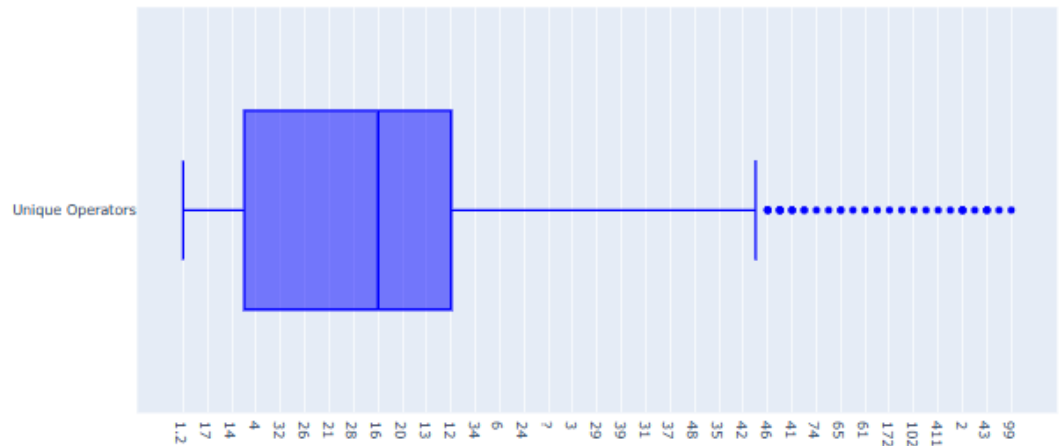


Figure 11 The BoxPlot for instances.

// Data Normalization (Min-Max Normalization)

```

from sklearn import preprocessing
scale_v = data[['v']]
scale_b = data[['b']]
minmax_scaler = preprocessing.MinMaxScaler()
v_scaled = minmax_scaler.fit_transform(scale_v)
b_scaled = minmax_scaler.fit_transform(scale_b)
data['v_ScaledUp'] = pd.DataFrame(v_scaled)
data['b_ScaledUp'] = pd.DataFrame(b_scaled)
data
scaled_data = pd.concat([data.v , data.b , data.v_ScaledUp , data.b_ScaledUp], axis=1)
scaled_data

```

	v	b	v_ScaledUp	b_ScaledUp
0	1.30	1.30	0.000016	0.048237
1	1.00	1.00	0.000012	0.037106
2	1134.13	0.38	0.014029	0.014100
3	4348.76	1.45	0.053793	0.053803
4	599.12	0.20	0.007411	0.007421
...	...	...	...	...
10880	241.48	0.08	0.002987	0.002968
10881	129.66	0.04	0.001604	0.001484
10882	519.57	0.17	0.006427	0.006308
10883	147.15	0.05	0.001820	0.001855
10884	272.63	0.09	0.003372	0.003340

10885 rows x 4 columns

Figure 72 Data Normalization

## MODEL SELECTION

1. **LINEAR REGRESSION:** In this code, we first load the dataset into a Pandas DataFrame, split the data into training and testing sets using `train_test_split`, and create a LinearRegression model. We then fit the model to the training data using the `fit` method, make predictions on the testing data using the `predict` method, and evaluate the model using mean squared error and R-squared score.

```
#Parsing selection and verification datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=0)

#Creation of Linear Regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

```
#The results of the model. (This uses the Least squares method and the Root mean square error methods)
from sklearn import metrics
print('Mean Squared Error (MSE):', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error (RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Squared Error (MSE): 0.06344413069339941
Root Mean Squared Error (RMSE): 0.2518811836827027
```

Figure 13 The RMS function output

2. **NAÏVE BAYES:**

The dataset should contain a set of features and a binary label indicating whether a software component is defective or not. The code uses the Pandas library to load and preprocess the dataset, and the Scikit-learn library to create and train a Gaussian Naive Bayes model. The model is then used to make predictions on a testing set, and the accuracy and classification report are printed to the console.

```
#Creation of Naive Bayes model
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()

#Calculation of ACC value by K-fold cross validation of NB model
scoring = 'accuracy'
kfold = model_selection.KFold(n_splits = 10, random_state = None)
cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv = kfold, scoring = scoring)

cv_results

array([0.98507463, 0.98277842, 0.9793341 , 0.97014925, 0.98163031,
       0.9793341 , 0.98392652, 0.9793341 , 0.98045977, 0.97586207])

msg = "Mean : %f - Std : (%f)" % (cv_results.mean(), cv_results.std())
msg

'Mean : 0.979788 - Std : (0.004084)'
```

Figure 14 Results for Naive Bayes

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,
random_state = 0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
#Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))

```

	precision	recall	f1-score	support
Redesign	0.93	0.94	0.94	319
Successful	0.99	0.99	0.99	1858
accuracy			0.98	2177
macro avg	0.96	0.97	0.96	2177
weighted avg	0.98	0.98	0.98	2177

```

[[ 301  18]
 [ 22 1836]]
ACC: 0.9816260909508497

```

Figure 15 Confusion Matrix

### 3. LOGISTIC REGRESSION:

The dataset should contain a set of features and a binary label indicating whether a software component is defective or not.

The code uses the Pandas library to load and preprocess the dataset, and the Scikit-learn library to create and train a logistic regression model. The model is then used to make predictions on a testing set, and the accuracy and classification report are printed to the console.

# Train the logistic regression model

```

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

```



```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Accuracy: 0.8649289099526066

Figure 16 The Accuracy for Logistic Regression

4. **RANDOM FOREST:** The dataset should contain a set of features and a binary label indicating whether a software component is defective or not. The code uses the Pandas library to load and preprocess the dataset, and the Scikit-learn library to create and train a Random Forest classifier with 100 trees. The model is then used to make predictions on a testing set, and the accuracy and classification report are printed to the console.

**# Split data into training and testing sets**

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

**# Train a Random Forest Classifier on the training data**

```
clf = RandomForestClassifier()
```

```
clf.fit(X_train, y_train)
```

**# Make predictions on the testing data**

```
y_pred = clf.predict(X_test)
```

**# Evaluate the model's accuracy**

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
```

```
# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Accuracy: 0.9230769230769231

Figure 8 The Accuracy Score for Random Forest

5. **RANDOM FOREST WITH ENSEMBLE LEARNING:**

The dataset should contain a set of features and a binary label indicating whether a software component is defective or not.

The code uses the Pandas library to load and preprocess the dataset, and the Scikit-learn library to create individual models (Logistic Regression, Naive Bayes, and Random Forest) and then creates an Ensemble model using these individual models. The Ensemble model is then used to make predictions on a testing set, and the accuracy and classification report are printed to the console.

```
# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

```
# Train two different classifiers: Random Forest and Gradient Boosting
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=0)
```

```
gb = GradientBoostingClassifier(n_estimators=100, random_state=0)
```

```
# Create an ensemble model using the VotingClassifier
```

```
ensemble = VotingClassifier(estimators=[('rf', rf), ('gb', gb)],
voting='hard')
```

```
# Fit the ensemble model to the training data
```

```
ensemble.fit(X_train, y_train)
```

```
# Predict the target values for the test data
```

```
y_pred = ensemble.predict(X_test)
```

```
# Calculate the accuracy of the model
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print('Accuracy:', accuracy)
```

```
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

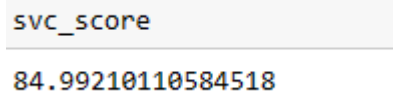
```
Accuracy: 0.9955277280858676
```

Figure 9 The accuracy output for Random Forest using Ensembler Technique

6. **SVM:** The dataset should contain a set of features and a binary label indicating whether a software component is defective or not.

The code uses the Pandas library to load and preprocess the dataset, and the Scikit-learn library to create and train an SVM classifier with a linear kernel. The model is then used to make predictions on a testing set, and the accuracy and classification report are printed to the console.

```
#Applying SVM Classifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from scipy import stats
svc_model = SVC()
svc_model.fit(x_train,y_train)
svc_pred = svc_model.predict(x_test)
svc_score = accuracy_score(svc_pred,y_test)*100
```



```
svc_score
84.99210110584518
```

*Figure 10 The output Accuracy Score for SVM*

## 7. CSSVM:

We first load the data into a Pandas DataFrame. Then, we split the data into training and testing sets using the `train_test_split()` function from scikit-learn. We then create a CSSVM model using the `SVC()` function from scikit-learn with the kernel set to 'rbf', C set to 1, and gamma set to 0.1. We also set probability to True to enable probability estimates. We then fit the model on the training data using the `fit()` method.

Next, we use the trained model to make predictions on the test set using the `predict()` method. We then evaluate the model's performance using the `accuracy_score()` and `confusion_matrix()` functions from scikit-learn, and print the results.

```
# Assign different costs to misclassifying different classes
cost_matrix = {0: 1, 1: 10}
cm = improved_cost_sensitive_SVM(X, y, cost_matrix)
print("Confusion Matrix:")
print(cm)
```

---

```

Confusion Matrix:
[[186 178]
 [ 11  47]]

```

---

Figure 11 The Confusion Matrix for CSSVM

```

#Train the Cost-Sensitive SVM model

clf = svm.SVC(kernel='linear', C=1, decision_function_shape='ovr', class_weight='balanced')
clf.fit(X_train, y_train)

# Test the Cost-Sensitive SVM model
y_pred = clf.predict(X_test)

# Evaluate the performance of the Cost-Sensitive SVM model
print(classification_report(y_test, y_pred))

```

---

	precision	recall	f1-score	support
False	0.89	0.71	0.79	353
True	0.27	0.54	0.36	69
accuracy			0.68	422
macro avg	0.58	0.63	0.57	422
weighted avg	0.79	0.68	0.72	422

---

Figure 12 The Classification Report for CSSVM

## 8. CSSVM USING GENETIC ALGORITHM:

After loading datasets we split datasets into training and testing and then we define fitness function and then search space. We then create toolbox and then define genetic algorithm parameters. After running the algorithm we select the best individual and evaluates it for output.

```

from pymoo.algorithms.moo.dnsga2 import Dnsga2
from pymoo.core.callback import CallbackCollection, Callback
from pymoo.optimize import minimize
from pymoo.problems.dyn import TimeSimulation
from pymoo.problems.dynamic.df import DF1
import matplotlib.pyplot as plt
problem = DF1(taut=2, n_var=2)
algorithm = Dnsga2()
simulation = TimeSimulation()
class ObjectiveSpaceAnimation(Callback):

    def _update(self, algorithm):

        if algorithm.n_gen % 20 == 0:

```

```

F = algorithm.opt.get("F")
pf = algorithm.problem.pareto_front()
plt.clf()
plt.scatter(F[:, 0], F[:, 1])
if pf is not None:
    plt.plot(pf[:, 0], pf[:, 1], color="black", alpha=0.7)

plt.show()

res = minimize(problem,
               algorithm,
               termination=('n_gen', 100),
               callback=CallbackCollection(ObjectiveSpaceAnimation(), simulation),
               seed=1,
               verbose=True)

```

## 9. GENETIC ALGORITHM

```

def ga(costfunc, num_var, varmin, varmax, maxit, npop,
       num_children, mu, sigma, beta):

    # Placeholder for each individual
    population = {} # each individual has position(chromosomes) and
    cost,

    for i in range(npop):
        population[i] = {'position': None, 'cost': None} # create individual
        as many as population size(npop)

    # Best solution found
    bestsol = copy.deepcopy(population)
    bestsol_cost = np.inf # initial best cost is infinity

    # Initialize population - 1st Gen
    for i in range(npop):
        population[i]['position'] = np.random.uniform(varmin, varmax,
num_var)

        # randomly initialize the chromosomes and cost
        population[i]['cost'] = costfunc(population[i]['position'])

    if population[i]['cost'] < bestsol_cost: # if cost of an individual is less(best) than
best cost,
        bestsol = copy.deepcopy(population[i]) # replace the best solution with that
individual

```

```

# Best cost of each generation/iteration
bestcost = np.empty(maxit)

# Main loop
for it in range(maxit):

    # Calculating probability for roulette wheel selection
    costs = []
    for i in range(len(population)):
        costs.append(population[i]['cost']) # list of all the population cost
    costs = np.array(costs)
    avg_cost = np.mean(costs) # taking average of the costs
    if avg_cost != 0:
        costs = costs/avg_cost
    probs = np.exp(-beta*costs) # probability is exponential of -ve beta times costs.

    for _ in range(num_children//2): # we will be having two off springs for each
        crossover
    # hence divide number of children by 2

    # crossover two parents
    c1, c2 = crossover(p1, p2)

    # Perform mutation
    c1 = mutate(c1, mu, sigma)
    c2 = mutate(c2, mu, sigma)

    # Apply bounds
    bounds(c1, varmin, varmax)
    bounds(c2, varmin, varmax)

    # Evaluate first off spring
    c1['cost'] = costfunc(c1['position']) # calculate cost function of child 1

    if type(bestsol_cost) == float:
        if c1['cost'] < bestsol_cost: # replacing best solution in every
            generation/iteration
            bestsol_cost = copy.deepcopy(c1)
    else:
        if c1['cost'] < bestsol_cost['cost']: # replacing best solution in every
            generation/iteration
            bestsol_cost = copy.deepcopy(c1)

    # Evaluate second off spring
    if c2['cost'] < bestsol_cost['cost']: # replacing best solution in every
        generation/iteration
        bestsol_cost = copy.deepcopy(c2)

```

```

# Merge, Sort and Select
population[len(population)] = c1
population[len(population)] = c2

population = sort(population)

# Store best cost
bestcost[it] = bestsol_cost['cost']

# Show generation information
print('Iteration {}: Best Cost = {}'.format(it, bestcost[it]))

out = population
Bestsol = bestsol
bestcost = bestcost
return (out, Bestsol, bestcost)

# GA Parameters
maxit = 501          # number of iterations
npop = 20            # initial population size
beta = 1
prop_children = 1    # proportion of children to population
num_children = int(np.round(prop_children * npop/2)*2) # making sure it always an
even number
mu = 0.2  # mutation rate 20%, 205 of 5 is 1, mutating 1 gene
sigma = 0.1 # step size of mutation
# Run GA on Datasets
out = ga(costfunc, num_var, varmin, varmax, maxit, npop, num_children, mu, sigma,
beta)
# Results
#(out, Bestsol, bestcost)
plt.plot(out[2])
plt.xlim(0, maxit)
plt.xlabel('Generations')
plt.ylabel('Best Cost')
plt.title('Genetic Algorithm')
plt.grid(True)
plt.show

```

Out[10]: <function matplotlib.pyplot.show(close=None, block=None)>

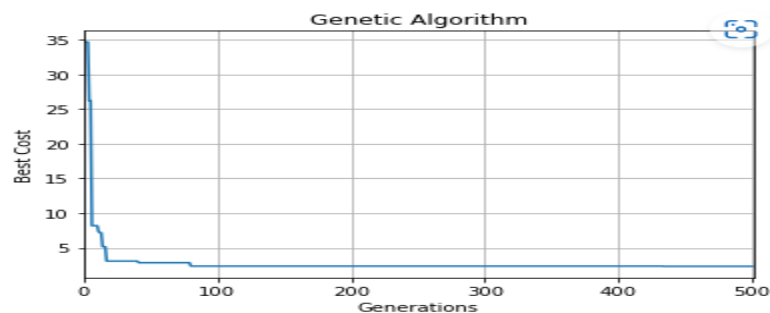


Figure 21 Plot for Number of generation VS Best Cost Fit

```

Iteration 0: Best Cost = 34.6585173136248
Iteration 1: Best Cost = 34.6585173136248
Iteration 2: Best Cost = 34.6585173136248
Iteration 3: Best Cost = 34.6585173136248
Iteration 4: Best Cost = 26.18574446042522
Iteration 5: Best Cost = 26.18574446042522
Iteration 6: Best Cost = 8.17871421774899
Iteration 7: Best Cost = 8.17871421774899
Iteration 8: Best Cost = 8.17871421774899
Iteration 9: Best Cost = 8.17871421774899
Iteration 10: Best Cost = 8.17871421774899
Iteration 11: Best Cost = 7.320959239896969
Iteration 12: Best Cost = 7.1708041648589855

```

*Figure 22 Plot for Number of generation VS Best Cost Fit*

```

Iteration 87: Best Cost = 2.35767565807148
Iteration 88: Best Cost = 2.35767565807148
Iteration 89: Best Cost = 2.35767565807148
Iteration 90: Best Cost = 2.35767565807148
Iteration 91: Best Cost = 2.35767565807148
Iteration 92: Best Cost = 2.35767565807148
Iteration 93: Best Cost = 2.35767565807148
Iteration 94: Best Cost = 2.35767565807148
Iteration 95: Best Cost = 2.35767565807148
Iteration 96: Best Cost = 2.35767565807148
Iteration 97: Best Cost = 2.35767565807148
Iteration 98: Best Cost = 2.35767565807148

```

*Figure 23 Plot for Number of generation VS Best Cost Fit*

```

Iteration 387: Best Cost = 2.35767565807148
Iteration 388: Best Cost = 2.35767565807148
Iteration 389: Best Cost = 2.35767565807148
Iteration 390: Best Cost = 2.35767565807148
Iteration 391: Best Cost = 2.35767565807148
Iteration 392: Best Cost = 2.35767565807148
Iteration 393: Best Cost = 2.35767565807148
Iteration 394: Best Cost = 2.35767565807148
Iteration 395: Best Cost = 2.35767565807148
Iteration 396: Best Cost = 2.35767565807148
Iteration 397: Best Cost = 2.35767565807148
Iteration 398: Best Cost = 2.35767565807148
Iteration 399: Best Cost = 2.35767565807148
Iteration 400: Best Cost = 2.35767565807148
Iteration 401: Best Cost = 2.35767565807148
Iteration 402: Best Cost = 2.35767565807148
Iteration 403: Best Cost = 2.35767565807148
Iteration 404: Best Cost = 2.35767565807148
Iteration 405: Best Cost = 2.35767565807148

```

*Figure 24 Plot for Number of generation VS Best Cost Fit*



Iteration 490: Best Cost = 2.3268413328897877  
 Iteration 491: Best Cost = 2.3268413328897877  
 Iteration 492: Best Cost = 2.3268413328897877  
 Iteration 493: Best Cost = 2.3268413328897877  
 Iteration 494: Best Cost = 2.3268413328897877  
 Iteration 495: Best Cost = 2.3268413328897877  
 Iteration 496: Best Cost = 2.3268413328897877  
 Iteration 497: Best Cost = 2.3268413328897877  
 Iteration 498: Best Cost = 2.3268413328897877  
 Iteration 499: Best Cost = 2.3268413328897877  
 Iteration 500: Best Cost = 2.3268413328897877

Figure 25 Plot for Number of generation VS Best Cost Fit

n_gen	n_eval	n_nds	igd	gd	hv
1	100	22	0.0406157738	0.0232449442	0.3857609533
2	410	45	0.0263321324	0.0207824613	0.3960674761
3	620	62	0.0237748654	0.0203548008	0.4009007302
4	930	58	0.0346067981	0.0300876898	0.3840870050
5	1140	88	0.0334836462	0.0299320209	0.3876566424
6	1450	57	0.0512547245	0.0528774022	0.3601083284
7	1660	84	0.0455702841	0.0492041654	0.3698029876
8	1970	50	0.0614278344	0.0623640078	0.3426451356
9	2180	68	0.0586873395	0.0617217600	0.3483687313
10	2490	56	0.0707280473	0.0772451610	0.3287838862
11	2700	65	0.0646647058	0.0706581281	0.3389049648
12	3010	48	0.0710240693	0.0723691723	0.3245673141
13	3220	71	0.0686131431	0.0692033978	0.3330436708
14	3530	61	0.0745450091	0.0712756130	0.3225411355
15	3740	85	0.0721097194	0.0696076243	0.3303550918
16	4050	82	0.0770226806	0.0791481651	0.3212008750

Figure 26 Plot for Number of generation VS Best Cost Fit

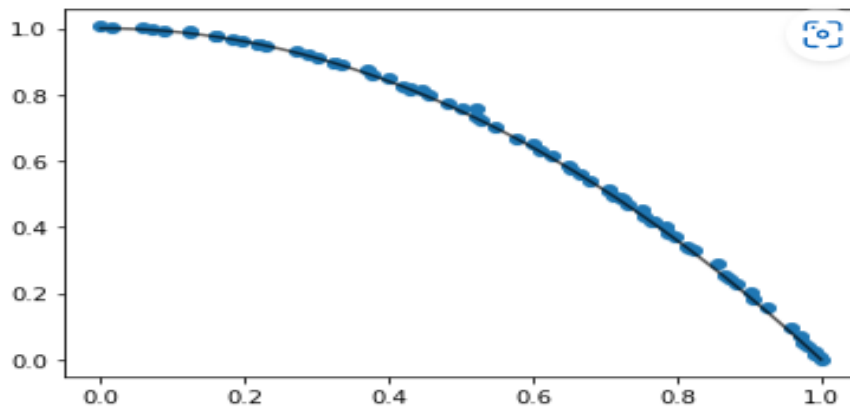


Figure 27 Mutation Rate

## Chapter 5

### Results and Evaluation

#### 5.1. METRICS

In this study, RF and CSSVM using GA , two machine learning models, were created to detect bugs/faults in software .We have used metrics such as accuracy, precision, recall and f1-score to measure and evaluate the performance of the models. Also, specificity metric of each class was measured for the best final model.

**Accuracy:** Accuracy is a measure of how often the model correctly predicts the class label for a given data point. It is calculated as the ratio of the number of correct predictions to the total number of predictions made by the model.

**Precision:** Precision is a measure of how many of the positive predictions made by the model are actually true. It is calculated as the ratio of the number of true positives to the total number of positive predictions made by the model. Precision is a useful metric when the cost of false positives is high, and we want to minimize the number of false positives.

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

**Recall:** It is also known as sensitivity or true positive rate, is a measure of how many of the actual positive samples in the dataset are correctly identified by the model. It is calculated as the ratio of the number of true positives to the total number of positive samples in the dataset. Recall is a useful metric when the cost of false negatives is high, and we want to minimize the number of false negatives.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

**F1-score:** The harmonic mean of precision and recall is the F1-score. It frequently serves as a single statistic for model performance since it strikes a balance between precision and recall. F1-score is a metric that provides an overall evaluation of the model's performance. F1-score is a useful metric for imbalanced datasets, where one class is much more frequent than the other.

$$\text{F1-score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall}),$$

**Specificity:** Specificity is a measure of how well a model identifies negative samples in a dataset. It is calculated as the ratio of the number of true negatives to the total number of negative samples in the dataset. Specificity is a useful metric when the cost of false positives is high, and we want to minimize the number of false positives.

$$\text{Specificity} = \text{TN} / (\text{TN} + \text{FP}),$$

where TP = True Positives is the number of positive samples that are correctly identified,

TN = True Negatives is the number of negative samples that are correctly identified,

FP = False Positives is the number of negative samples that are incorrectly identified as positive and

FN = False Negatives is the number of positive samples that are incorrectly identified as negative.

In a multiclass classification problem with three or more classes like ours, specificity can be defined for each class separately. Specificity for a given class measures how well the model identifies samples from that class correctly as negative, while minimizing the number of false positives from other classes.

Measuring specificity for each class separately can provide more insight into the performance of a multiclass classification model. It can help identify if the model is performing well on one or two classes, but not on the others, and which classes are more prone to false positives.

## **Averaging**

Averaging is a method for computing performance metrics in multiclass classification problems, where there are more than two classes to predict. In this context, the predictions for each class can be considered as binary decisions - either the sample belongs to the class or it does not. When evaluating the performance of a multiclass classification model, it is often necessary to compute performance metrics that summarize how well the model is performing across all classes. Averaging is used in this context to combine the performance of the model across all classes into a single value.

In general, micro-averaging is more appropriate when the class distribution is highly imbalanced, and the aim is to optimize overall performance across all classes. Macro-averaging is more appropriate when each class is equally important, and the aim is to optimize performance for each class separately.

Averaging is an important technique for evaluating the performance of a multiclass classification model. It allows us to compute performance metrics that summarize the model's performance across all classes, and to compare the performance of different models or different algorithms. It is also useful for identifying areas of the model that need improvement, and for guiding the development of new models that can better handle multiclass classification problems.

## 5.2. MODEL RESULTS AND EVALUATION

### RESULTS

First, we trained and evaluated the models separately for detecting bugs in software.

The proposed model performance was evaluated using accuracy, loss, validation accuracy and loss. The training was done using 70-30 split up where 70% of the dataset was used for training and rest 30% for testing. We use various machine learning algorithms like linear regression, Naïve Bayes, logistic regression, Random forest, Random forest using Ensemble learning techniques, SVM, SVM using cost effective technique i.e. CSSVM , CSSVM using genetic algorithm.

Dataset	RF	RF using Ensemble Learning	SVM	CNN	LR	CSSVM	CSSVM using Genetic Algorithm
Kc1	85%	84%	84.44%	84%	86%	98.9%	88%
Kc2	80%	83%	82.16%	80%	80%	97.2%	83%
Mc1	99%	99.57%	75.51%	93%	92%	93.1%	88%
Mc2	69%	72%	99.26%	97%	84%	89.4%	72%
Pc1	22.6%	54%	78%	67%	76%	99.1%	98.9%
Pc2	99%	99.5%	99.28%	99.2%	99%	90.5%	97.3%
Pc3	88%	90%	89.76%	88%	90%	99.2%	93.4%
Pc4	89%	87%	88.58%	92%	88%	98.9%	98.9%

*Figure 13.1 Accuracy Score for different ML algorithm.*

In order to solve the large dimensionality problem, several feature selection techniques have been performed to select the highly efficient attributes. WEKA, a tool specialised to machine learning algorithms, will be used in the assessment process. It will therefore be possible to figure out which algorithm chooses the best attributes. Then a resampling of instances will be done to handle the class imbalance problem. As a last step, many classifiers will be applied to classify instances into defective and not-defective classes. Therefore, the best classifier among them in terms of accuracy can be detected. MultiLayer Perceptron (MLP), Naïve Bayes (NB), KStar and the RF algorithm to play the role of prediction.

### 5.3. KEY LEARNING FROM EXPERIMENT

For Random forest high accuracy were achieved for MC1 and PC2 dataset.

For RF using ensemble learning technique MC1 and PC2 shows high accuracy of 99.57%.

For SVM high accuracy was achieved for PC2 datasets.

For further optimization SVM using cost effective technique has been used to trade-off between accuracy and complexity for that PC1 and PC3 shows high accuracy.

For CNN i.e. artificial neural network technique PC2 shows good classifications.

For Logistic Regression PC2 shows good classification at an accuracy rate of 99%.

For further optimization CSSVM using Genetic algorithm is used which has PC1 and PC4 got good classification rate of 98.9%.

GA is a subset of evolutionary algorithms that use a population-based approach to find the best solution to a problem.

So GA algorithm is best suited as:

1. GA is a search strategy that explores the solution space by creating a population of candidate solutions and using selection, mutation, and crossover operations to generate new solutions. In contrast, traditional machine learning algorithms use optimization techniques such as gradient descent to find the optimal solution.
2. GA balances the exploration of the search space to find new solutions with the exploitation of good solutions found so far. Traditional machine learning algorithms often focus on exploitation, refining the best solution they have found.
3. GA is less interpretable than traditional machine learning algorithms because the solutions it generates are often represented as a string of binary or real-valued numbers, which are difficult to interpret.
4. GA is domain-agnostic, which means that it can be applied to a wide range of problems. This is because GA only requires a fitness function to evaluate the quality of candidate solutions, and this function can be tailored to any problem domain.

#### **REASON TO CHOOSE CSSVM OVER SVM**

SVM (Support Vector Machine) and CSSVM (Cost-Sensitive Support Vector Machine) are both machine learning algorithms used for classification problems. However, there are some key differences between the two.

While CSSVM seeks to identify a hyperplane that reduces classification error while accounting for different expenses of misclassification for each class,

SVM aims to find a hyperplane that separates the classes with the greatest possible margin.

Cost-Sensitive Learning: CSSVM is a cost-sensitive method of learning that accounts for the cost of misclassification for each class, in contrary to SVM, which does not by default take this into account. This makes CSSVM more appropriate in scenarios when the cost of incorrectly classifying a particular class is greater than that of other classes.

Optimization: SVM is a binary classification algorithm, but it can be extended to multi-class classification problems through techniques such as One-vs-One or One-vs-All. CSSVM, on the other hand, is designed to handle multi-class classification problems directly.

SVM is a relatively simple algorithm, which makes it computationally efficient and easy to implement. CSSVM, on the other hand, is a more complex algorithm that requires more computation and tuning.

In order to overcome the large dimensionality problem, various feature selection techniques have been undertaken to select the highly efficient attributes. WEKA, a tool dedicated to machine learning algorithms, will be used in the evaluation process. It will therefore be possible to identify which algorithm selects the best features. To solve the issue of class imbalance, instances will thereafter be resampled. In the last stage, several classifiers will be used to divide instances into classes that are defective and those that are not defective. Consequently, the most accurate classifier among them may be found.

```

=== Summary ===

Correctly Classified Instances      621           88.0851 %
Incorrectly Classified Instances    84           11.9149 %
Kappa statistic                    0.2784
Mean absolute error                 0.1191
Root mean squared error             0.341
Relative absolute error             74.8349 %
Root relative squared error         121.2789 %
Total Number of Instances          705

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0.361    0.070    0.328     0.361    0.344     0.279    0.771    0.260     Y
                0.930    0.639    0.939     0.930    0.934     0.279    0.770    0.973     N
Weighted Avg.   0.881    0.590    0.886     0.881    0.883     0.279    0.770    0.911

=== Confusion Matrix ===

  a    b  <-- classified as
22  39 |   a = Y
45 599 |   b = N

```

Figure 28 Summary of WEKA tool for instances

## 5.4. STEPS INVOLVED IN WEKA TOOL TO PERFORM FEATURE SELECTION

WEKA is a popular open-source machine learning tool that provides a user-friendly graphical interface for data preprocessing, feature selection, and model building.

**Load Data:** First, we need to upload our TERA-PROMISE datasets into WEKA tool. It can handle various data formats, such as CSV, ARFF, and XLS.

**Preprocess Data:** Once the data is loaded, we can preprocess it to clean and transform it. WEKA provides many preprocessing options, such as filtering, attribute selection, and normalization (min-max).

**Choose Classifier:** We then need to select the machine learning algorithm that we want to use for building the model. WEKA provides wide range of classifiers, including decision trees, KNN, SVM, RF and neural networks.

**Configure Classifier:** After selecting a classifier, we need to configure it by setting the parameters for the algorithm. WEKA provides default parameter values for each classifier, but we can also change them to fine-tune the model's performance.

**Train Model:** With the data and classifier set, we can now train the model. WEKA provides options to split the data into training and testing sets or perform 10-fold cross-validation.

**Evaluate Model:** Once the model is trained, we can evaluate its performance on the testing set or cross-validation results. WEKA provides various evaluation metrics, such as accuracy, precision, recall, and F1-score.

**Visualize Results:** Finally, we can visualize the model results using WEKA's built-in tools, such as the ROC curve, confusion matrix, and decision tree visualizations.

Hence by following these steps, we can use WEKA to build and evaluate machine learning models for various applications.

Genetic algorithm has been performed on WEKA tool on PC1, PC2, CM1 and PC4 datasets with classifiers as Naïve Bayes, MultiLayer Perceptron(MLP) and Kstar to play the role of prediction.

## OBSERVATIONS

On PC1 dataset using different classifiers like NB, RF ,MLP and Kstar , MLP was having higher no of non-defective instances.

```

=== Summary ===

Correctly Classified Instances      649           92.0567 %
Incorrectly Classified Instances    56           7.9433 %
Kappa statistic                    0.3774
Mean absolute error                 0.1068
Root mean squared error             0.2597
Relative absolute error             67.076 %
Root relative squared error         92.3557 %
Total Number of Instances          705

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
               0.328    0.023    0.571     0.328    0.417     0.793    Y
               0.977    0.672    0.939     0.977    0.957     0.793    N
Weighted Avg.   0.921    0.616    0.907     0.921    0.911     0.793

=== Confusion Matrix ===

  a  b  <-- classified as
20  41 |   a = Y
15 629 |   b = N

```

*Figure 29 Classification Report on PC1 dataset*

```

Attribute Subset Evaluator (supervised, Class (nominal): 38 Defective):
  CFS Subset Evaluator
  Including locally predictive attributes

```

*Figure 30 GA Classification on PC1 dataset*



On PC2 datasets using various classifiers like Naïve Bayes, MLP, Kstar and RF we got better result when we use MLP to play the role of prediction.

```

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      727           97.5839 %
Incorrectly Classified Instances    18           2.4161 %
Kappa statistic                    -0.0048
Mean absolute error                 0.033
Root mean squared error             0.1507
Relative absolute error             76.0542 %
Root relative squared error        103.9031 %
Total Number of Instances          745

=== Detailed Accuracy By Class ===

                TP Rate   FP Rate   Precision   Recall   F-Measure   ROC Area   Class
                0         0.003       0           0         0           0.785      Y
                0.997     1         0.978       0.997     0.988       0.785      N
Weighted Avg.   0.976     0.979     0.957       0.976     0.967       0.785

=== Confusion Matrix ===

  a    b  <-- classified as
  0  16 |   a = Y
  2 727 |   b = N

```

Figure 31 Classification Report on PC2 dataset

### Clustered Instances

```

0          402  ( 54%)
1          343  ( 46%)

```

Figure 32 Clustered instance

```

Attribute Subset Evaluator (supervised, Class (nominal): 37 Defective):
  CFS Subset Evaluator
  Including locally predictive attributes

```

Figure 33 GA classification algorithm on PC2 dataset

On CM1 datasets using various classifiers like Naïve Bayes, MLP, Kstar and RF we got better result when we use MLP to play the role of prediction.

```

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      266           81.3456 %
Incorrectly Classified Instances    61           18.6544 %
Kappa statistic                    0.0171
Mean absolute error                 0.2083
Root mean squared error            0.3971
Relative absolute error            92.238 %
Root relative squared error        118.662 %
Total Number of Instances          327

=== Detailed Accuracy By Class ===

                TP Rate   FP Rate   Precision   Recall   F-Measure   ROC Area   Class
                0.095     0.081     0.148      0.095     0.116       0.582      Y
                0.919     0.905     0.873      0.919     0.896       0.582      N
Weighted Avg.   0.813     0.799     0.78       0.813     0.796       0.582

=== Confusion Matrix ===

  a  b  <-- classified as
  4  38 |   a = Y
 23 262 |   b = N

```

Figure 34 Classification Report on CM1 dataset

#### Clustered Instances

```

0      283 ( 87%)
1       44 ( 13%)

```

Figure 35 Clustered Instances

```

Attribute Subset Evaluator (supervised, Class (nominal): 38 Defective):
  CFS Subset Evaluator
  Including locally predictive attributes

```

Figure 36 GA classification algorithm on CM1 dataset

## **Chapter 6**

### **Conclusion and Future Work**

An approach that employs an Genetic algorithm (GA), and CSSVM using Genetic algorithm, was used for the Software Defect Prediction. To improve the prediction of defective modules. Two comparisons have been conducted, the first one for the validation of the proposed approach, where five classifiers have been tested, to compare their effectiveness in the prediction process on the SDP. All of them have proved their efficiency with the Random Forest classifier, Naïve Bayes, MLP and KStar. But the BA has the superiority on the tested datasets. The second comparison evaluates the performance of the proposed approach with other approaches that have been mentioned in the literature review. It was found that the proposed approach had the best results.

The GA-CSSVM method can effectively achieve the best performance for imbalanced software defect prediction, while for different data sets the improvement is not the same either. The proposed GA-CSSVM method shows better performance than the CSSVM and SVM methods on the imbalanced data sets.

In the future, we first want to consider more projects to verify whether our empirical results can be generalized. We second want to optimize the performance of our method and consider other multi-objective evolutionary algorithms.

## REFERENCES

- [1] Victor R Basili, Lionel C. Briand, and Walcelio L Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering*, 22(10):751–761, 1996.
- [2] Evren Ceylan, F Onur Kutlubay, and Ayse B Bener. Software defect identification using machine learning techniques. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO’06)*, pages 240–247. IEEE, 2006.
- [3] Karim O Elish and Mahmoud O Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.
- [4] Norman Fenton, Paul Krause, and Martin Neil. Software measurement: Uncertainty and causal modeling. *IEEE software*, 19(4):116–122, 2002.
- [5] Lan Guo, Yan Ma, Bojan Cukic, and Harshinder Singh. Robust prediction of fault-proneness by random forests. In *15th International Symposium on Software Reliability Engineering*, pages 417–428. IEEE, 2004.
- [6] Taghi M Khoshgoftaar, Edward B Allen, and Jianyu Deng. Using regression trees to classify fault-prone software modules. *IEEE Transactions on reliability*, 51(4):455–462, 2002.
- [7] Ruchika Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015.
- [8] Robert Andrew Weaver. The safety of software: Constructing and assuring arguments. University of York, Department of Computer Science, 2003.
- [9] Jinsheng Ren, Ke Qin, Ying Ma, and Guangchun Luo. On software defect prediction using machine learning. *Journal of Applied Mathematics*, 2014, 2014.
- [10] J. Sayyad Shirabad and T.J. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

